Efficient and Robust Out-Of-Distribution Vector Similarity Search with Cross-Distribution Monotonic Graph

Qiang Yue
Hangzhou Dianzi University
yq@hdu.edu.cn

Mengzhao Wang
Zhejiang University
wmzssy@zju.edu.cn

Xiaoliang Xu
Hangzhou Dianzi University
xxl@hdu.edu.cn

Cheng Long
Nanyang Technological University
c.long@ntu.edu.sg

Jiahui Wang
Hangzhou Dianzi University
jiahuiwang@hdu.edu.cn

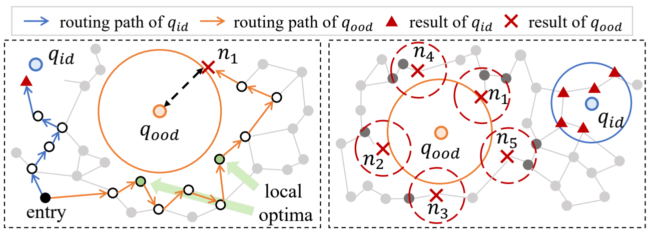
ABSTRACT

Vector similarity search is a key component in many artificial intelligence (AI) applications. While graph-based methods represent the state-of-the-art for vector similarity search, existing graph indexes often suffer from poor navigability and clusterability in Out-Of-Distribution (OOD) scenarios (e.g., database and queries are sourced from different modalities). Recent studies attempt to mitigate this issue by projecting and integrating query-derived auxiliary index structures, but these approaches remain largely heuristic and lack theoretical grounding. In this work, we propose Cross-Distribution Monotonic Graph (CDMG), a novel graph index designed to inherently support navigability and clusterability for OOD queries. CDMG introduces a fusion strategy that bridges the distribution gap between database and query vectors, ensuring a key property—cross-distribution monotonicity—that guarantees monotonic search paths for OOD queries on graph indexes. Theoretical analysis demonstrates that CDMG achieves optimal search time complexity in OOD scenarios compared to existing graph indexes. Furthermore, we introduce CDMG+, a practical variant of CDMG that improves construction efficiency and search robustness by optimizing distance computation, query sampling, and graph structure. Extensive empirical evaluations demonstrate that our techniques outperforms state-of-the-art methods, achieving up to a 3.6× speedup at a 95% recall rate across all real-world OOD datasets.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Lsyhprum/CDMG.

1 INTRODUCTION

The growing prevalence of multimedia content, encompassing images, text, and audio, has intensified the need for effective multimedia data management solutions [20, 50, 74]. A widely adopted approach involves embedding such multimedia content into high-dimensional space as *vectors* using neural networks [30, 32, 59], followed by employing *vector similarity search* [13, 14, 70, 83] to retrieve relevant content. Specifically, given a collection of database vectors and a query vector, an index is constructed based on the database vectors. Vector similarity search then efficiently retrieves the k nearest neighbors of the query from the database vectors using the prebuilt vector index. Recent studies and industrial systems have demonstrated that *graph-based vector indexes* achieve state-of-the-art performance in balancing search efficiency and accuracy [2, 7, 34, 56, 64, 71]. In these graph indexes, each vertex represents a database vector, and each edge establishes a similarity relationship between corresponding vectors. The search process over a graph index employs a greedy routing strategy, guiding vertex traversal to rapidly approach the nearest neighbors of the query. Owing to their *navigability* for search processes [10, 31, 44] and *clusterability* for search results [11, 26, 66], graph-based indexes have become dominant in vector similarity search algorithms and are extensively integrated into mainstream vector databases [20, 22, 48, 67, 71].



(a) Navigability Issue (b) Clusterability Issue

Figure 1: The navigability and clusterability of graph indexes.

When designing graph indexes, it is typically assumed that both query and database data originate from single-modal sources [8, 9, 33, 40, 52, 62]. In such cases, query vectors follow the same distribution as database vectors in high-dimensional space, referred to as In-Distribution (ID) queries (e.g., q_{id} in Figure 1). This assumption is a critical prerequisite for the effectiveness of current graph-based methods in addressing the vector similarity search problem [18, 44, 49]. With the rise of multi-modal learning models, particularly multi-modal large language models (LLMs), multi-modal retrieval has become a fundamental component of many emerging artificial intelligence (AI) applications, such as retrieval-augmented generation (RAG) [1, 36, 41, 79]. In this context, query and database data are derived from multi-modal sources. Due to the inherent *modality gap* [35, 38, 55], query vectors often exhibit a different distribution from database vectors, even when they share the same representation space [68, 86]. Such queries are termed Out-Of-Distribution (OOD) queries (e.g., q_{ood} in Figure 1). While current graph indexes perform well for ID queries, OOD queries face significant performance bottlenecks. For instance, on the Text-to-Image dataset [3], OOD queries using HNSW [44]—a state-of-the-art graph index—experience a 7× increase in search latency compared to ID queries to achieve the same search accuracy. Efficiently and effectively handling OOD queries on graph indexes remains a widely recognized challenge in the field [7, 26, 64].

We identify that the performance bottleneck of OOD queries stems from the disruption of two key properties—navigability and clusterability—that are essential for ensuring high search efficiency and accuracy in the graph indexes.

- **Navigability Issue.** Current graph indexes rely on the assumption that queries are located near database vectors, enabling the greedy routing to converge efficiently toward the nearest neighbors [17, 49]. However, OOD queries deviate significantly from database vectors, violating this assumption. For instance, on the WebVid dataset [4], OOD queries are positioned up to 9× farther from database vectors compared to ID queries. As illustrated in Figure 1(a), while the search path for the ID query q_{id} is short and direct, the greedy routing for the OOD query q_{ood} oscillates unpredictably, as the graph index directs the search toward local optima rather than true nearest neighbors.
- **Clusterability Issue.** Clusterability in graph indexes assumes that query results are closely grouped, allowing the greedy routing to accurately locate all nearest neighbors once one is identified [66]. However, the nearest neighbors of OOD queries tend to be widely dispersed and scattered across the graph index. For example, on WebVid, distances between neighbors of OOD queries are more than 2× larger than those of ID queries. As shown in Figure 1(b), the five results (n_1 to n_5) for the OOD query q_{ood} exhibit poor clusterability, with the neighbors of each result not overlapping with the retrieved results. In contrast, the nearest neighbors of the ID query q_{id} demonstrate strong clusterability.

To address the challenges posed by OOD queries on graph indexes, current solutions enhance existing graph indexes by incorporating the distribution information of OOD queries during index construction. Two state-of-the-art methods are particularly noteworthy: RobustVamana [26] and RoarGraph [7]. RobustVamana improves navigability and clusterability by adding edges among the nearest neighbors associated with OOD queries. By this optimization, RobustVamana reduces query latency by 15% compared to its original counterpart while maintaining the same search accuracy on the Text-to-Image dataset. Similar to RobustVamana, RoarGraph refines the edge selection for the nearest neighbors linked to OOD queries in the database vectors, thereby further enhancing navigability and clusterability. Consequently, RoarGraph achieves a 33% reduction in query latency compared to RobustVamana under the same search accuracy on the Text-to-Image dataset.

Despite recent advancements, the search performance of OOD queries still lags significantly behind that of ID queries on graph-based indexes. For instance, even with the leading RoarGraph, the query latency for OOD queries remains 5.2× higher than for ID queries on the Text-to-Image dataset. We argue that these optimizations fail to fundamentally resolve the navigability and clusterability problems of OOD queries. Specifically, they refine the neighbor relationships for only a small subset of database vectors (i.e., vertices) using a sampled query set (typically 10% of the database [7]), leaving the *majority of vertices unoptimized* (referred to as unoptimized vertices). These unoptimized vertices employ the same candidate acquisition and neighbor selection strategies as the original graph indexes (see Figure 4), which are oriented toward ID queries. Consequently, OOD queries are forced to traverse numerous unnecessary vertices on the graph index, guided by these unoptimized vertices. Furthermore, if a nearest neighbor of an OOD query lies within these unoptimized vertices, the query cannot efficiently locate other nearest neighbors through it. Our evaluation reveals that even with RoarGraph, 57% of vertices remain unoptimized on the Text-to-Image dataset[1]. Therefore, existing optimizations only partially mitigate the navigability and clusterability issues in graph indexes. Additionally, current optimization methods are *heuristic in nature and lack theoretical guarantees for search efficiency and accuracy*, limiting their applicability in real-world systems.

Our Solution and Contributions. This paper presents a novel graph index tailored for OOD queries. We first demonstrate that by reasonably bridging the database and query distributions, OOD greedy routing can be guided to follow the ID routing behavior. We then introduce a key property, cross-distribution monotonicity (**§3.1**), which ensures that the search path from a start vertex to the nearest neighbor of an OOD query remains monotonic on the graph indexes. Analogous to the established monotonicity property for ID queries [18, 49], cross-distribution monotonicity serves as a theoretical basis for achieving high search performance in OOD settings. Building upon this theoretical framework, we design the Cross-Distribution Monotonic Graph (CDMG) (**§3.2**), which integrates OOD query information into database vectors and connects neighbors under both distributions for each vertex via a new edge selection algorithm. Our theoretical analysis proves that CDMG achieves optimal OOD search complexity compared to existing methods. To enhance construction efficiency and search robustness, we present CDMG+, a practical variant of CDMG that optimizes distance calculations, query sampling, and graph structure (**§3.3**). Our key contributions are highlighted as follows:

- We identify the key fundamental theoretical limitation of graph-based methods when handling OOD queries. (**§2.2**)
- We formalize cross-distribution monotonicity, which ensures monotonic routing of OOD queries on graph indexes. (**§3.1**)
- We propose CDMG, an index characterized by cross-distribution monotonicity, and prove that greedy search under this property achieves an optimal complexity for OOD queries. (**§3.2**)
- We propose CDMG+, which enhances construction efficiency and search robustness through a series of simple yet effective optimization techniques. (**§3.3**)
- Extensive evaluations conducted on real-world OOD datasets clearly demonstrate the superiority of our proposed method. (**§4**)

Additionally, **§6** reviews related work, §**5** discusses potential future research directions, and **§6** concludes the paper.

2 PRELIMINARIES AND PROBLEM

2.1 Preliminaries

Definition 1 (**Vector Similarity Search** [13, 14, 70, 83]). *Given a database $\mathcal{X}$ and a query vector q, let $\delta(\cdot,\cdot)$ represent the similarity metric between two vectors. Vector similarity search aims to retrieve the k results $\mathcal{R}(q) = \{n_1, \ldots, n_k\}$ for the query vector q. Each result $n_i \in \mathcal{R}(q)$ satisfies $\delta(n_i, q) \leq \delta(x_j, q)$ for all $x_j \in \mathcal{X} \setminus \mathcal{R}(q)$.*

Achieving exact vector similarity search in a large database is generally impractical due to computational and storage constraints

[1]The percentage of vertices retaining identical edges was calculated between two RoarGraph versions: one built with and the other without queries. Both follow the best parameters based on official guidelines [7].

Table 1: Frequently used notations

Notation	Description
$\mathbb{R}^D$	D-dimensional space.
$\mathcal{X}, x_i \in \mathcal{X}$	The database and the database vector.
$\mathcal{Q}, q_i \in \mathcal{Q}$	The query set and the query vector.
n_i	The i-th nearest neighbor of a vector.
$\delta(\cdot,\cdot)$	The similarity metric between two vectors.
Δ_i	The average distance from each database vector in $\mathcal{X}$ to its i-th nearest neighbor.
$G(V, E)$	A graph index G where the set of vertices and edges are V and E, respectively.
$N_{out}(x_i)$	The out-neighbors of database vector x_i in G.

[19, 27]. Therefore, existing methods focus on finding approximate results to balance efficiency and accuracy while minimizing overhead [23, 63, 84, 85]. We adopt the recall rate *Recall@k* to measure the quality of the k results for q. The *Recall@k* is defined as follows:

$$Recall@k = \frac{|\mathcal{R}(q) \cap \mathcal{G}(q)|}{k} \quad , \tag{1}$$

where $\mathcal{G}$ represents the k nearest neighbors of the query vector q.

A rich line of vector similarity search methods has been developed, with graph-based methods demonstrating particular promise. We introduce the graph-based vector similarity search as follows:

Graph-Based Vector Similarity Search. Graph-based vector similarity search consists of two processes: construction and search.

In the construction process, a graph index [37, 47, 69, 80] is constructed from the database $\mathcal{X}$. Specifically, given a database $\mathcal{X}$ and a predefined edge selection criterion, the graph index is defined as $G = (V, E)$. The vertices V correspond to the vectors in $\mathcal{X}$, and an edge $(x_i, x_j) \in E$ is established between two close vectors x_i and x_j based on the predefined edge selection criterion. In the search process, graph-based vector similarity search utilizes a greedy routing algorithm on the graph index G to identify the top-k results for a query q [2, 6, 39, 49, 57, 76]. Starting from a random or pre-selected entry vector e, the greedy routing iteratively converges toward vectors that are closer to the query q. At each iteration, the closest unvisited vector from the candidate set is selected as the next hop, and its out-neighbors are added to the candidate pool. The process terminates when all candidates have been visited, and the top-k candidates are returned as the final results.

Out-of-Distribution Query. Existing graph indexes are fundamentally designed under the assumption that the database $\mathcal{X}$ and the query vector q originate from the same underlying distribution [5, 7, 26]. However, with the rise of complex AI systems [22, 24, 73], recent developments have increasingly focused on generating vectors learned from multiple modalities to tackle cross-modal tasks. In such scenarios, queries often significantly deviate from the distribution of the database. Given a database $\mathcal{X}$, where the vectors are independently and identically distributed (i.i.d.) from a distribution P_{id}, a query q is considered out-of-distribution if it is drawn i.i.d. from a distribution P_{ood} that differs from P_{id}, i.e., $P_{ood} \neq P_{id}$.

Current vector similarity search methods lack rigorous quantification of OOD queries [26]. A common empirical approach involves

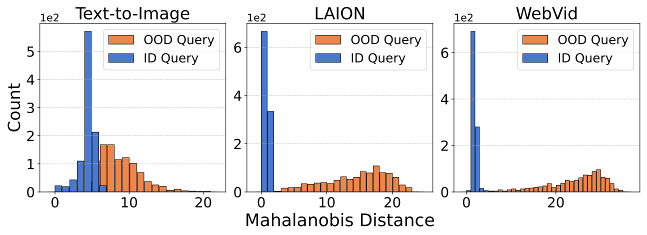


Figure 2: Mahalanobis distances of OOD and ID queries.

using the Mahalanobis distance [42] histogram for evaluation. The Mahalanobis distance $d_M(q, P_{id})$ quantifies the deviation of a query q from the database distribution P_{id}. As depicted in Figure 2, we present the Mahalanobis distances of OOD queries and ID queries relative to the database for three representative real datasets: Text-to-Image [3], LAION [53], and WebVid [4]. The histograms demonstrate that the Mahalanobis distances for OOD queries exhibit significant deviations compared to those of ID queries.

2.2 Problem Analysis

Based on the preceding definitions, this paper investigates the problem of designing a graph-based method specifically optimized for OOD queries. Recent graph-based methods are predominantly derived from the Monotonic Relative Neighborhood Graph (MRNG) [18], which ensures both navigability and clusterability. We first provide a detailed introduction to MRNG and its variants, followed by an analysis of their inherent limitations. Finally, we highlight the shortcomings of recent OOD-oriented graph-based methods.

MRNG and Its Variants. As shown in Figure 4 (the black arrow), the construction of MRNG involves two important phases [71]: *candidate acquisition* and *neighbor selection*, which are designed to enable clusterability and navigability.

Clusterability of MRNG. Clusterability refers to the likelihood that neighbors of a neighbor are also neighbors [11]. When the greedy routing identifies a result, the clusterability of graph indexes enables the rapid retrieval of all nearest neighbors [66, 75]. The clusterability of a graph index can be quantified as the ratio of mutual neighbors within the k nearest neighbors of a query q. Higher clusterability increases the probability that the nearest neighbors of q are also neighbors of each other. To achieve high clusterability, MRNG and its variants [18, 49] acquire candidates using an approximate k-Nearest Neighbor Graph (KNNG) (e.g., KGraph [11] and NSW [43]), which is constructed through either an iterative or incremental process [68]. The iterative process refines neighbors by iteratively updating connections, while the incremental process refines neighbors through a greedy search. In both cases, each vertex is linked to vertices that are closer to it in the final KNNG.

Navigability of MRNG. The concept of navigability originates from early research on the small-world phenomenon [31, 45]. A graph index is considered navigable if the greedy routing can always find a monotonic path between any two vertices [81]. While a complete graph is navigable, it is inefficient [10]. The goal of MRNG and its variants is to construct a sparser graph that maintains monotonic routing [18, 77]. However, they assume that the query is a vertex within the graph index [18, 87], which is often impractical in real-world scenarios. Recent advancements, including NSSG [18],

Table 2: Comparison of $\delta(q, n_1)$ and Δ_i on OOD/ID datasets[2]

Dataset ↓	$\delta(q, n_1)$	Δ_1	Δ_{20}	Δ_{40}	Δ_{60}	Δ_{80}	Δ_{100}
WebVid	0.719	0.109	0.176	0.190	0.199	0.206	0.211
LAION	0.695	0.247	0.338	0.366	0.375	0.385	0.392
MNIST	0.261	<u>0.253</u>	0.419	0.471	0.505	0.532	0.554
DEEP	0.575	<u>0.574</u>	0.831	0.907	0.958	0.997	1.028

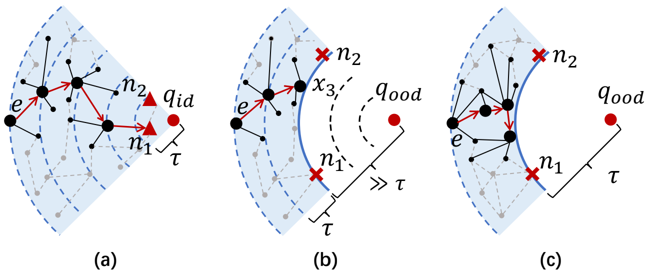


Figure 3: (a) illustrates the greedy routing for an ID query q_{id} over MRNG. (b) and (c) compare the greedy routing for an OOD query q_{ood} under different τ settings.

Vamana [27], and τ-MNG [49], relax this limitation but still operate under the assumption that the query and database share an identical distribution [16, 17, 78, 83]. For instance, given an approximate KNNG, the state-of-the-art τ-MNG employs RobustPruning (described in Algorithm 1) to select the out-neighbors $N_{out}(x_i)$ of x_i based on a query relaxation parameter τ in two steps: (1) x_i directly links to x_j if $\delta(x_i, x_j) \leq 3\tau$ (line 9); (2) If $\delta(x_i, x_j) > 3\tau$, x_i links to x_j if and only if no existing neighbor $x^* \in N_{out}(x_i)$ occludes the edge from x_i to x_j (line 14). As illustrated in Figure 3(a), if $\delta(q_{id}, n_1) \leq \tau$, the greedy routing over τ-MNG moves at least τ closer to the query q_{id} in each iteration [49], ultimately guaranteeing the discovery of the nearest neighbors (n_1 and n_2).

Limitations of MRNG. For an ID query, the query results tend to be mutual neighbors, and the query is close to its nearest neighbor. Thus, when constructing graph indexes for ID queries, MRNG can leverage KNNG to approximate the neighbor relationships among query results, and an appropriate query relaxation parameter τ can be readily determined to satisfy monotonicity. In contrast, for an OOD query: (1) *The query results are widely dispersed*, and the misalignment between KNNG and the actual query results fundamentally violates the principle of MRNG. (2) Additionally, *the query itself is often distant from its nearest neighbor*, further complicating the process. The underlying cause can be illustrated as follows.

If the query relaxation parameter remains small, the condition $\delta(q_{ood}, n_1) < \tau$ is rarely satisfied. This impedes the greedy routing from converging along a specific direction, trapping it in a local minimum [49]. Let $B(q, r)$ denote a ball of radius r centered at q, and $R(q, r_1, r_2)$ represent a ring within $B(q, r_2)$ but outside $B(q, r_1)$. As shown in Figure 3(a), given an ID query q_{id} and an entry $e \in R(q_{id}, 4\tau, 5\tau)$, the greedy routing on the graph index identifies the next hop in $R(q_{id}, 3\tau, 4\tau)$, progressing until it reaches the nearest neighbors (n_1 and n_2). In contrast, for an OOD query q_{ood} (Figure 3(b)), all vectors in $R(q_{ood}, 3\tau, 4\tau)$ may be considered potential nearest neighbors due to the disparity between q_{ood} and the

[2] $\delta(q, n_1)$ denotes the distance between a query q (either an ID query q_{id} or an OOD query q_{ood}) and its nearest neighbor n_1. Cosine similarity measures $\delta(q, n_1)$ and Δ_i for WebVid and LAION, whereas Euclidean distance is used for MNIST and DEEP.

Algorithm 1 RobustPruning(x_i, $\mathcal{V}$, τ, o)

```
1:  **Input:** vector $x_i$, candidate set $\mathcal{V}$, relaxation parameter $\tau > 0$,
    threshold of out-degree $o > 0$
2:  **Output:** out-neighbors $N_{out}(x_i)$
3:  Heap $\leftarrow (\mathcal{V} \cup N_{out}(x_i)) \setminus \{x_i\}$;
4:  $N_{out}(x_i) \leftarrow \emptyset$;
5:  **while** Heap $\neq \emptyset$ and out-degree $|N_{out}(x_i)| < o$ **do**
6:      $x_j \leftarrow$ closest vector to $x_i$ in Heap;
7:      Heap $\leftarrow$ Heap$\setminus\{x_j\}$;
8:      // Step 1. Directly link if within relaxation radius
9:      **if** $\delta(x_i, x_j) \leq 3\tau$ **then**
10:         $N_{out}(x_i) \leftarrow N_{out}(x_i) \cup \{x_j\}$;
11:         **continue**;
12:     **end if**
13:     // Step 2. Ensure the new link is not occluded
14:     **if** $\forall x^* \in N_{out(x_i)}$ s.t. $\delta(x_j, x^*) + 3\tau > \delta(x_i, x_j)$ **then**
15:         $N_{out}(x_i) \leftarrow N_{out}(x_i) \cup \{x_j\}$;
16:     **end if**
17: **end while**
18: **return** $N_{out}(x_i)$
```

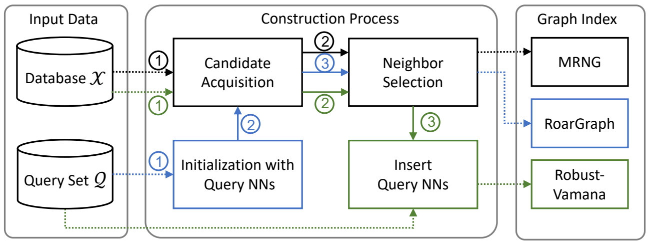


Figure 4: Overview of construction pipeline for ID-oriented graph-based methods (MRNG) and OOD-oriented graph-based methods (RobustVamana and RoarGraph).

database. As a result, the greedy routing may converge to a false result x_3, necessitating backtracking and traversal of numerous unnecessary vertices to locate the true nearest neighbors.

When the query relaxation parameter is set to $\delta(q_{ood}, n_1)$, it substantially exceeds the average distance between database vectors and their respective nearest neighbors, resulting in the condition $\delta(x_i, x_j) \leq 3\tau$ is always met. As Table 2 shows, unlike ID datasets (MNIST and DEEP), where $\delta(q_{id}, n_1)$ closely approximates Δ_1, OOD datasets (WebVid and LAION) exhibit $\delta(q_{ood}, n_1)$ values far exceeding Δ_{100}. Note that 100 exceeds the typical out-degree setting for τ-MNG, which has an expected out-degree of $O(\ln n)$ (e.g., for a billion-scale database, the expected out-degree of τ-MNG is only 21). In this case, RobustPruning selects out-neighbors solely based on $\delta(x_i, x_j)$, where x_j with smaller $\delta(x_i, x_j)$ is preferred. Thus, when τ is determined by $\delta(q_{ood}, n_1)$, the τ-MNG degenerates into a KNNG, which lacks navigability and has proven inefficient in prior work [18, 68]. Figure 3(c) illustrates how an excessively large τ still leads to prolonged search paths and local optima for OOD queries.

Limitations of OOD-oriented Methods. As illustrated in Figure 4, recent advancements have focused on optimizing graph indexes for OOD queries by incorporating query distribution information during their construction. For instance, RobustVamana extends an ID-oriented MRNG variant (i.e., Vamana) and utilizes the nearest

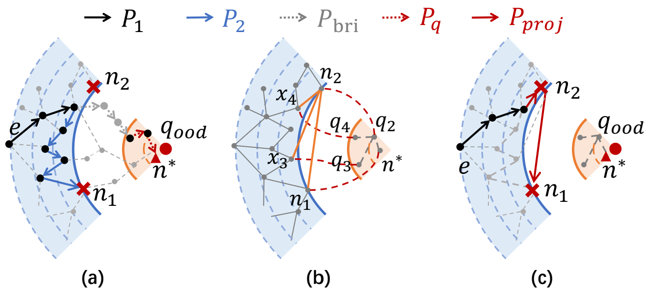


Figure 5: (a) illustrates the difference in routing behavior between ID and OOD queries. (b) and (c) depict the construction and search process for cross-distribution monotonic path.

neighbors of corresponding queries to fill vertices with available space in their neighbor lists. Similarly, RoarGraph directly initializes part of the graph index from a query-derived nearest neighbor subgraph (i.e., a projected graph [7]). However, these methods *continue to rely on ID-oriented candidate acquisition and neighbor selection*, which do not align with the inherent characteristics of OOD queries [26]. Specifically, during candidate generation, most candidates are selected based on the distances between database vectors, which fail to capture the underlying cluster structure of the query results. Additionally, the edge occlusion rule employed for neighbor selection becomes invalid when $\delta(q, n_1)$ significantly exceeds the average distance between database vectors and their nearest neighbors (i.e., Δ_1 in Table 2), leading to inefficient navigation.

In summary, the limited performance of existing solutions stems from their reliance on the ID paradigm. This motivates us to (1) *generalize MRNG theory for OOD queries*, and (2) *optimize candidate acquisition and neighbor selection within the OOD context to enhance navigability and clusterability on graph indexes.*

3 THE PROPOSED CDMG

3.1 Cross-Distribution Monotonicity

In this section, we investigate the following question: *Can OOD queries perform efficient monotonic search on graph indexes in a manner analogous to the routing behavior in ID queries?* To address this, we analyze the differences in routing behavior between ID and OOD queries on MRNG.

In ID scenarios, the alignment between the distributions of the database and query vectors ensures a monotonic search path on MRNG: even when initiated from a distant entry point, greedy routing can efficiently locate the query's nearest neighbor by following a monotonic path [18, 49]. In contrast, the search paths induced by OOD queries on MRNG lack monotonicity due to distributional discrepancies. Specifically, the routing process for an OOD query on MRNG unfolds in two distinct phases: (1) $\underline{\text{In the first phase}}$, while the visited vertices remain distant from the distribution boundary, greedy routing follows a monotonic path and progressively converges toward the OOD query. (2) $\underline{\text{In the second phase}}$, as the search approaches the distribution boundary, convergence toward the OOD query is disrupted due to the distribution gap, leading to a failure in maintaining a monotonic path.

Example 1. *As shown in Figure 5(a), consider a database $\mathcal{X}$ (points in the blue region) and an OOD query set Q (points in the orange region). Let $\mathcal{X}^*$ denote the minimum convex hull that encloses both $\mathcal{X}$ and Q. The bridging set is defined as $\mathcal{B} = \mathcal{X}^* \setminus (\mathcal{X} \cup Q)$, which characterizes the region separating $\mathcal{X}$ and Q. Conceptually, an OOD query q_{ood} w.r.t. $\mathcal{X}$ can be interpreted as an ID query within the extended space $\mathcal{X}^*$. When performing search on the MRNG constructed over $\mathcal{X}^*$, q_{ood} follows a monotonic path $P^* = [P_1, P_{bri}, P_q]$ and finds its nearest neighbor n^* as the result, where P_1, P_{bri}, and P_q consist of points from $\mathcal{X}$, $\mathcal{B}$, and Q, respectively. However, in practical settings, the bridging set $\mathcal{B}$ is not available, leading to erratic oscillations in the second phase of the OOD query search, denoted as P_2.*

To bridge the distribution gap, a natural approach is to establish connections between database and query vectors during graph construction, thereby enabling monotonic search over the query set Q in the second-phase routing. However, this method is inefficient as it requires the graph to explicitly index Q, resulting in elongated routing paths and the inclusion of redundant query vectors in the retrieval results. To overcome this limitation, an intuitive strategy is to incorporate the topological structure induced by the MRNG built on Q into the MRNG constructed on $\mathcal{X}$. Consequently, the two routing phases on the resulting integrated graph index can be characterized as follows: (1) $\underline{\text{In the first phase}}$, greedy routing proceeds along a monotonic path guided by the database distribution toward the distribution boundary, and (2) $\underline{\text{in the second phase}}$, routing continues along a monotonic path shaped by the query distribution until reaching the target query result.

Example 2. *As illustrated in Figure 5(b), each query vector in Q is linked to its nearest database vector (red dashed lines). Additional projected edges (orange lines) can be introduced between database vectors based on the neighborhood relationships of their corresponding query vectors. For example, since q_2 is connected to n^* and q_4, n_2 is linked to n_1 and x_4. During the search process (Figure 5(c)), P_1 and P_{proj} combine to form a new monotonic routing path $P = [P_1, P_{proj}]$ entirely within $\mathcal{X}$. The projected path P_{proj} traces the path P_q, returning n_1 and n_2 as query results because their corresponding query vectors (q_2 and n^*) are closer to q_{ood}. The OOD routing path (i.e., $[P_1, P_2]$) in Figure 5(a) requires seven hops to retrieve a single correct result (n_1), whereas the monotonic routing path P (i.e., $[P_1, P_{proj}]$) retrieves two results (n_1 and n_2) in just four hops.*

Definition and Notation. Based on the preceding analysis, our key insight is to construct a graph index over database vectors that ensures *cross-distribution monotonicity* for OOD queries. We formally define the cross-distribution monotonic path as follows:

Definition 2 (Cross-Distribution Monotonic Path). *For a database $\mathcal{X}$ and a query set Q, each query $q_i \in Q$ is associated with its nearest database vector $x_i \in \mathcal{X}$. Let G be a graph over $\mathcal{X}$, and let the query relaxation parameter $\tau \geq 0$ be a constant. A routing path $P = [x_1, \ldots, x_b, \ldots, x_\ell, n_1]$ on G is a cross-distribution monotonic path for an OOD query q_{ood} if it satisfies: $\forall i = 1, \ldots, b-1$, $\delta(q_{ood}, x_i) - \tau > \delta(q_{ood}, x_{i+1})$, $\forall j = b, \ldots, \ell-1$, $\delta(q_{ood}, q_j) - \tau > \delta(q_{ood}, q_{j+1})$, and $\delta(q_{ood}, q_\ell) > \delta(q_{ood}, n^*)$, where n^* is the nearest query to q_{ood} and n_1 is the nearest database vector of n^*.*

The cross-distribution monotonic path extends the concept of the τ-monotonic path [49] to the OOD scenario. When the query set Q and the database $\mathcal{X}$ follow the same distribution, a monotonic search path under the query distribution is equivalent to

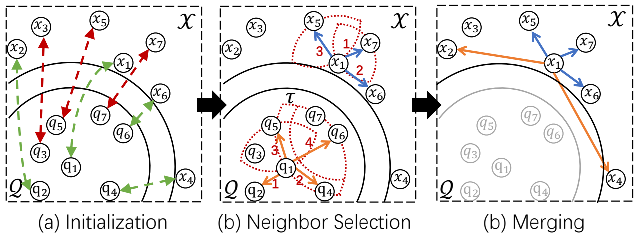


Figure 6: Illustration of the CDMG construction process.

a monotonic search path under the database distribution. In this case, the cross-distribution monotonic path condition reduces to: $\forall i = 1, \ldots, \ell - 1, \delta(q_{ood}, x_i) - \tau > \delta(q_{ood}, x_{i+1})$, which corresponds to the τ-monotonic path in the ID scenario. Building upon this extended concept, we introduce cross-distribution monotonicity:

Definition 3 (**Cross-Distribution Monotonicity**). *Given a database $\mathcal{X}$ and query set $\mathcal{Q}$, let $\tau > 0$ be a constant. A graph G over $\mathcal{X}$ has cross-distribution monotonicity if for any query q_{ood} satisfying $\delta(q_{ood}, n^*) \le \tau$, there exists a cross-distribution monotonic path in G from any entry point to the result n_1. Here, n^* is the nearest query to q_{ood} in $\mathcal{Q}$ and n_1 is the nearest database vector to n^* in $\mathcal{X}$.*

3.2 Cross-Distribution Monotonic Graph

We introduce the Cross-Distribution Monotonic Graph (CDMG), which incorporates OOD query information for each vertex to enhance clusterability while maintaining monotonicity through a unified edge occlusion rule. The formal definition follows:

Definition 4 (**Cross-Distribution Monotonic Graph**). *Given a database $\mathcal{X}$ and query set $\mathcal{Q}$, let $\tau \ge 0$ be a constant. A CDMG is a directed graph G where for any vertices $v_i, v_j \in G$ with corresponding vectors $x_i, x_j \in \mathcal{X}$ and queries $q_i, q_j \in \mathcal{Q}$: (1) If $\delta(q_i, q_j) \le 3\tau$, $(v_i, v_j) \in G$; (2) If $\delta(q_i, q_j) > 3\tau$ and $(v_i, v_j) \notin G$, there exists v^* with $(v_i, v^*) \in G$ such that its corresponding x^* and q^* satisfy: $x^* \in (B(x_i, \delta(x_i, x_j)) \cap B(x_j, \delta(x_i, x_j) - 3\tau))$ and $q^* \in (B(q_i, \delta(q_i, q_j)) \cap B(q_j, \delta(q_i, q_j) - 3\tau))$.*

Lemma 1 establishes the monotonicity guarantee of CDMG:

Lemma 1. *CDMG guarantees cross-distribution monotonicity.*

Proof. If the edge $(v_1, n_1) \in G$, the path $P = [v_1, n_1]$ is a cross-distribution monotonic path. We now consider the case where $(v_1, n_1) \notin G$. Case 1: If $\delta(q_1, n^*) \le 6\tau$, then G must contain a cross-distribution monotonic path $P = [v_1, v_2, n_1]$. This follows because v_1 must have an out-neighbor v_2 such that $q_2 \in B(q_1, \delta(q_1, n^*)) \cap B(n^*, \delta(q_1, n^*) - 3\tau)$. Since $\delta(q_2, n^*) \le 3\tau$, it follows that $(v_2, n_1) \in G$. When $\delta(q_{ood}, n^*) \le \tau$, we have $\delta(q_{ood}, q_2) \le \delta(q_1, n^*) - 2\tau$ and $\delta(q_{ood}, q_1) > \delta(q_1, n^*) - \tau$, which leads to $\delta(q_{ood}, q_2) < \delta(q_{ood}, q_1) - \tau$. Therefore, the path $P = [v_1, v_2, n_1]$ is a cross-distribution monotonic path. Case 2: If $\delta(q_1, n^*) > 6\tau$, then G must contain a cross-distribution monotonic path $P = [v_1, v_2, \ldots, v_b, v_{b+1}, \ldots, v_{\ell-1}, v_\ell, n_1]$. In the first phase, since G is a CDMG, it must satisfy $\delta(x_2, n^*) < \delta(x_1, n^*) - 3\tau$ and $\delta(x_2, q_{ood}) < \delta(x_1, q_{ood}) - \tau$, and at each step, the path moves closer to q_{ood} by at least τ until reaching a vertex v_b. In the second phase, similar to the case above, we have $\delta(q_{b+1}, n^*) < \delta(q_b, n^*) - 3\tau$ and $\delta(q_{ood}, q_{b+1}) < \delta(q_{ood}, q_b) - \tau$. Eventually, the search reaches a vertex $v_{\ell-1}$ such that $\delta(q_{\ell-1}, n^*) < 6\tau$, whose monotonicity has already been established. □

Algorithm 2 BuildBipartiteGraph($\mathcal{X}, \mathcal{Q}$)

```
1:  Input: database $\mathcal{X}$, query set $\mathcal{Q}$
2:  Output: Bipartite graph $G_{bri}$
3:  // Step 1. Query-base assignment
4:  $G_{bri} \leftarrow \emptyset, C \leftarrow \emptyset, \mathcal{A} \leftarrow \mathcal{X}$;
5:  for $\forall q_i \in \mathcal{Q}$ do
6:      $x_i \leftarrow$ closest database vector to $q_i$ in $\mathcal{X}$;
7:      $C \leftarrow C \cup \{\langle x_i, q_i \rangle\}$;
8:  end for
9:  while $C \neq \emptyset$ do
10:     $\langle x_i, q_i \rangle \leftarrow$ pair in $C$ with the largest distance $\delta(x_i, q_i)$;
11:     $G_{bri} \leftarrow G_{bri} \cup (x_i, q_i), C \leftarrow C \setminus \{\langle x_i, q_i \rangle\}$;
12:     $\mathcal{A} \leftarrow \mathcal{A} \setminus \{x_i\}$;          ▷ Mark $x_i$ as occupied
13: end while
14: // Step 2. Assign remaining database vectors
15: for $\forall x_i \in \mathcal{A}$ do
16:     $q_i \leftarrow$ closest query to $x_i$ in $\mathcal{Q}$;
17:     $G_{bri} \leftarrow G_{bri} \cup (x_i, q_i)$;
18: end for
19: return $G_{bri}$
```

Construction Process. As shown in Figure 6, CDMG is constructed through three components: (1) Initialization, (2) Neighbor Selection, and (3) Merging. We detail these components as follows. (1) Initialization. Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$, we construct a 1-regular bipartite graph $G_{bri}(V, U, E_{bri})$ to bridge data from different distributions, where V corresponds to database vectors and U to query vectors. The edge set $E_{bri} \subset V \times U$ forms a perfect match, pairing each $x_i \in V$ with a $q_i \in U$. Specifically, we employ BuildBipartiteGraph (Algorithm 2) to construct G_{bri}. This procedure initially assigns each query to its nearest neighbor in the database (indicated by green arrows), giving priority to queries located near the core of the query distribution. Subsequently, any remaining unassigned database vectors are matched to their nearest queries (indicated by red arrows), ensuring each vertex references two vectors from different distributions. This approach leverages queries that are farther from the database and generates more dispersed results to effectively guide the construction process.

(2) Neighbor Selection. For each vertex $v_i \in G$, CDMG selects out-neighbors from both the database and query set. Despite differences in data distribution, the same edge occlusion rule (Algorithm 1) applies in both cases. To select out-neighbors $N_q(v_i)$ under the query distribution, the remaining query vectors $\mathcal{Q} \setminus \{q_i\}$ are ranked in a candidate list Heap by their distance to q_i. Each candidate $q_j \in$ Heap is evaluated for occlusion by existing out-neighbors: if any $q^* \in N_q(v_i)$ occludes q_j, the edge (q_i, q_j) is discarded; otherwise, q_j is added to $N_q(v_i)$. This process iterates until all candidates in Heap are processed. The out-neighbors $N_x(v_i)$ are selected analogously by applying the same rule to x_i with candidates from $\mathcal{X} \setminus \{x_i\}$.

(3) Merging. Since all elements in $N_q(v_i)$ are query vectors, we replace each query with its corresponding database vector to eliminate the need for indexed queries. These database vectors are then merged with the out-neighbors $N_x(v_i)$ to form the final out-neighbor set $N_{out}(v_i)$. As illustrated in Figure 6, $N_q(v_1)$ contains q_2, q_4, q_5, and q_6, which correspond to x_2, x_4, x_5, and x_6. The resulting $N_{out}(v_1)$ combines neighbors selected from both distributions.

Algorithm 3 BuildCDMG($\mathcal{X}, \mathcal{Q}, \tau, o$)

```
1:  Input: database $\mathcal{X}$, query set $\mathcal{Q}$, relaxation parameter $\tau > 0$,
    threshold of out-degree $o > 0$
2:  Output: CDMG $G$
3:  // Initialization
4:  $G_{bri} \leftarrow$ BuildBipartiteGraph($\mathcal{X}$,$\mathcal{Q}$);            ▷ Algorithm 2
5:  $G \leftarrow \emptyset$
6:  for $\forall x_i \in \mathcal{X}$ do
7:      $q_i \leftarrow G_{bri}(x_i)$;
8:      // Dual neighbor selection;
9:      $N_x(v_i) \leftarrow$ RobustPruning($x_i$,$\mathcal{X} \setminus \{x_i\}$,$\tau$,$o$);   ▷ Algorithm 1
10:     $N_q(v_i) \leftarrow$ RobustPruning($q_i$,$\mathcal{Q} \setminus \{q_i\}$,$\tau$,$o$);
11:     // Merging
12:     $N_{out}(v_i) \leftarrow N_x(v_i) \cup G_{bri}(N_q(v_i))$;
13:     $G \leftarrow G \cup N_{out}(v_i)$;
14: end for
15: return $G$
```

Search Process. The search process of CDMG adopts the standard GreedySearch approach (Algorithm 4), which is commonly utilized in graph-based methods. GreedySearch uses a parameter s to regulate the capacity of the candidate container, referred to as Heap. The search initiates from an entry point e (line 5). In each iteration, GreedySearch selects the vertex x_i that is closest to the query q_{ood} from Heap, and computes the distances between q_{ood} and the out-neighbors of x_i (lines 8–10). A new vertex is inserted into Heap if it is closer to q_{ood} than the current candidates or if Heap has not yet reached its capacity L (lines 12–15). The search terminates when no closer vertex can be identified. Ultimately, the top-k vectors in Heap are returned as the retrieval results.

Complexity Analysis. We provide theoretical analysis of CDMG's graph index size (Lemma 2), construction complexity (Lemma 3), and search complexity (Lemma 4), as detailed below.

Lemma 2. *Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$, let $\tau > 0$ be a constant, the CDMG $G(V, E)$ built on $\mathcal{X}$ and $\mathcal{Q}$ has expected out-degree $O(2 \cdot \ln |V|)$ and expected index size $O(2 \cdot |V| \ln |V|)$.*

Proof. Please see the detailed proof in Appendix A. □

Lemma 3. *Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$, let relaxation parameter $\tau > 0$ be a constant, the construction complexity of CDMG $G(V, E)$ in Algorithm 3 is $O(|V||\mathcal{Q}| + |\mathcal{Q}| \ln |\mathcal{Q}| + 2 \cdot |V|^2 \ln |V|)$.*

Proof. The complexity of Algorithm 3 is derived as follows. Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$: (1) In the initialization phase, the algorithm first traverses each query to identify its nearest neighbor in the database, which requires $O(|V||\mathcal{Q}|)$ time. After collecting all candidate pairs, it selects the pair with the maximum distance in each iteration, an operation that can be performed in $O(|\mathcal{Q}| \ln |\mathcal{Q}|)$ time. In the worst-case scenario, many database vectors remain unassigned and must undergo reverse matching, introducing an additional cost of $O(|V||\mathcal{Q}|)$. (2) In the neighbor selection phase, the RobustPruning procedure has a known time complexity of $O(|V| \ln |V|)$ over $\mathcal{X}$ [49]. As this procedure is executed once for each database vector and its corresponding query, the total cost becomes $O(2 \cdot |V|^2 \ln |V|)$. (3) For the merging phase, since the out-degrees of both $N_x(v_i)$ and $N_q(v_i)$ are bounded by $O(\ln |V|)$, the merging operation incurs a cost of $O(|V| \ln |V|)$.

Algorithm 4 GreedySearch(G, q, k, L)

```
1:  Input: graph index $G$, query $q$, results size $k$, candidates size $L$
2:  Output: top-$k$ results of $q$
3:  $e \leftarrow$ entry vector in $G$;
4:  // Initialize candidates container
5:  Heap $\leftarrow \{e\}$;
6:  while Heap has unvisited vector do
7:      // Find next hop
8:      $x_i \leftarrow$ closest unvisited vector to $q$ in Heap;
9:      mark vector $x_i$ as visited;
10:     $N_{out}(x_i) \leftarrow$ out-neighbors of vector $x_i$ in $G$;
11:     // Expand candidates
12:     Heap $\leftarrow$ Heap $\cup N_{out}(x_i)$;
13:     if |Heap| $> L$ then
14:         resize Heap to retain top-$L$ vectors for $q$;
15:     end if
16: end while
17: return [closest $k$ vectors from Heap]
```

Consequently, the overall construction complexity amounts to $O(|V||\mathcal{Q}|+|\mathcal{Q}| \ln |\mathcal{Q}|+2 \cdot |V|^2 \ln |V|)$, where the constant in last term is retained to reflect the dual invocation of RobustPruning. □

Lemma 4. *Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$ in the space $\mathbb{R}^D$, where D denotes the dimensionality, let τ be a non-negative constant. For an OOD query q_{ood} that follows the same distribution as $\mathcal{Q}$, the expected routing path length ℓ over the CDMG $G(V, E)$ is $O(|V|^{\frac{1}{D}} \ln |V|)$, and the search complexity is $O(|V|^{\frac{1}{D}} (\ln |V|)^2)$.*

Proof. The search complexity of graph-based methods can be expressed as $O(o\ell)$, where o denotes the expected out-degree and ℓ represents the routing path length. As established in Lemma 2, the expected out-degree of CDMG is $O(\ln |V|)$. We further show that the expected routing path length of CDMG is $O(|V|^{\frac{1}{D}} \ln |V|)$. The cross-distribution monotonic path $P = [P_1, P_{proj}]$ for q_{ood} comprises two phases: (1) To estimate the path length of P_1, we consider an ID routing path P^* on the minimal convex hull $\mathcal{X}^*$ that encloses both $\mathcal{X}$ and $\mathcal{Q}$. The points in $\mathcal{X}^*$ are uniformly distributed, with $|\mathcal{X}^*| = \lambda|\mathcal{V}|$, where λ is a constant. Since q_{ood} follows the same distribution as $\mathcal{X}^*$, the length of P^* is $O(|\mathcal{X}^*|^{1/D} \ln |\mathcal{X}^*|)$. However, in practice, points within $B(q_{ood}, \epsilon)$ are not indexed, where ϵ denotes the distance between q_{ood} and its nearest database vector. Therefore, the length of P_1 is bounded by $O\left(|\mathcal{X}^*|^{1/D} \ln |\mathcal{X}^*| - \epsilon/\tau\right)$. (2) In the second phase, it is straightforward to prove that the expected length of P_{proj} is $O(\epsilon/\tau)$. Combining both phases, the total expected routing path length ℓ is $O(|V|^{1/D} \ln |V|)$, and the overall search complexity becomes $O(|V|^{1/D} (\ln |V|)^2)$. □

3.3 Construction of CDMG+

In this section, we propose CDMG+, an *approximate variant* of CDMG, designed to address several practical challenges: (1) CDMG requires a *dual* neighbor selection process, which introduces additional computational overhead and doubles the index size; (2) the construction process is highly *sensitive to the size of the query set*, making it vulnerable to variations in sample quantity; and (3) both the database and the query set are fully treated as candidates, resulting in low construction efficiency. To overcome these issues,

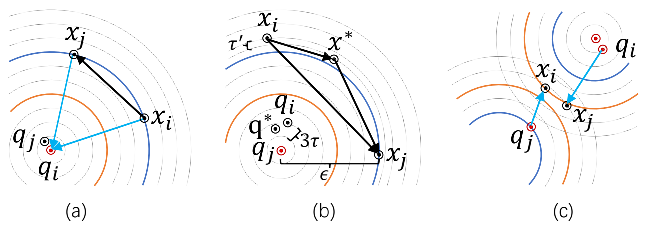


Figure 7: Illustration of three different cases.

we first introduce a unified *cross-distribution distance* to streamline the construction process. We then employ *query aggregation* to enhance the quality of query sampling. Finally, we design the *implementation* of the construction process to improve efficiency.

Cross-Distribution Distance. As stated in §3.2, CDMG necessitates independent neighbor selection for database vectors and their associated queries, owing to *the lack of a unified mechanism for cross-distribution comparison.* To address this limitation, we propose a cross-distribution distance that not only simplifies computation but also maintains clusterability and cross-distribution monotonicity. Specifically, given a 1-regular bipartite graph G_{bri}, in which each database vector is paired with a corresponding query (i.e., q_i with x_i, and q_j with x_j), we formally define the cross-distribution distance $\xi(v_i, v_j)$ between vertices v_i and v_j as follows:

$$\xi(v_i, v_j) = \delta(x_i + q_i, x_j + q_j) \quad . \tag{2}$$

To assess its effectiveness, we decompose $\xi(v_i, v_j)$ under the assumption that δ represents the Euclidean distance, which facilitates insight into its underlying mechanisms. This leads to:

$$\begin{aligned}\xi(v_i, v_j) &= \|x_i + q_i - (x_j + q_j)\|^2 = \|(q_i - x_j) - (q_j - x_i)\|^2 \\ &= \delta(q_i, x_j) + \delta(q_j, x_i) - 2\|q_i - x_j\|\|q_j - x_i\| \cos\theta \quad ,\end{aligned} \tag{3}$$

where θ denotes the angle between the vectors $q_i - x_j$ and $q_j - x_i$.

(1) Asymmetric Distance. For an OOD query q_{ood}, the search process determines whether a database vector x^* qualifies as a result based on the distance $\delta(q_{ood}, x^*)$. Consequently, a natural approach is to use $\delta(q_i, x_j)$ to determine whether v_j should be considered a neighbor of v_i. This choice is consistent with the distance metric employed during the search process. As shown in Figure 7(a), both x_i and x_j lie on the boundary of the database distribution. Although x_i and x_j are spatially distant, their distances $\delta(q_i, x_i)$ and $\delta(q_i, x_j)$ to the query q_i are equal (blue arrowed lines), permitting x_j to be selected as an out-neighbor of x_i (black arrowed lines). However, directly employing $\delta(q_i, x_j)$ as the distance measure overlooks the intrinsic relationships within the database and query distributions, namely, $\delta(q_i, q_j)$ and $\delta(x_i, x_j)$, which leads to two critical issues: (1) it violates the proposed edge occlusion rule, and (2) it fails to account for database vectors that deviate from the query distribution.

(2) Symmetric Distance. The symmetric distance form, $\xi_s(v_i, v_j) = \delta(q_i, x_j) + \delta(q_j, x_i)$, better reflects the mutual relationships between queries and database vectors. Its effectiveness is justified as follows:

First, symmetric distance reformulates CDMG construction while enabling to preserve cross-distribution monotonicity. Specifically, (1) if $\xi_s(v_i, v_j) \le 2\epsilon + 3\tau$, a direct link from v_i to v_j is established; (2) if $\xi_s(v_i, v_j) > 2\epsilon + 3\tau$, v_i links to v_j only if $\xi_s(v_j, v^*) + 3\tau > \xi_s(v_i, v_j)$. The rationale behind this is as follows: (1) When $\delta(q_i, q_j) \le 3\tau$, x_i and x_j are located near the boundary of the database distribution, such that both $\delta(q_i, x_j)$ and $\delta(q_i, x_i)$ are approximately equal to a large constant ϵ. Therefore, using the symmetry distance yields:

$$\delta(q_i, x_j) + \delta(q_j, x_i) \le \epsilon + \epsilon + \delta(q_i, q_j) \le 2\epsilon + 3\tau \quad ; \tag{4}$$

(2) For $\delta(q_i, q_j) \ge 3\tau$, let $\delta(x^*, x_i) < \tau^*$ with $\tau^* \ll \tau$. Given that $\delta(q_i, q_j) - \delta(q^*, q_j) \le 3\tau$, it follows that:

$$\begin{aligned}&\delta(q_i, x_j) + \delta(q_j, x_i) - \delta(q_j, x^*) - \delta(q^*, x_j) \\ &\le (\epsilon + \delta(q_i, q_j)) + (\delta(q_j, x^*) + \tau^*) - \delta(q_j, x^*) - (\epsilon + \delta(q^*, q_j)) \\ &\le \tau^* + \delta(q_i, q_j) - \delta(q^*, q_j) \le \tau^* + 3\tau \quad .\end{aligned} \tag{5}$$

Although symmetric distance preserves cross-distribution monotonicity in the second routing phase, it still has limitations. Specifically, (1) it is worth noting that symmetric distance is effective only when the corresponding queries are approximately co-directional. However, as illustrated in Figure 7(c), when the corresponding queries of x_i and x_j are far apart, v_j may still be mistakenly considered a neighbor of v_i. (2) In addition, it does not adequately handle database vectors located far from the distribution boundary.

(3) Cross-Distribution Distance. Counter-directional cases can be mitigated by incorporating a corrective term: $-2\|q_i - x_j\|\|q_j - x_i\| \cos\theta$. Under this configuration, $\cos\theta$ is negative, and the correction penalizes counter-directional queries pairs while preserving the desired behavior for co-directional pairs. Furthermore, for the databsae vectors that are close to each other yet deviate from the boundary, their corresponding queries are highly similar or even identical. As a result, their cross-distribution distance can be approximate as $\xi(v_i, v_j) \approx \delta(x_i, x_j)$. It is non-trivial to prove that it guarantees both clusterability and navigability.

Query Aggregation. The performance of CDMG improves with the size of the query set. However, in practice, acquiring a sufficiently large and representative query set is challenging [28]. A limited query set results in many database vectors being assigned identical or highly similar query vectors during `BuildBipartiteGraph`, thereby diminishing the effectiveness of query guidance in the construction process. Conversely, a large query set can lead to significant overhead [26]. To alleviate bias stemming from the small size of the query set, we employ a query aggregation strategy that enhances the quality of the query vector associated with each database vector in the bipartite graph. This approach mitigates the influence of outlier or noisy queries by averaging over multiple nearest neighbors. Specifically, for each database vector x_i, we identify its top-S nearest queries, denoted as $\mathrm{NN}(x_i)$ with $|\mathrm{NN}(x_i)| = S$, and compute the mean of these neighbors as the updated query vector: $q_i = \frac{1}{S}\sum_{q^* \in \mathrm{NN}(x_i)} q^*$. This method is both computationally efficient and effective in enhancing the robustness of CDMG+.

Implementation. We outline the construction algorithm BuildCDMG+ as follows. Given a database $\mathcal{X}$ and a query set $\mathcal{Q}$, the algorithm first constructs a 1-regular bipartite graph G_{bri} using the query aggregation strategy, which requires $O(|V||Q|)$ time. Each database vector is then fused with its corresponding query in G_{bri} to form the fused set $\mathcal{F}$, where each vector $f_i \in \mathcal{F}$ is defined as $f_i = x_i + G_{bri}(x_i)$ (lines 5-8 in Algorithm 5). This step takes $O(|V|)$ time and allows for pre-computation of cross-distribution distances, thus reducing search overhead. Using the fused set $\mathcal{F}$, BuildCDMG+ incrementally constructs the graph index. For each

Algorithm 5 BuildCDMG+($\mathcal{X}, Q, S, L, \tau, o$)

```
1: Input: database $\mathcal{X}$, query set $Q$, query aggregation parameter $S$, candidates size $L$, relaxation parameter $\tau$, threshold of out-degree $o$
2: Output: CDMG+ $G$
3: $\mathcal{F} \leftarrow \emptyset, G \leftarrow \emptyset$;
4: // Query Aggregation
5: for $\forall x_i \in \mathcal{X}$ do
6:     $\mathrm{NN}(x_i) \leftarrow$ top-$S$ queries to $x_i$ in $Q$;
7:     $q_i \leftarrow \frac{1}{S}\sum_{q^* \in \mathrm{NN}(x_i)} q^*$; $\mathcal{F} \leftarrow \mathcal{F} \cup \{x_i + q_i\}$;
8: end for
9: for $\forall v_i \in G$ do
10:     // Candidate Acquisition
11:     $\mathcal{V} \leftarrow$ GreedySearch$(G, f_i, L, L)$;          ▷ Algorithm 4
12:     // Neighbor Selection
13:     $N_{out}(v_i) \leftarrow$ RobustPruning$(f_i, \mathcal{V}, \tau, o)$;     ▷ Algorithm 1
14:     $G \leftarrow G \cup N_{out}(v_i)$;
15: end for
16: return $G$
```

vertex v_i, it is treated as a query, and GreedySearch is used to retrieve the top-L approximate nearest neighbors (line 11), avoiding exhaustive comparisons. The search starts from an entry point e and moves toward v_i by comparing distances $\delta(f_i, f^*)$, where f^* denotes the fused vector of a candidate in Heap. The candidate acquisition for each vertex requires $O(|V|^{1/D}(\ln|V|)^2)$ time. After candidate acquisition, a single round of RobustPruning is applied to sparsify the out-degree to $|N_{\text{out}}(v_i)| = o$ while preserving cross-distribution monotonicity (line 13). This neighbor selection takes $O(|V|\ln|V|)$ time. Once all vertices are processed, BuildCDMG+ returns G as the final graph index. The overall construction complexity is $O(|V||Q| + |V|\ln|V|(|V|^{1/D}\ln|V| + |V|))$. Since CDMG+ is designed to operate with a small query set, its construction complexity simplifies to $O(|V|^2\ln|V|)$, which is comparable to that of mainstream MRNG variants such as τ-MNG and NSG [17, 49].

4 EXPERIMENTAL STUDY

In this section, we evaluate our proposed CDMG+ with a comprehensive experimental study to answer the following questions:

- **Q1:** How do CDMG+ and other graph-based methods perform in terms of effectiveness and efficiency for OOD queries? **(§4.2)**
- **Q2:** How does CDMG+ impact the index size and construction time? **(§4.3)**
- **Q3:** To what extent does CDMG+ optimize the navigability and clusterability of the graph index? **(§4.4)**
- **Q4:** How do different strategies contribute to the search performance of CDMG+? **(§4.5)**
- **Q5:** How do parameters and query set size affect CDMG+? **(§4.6)**
- **Q6:** How is the scalability of CDMG?**(§4.7)**

4.1 Experimental Setup

Datasets. We evaluate our proposed method on five modern large-scale OOD datasets [7, 60], as summarized in Table 3. These datasets vary in dimensions, metrics, modality, and OOD degree. Specifically, we use the Wasserstein distance W_2 [29, 65] to measure the distribution difference between the OOD queries Q_{ood} and ID queries Q_{id} w.r.t. the database $\mathcal{X}$. Their ratio $W_2(\mathcal{X}, Q_{ood})/W_2(\mathcal{X}, Q_{id})$ is also shown in Table 3, and we use it to roughly evaluate the OOD degree of dataset. The larger the value, the stronger the OOD degree. The details of each dataset are provided below: Text-to-Image is a cross-domain dataset derived from a visual search engine. The database vectors are image embeddings and the query vectors are textual embeddings. The distance metric used for the ground truth is the inner product (IP). LAION is a widely used text-image dataset. The distance metric using cosine similarity (Cos). WIT is a dataset derived from a text-to-image application. The inner product is used as a metric. WebVid is a caption-video dataset sourced from stock footage sites. The distance metric used is cosine similarity. CC3M is a dataset of image-text pairs designed for image captioning systems. Its ground truth is generated using the inner product.

Table 3: Dataset Statistics

Dataset ↓	Dim	Metric	# Database	# Query	Type	OOD Deg
Text-to-Image [3]	200	IP	1000,000	1,000	Image, Text	1.17
LAION [51]	512	Cos	1000,000	1,000	Image, Text	1.47
WIT [58]	512	IP	1000,000	1,000	Image, Text	1.72
WebVid [4]	512	Cos	1000,000	1,000	Video, Text	2.46
CC3M [54]	256	IP	1000,000	1,000	Image, Text	2.81

Baselines. We compare CDMG+ with several state-of-the-art graph-based methods known for their efficiency in vector similarity search.

- **HNSW** [44]: A well-known hierarchical graph-based method.
- **τ-MNG** [49]: An optimized version of MRNG that incorporates query relaxation to enhance performance.
- **RobustVamana** [26]: A specialized variant of Vamana tailored for handling OOD queries. Unless otherwise specified, the size of the queries is 10% of the database.
- **RoarGraph** [7]: RoarGraph is a recently proposed efficient graph index designed for OOD queries. The size of the queries is the same as in RobustVamana, representing 10% of the database.

All algorithms are obtained from the official source codes, implemented in C++, and compiled using GCC 9.4.0. They were stripped of all relevant prefetching and SIMD optimizations. Our source code and datasets are available at: https://github.com/Lsyhprum/CDMG.

Evaluation metrics. We evaluate the search accuracy using *Recall@k*, which is a widely used metric for vector similarity search [12, 17]. It is defined in Equation 1. We evaluate search efficiency with *QPS* and *NDC* [49, 68], where *QPS* (Queries Per Second) measures the number of handled queries per second, and *NDC* (Number of Distance Computations) quantifies computational cost. Besides, we introduce the number of routing hops (*HOP*) [7] as a supplementary metric to assess the navigability of the graph index. To measure the clusterability of the graph index, we propose a novel metric called Query Nearest Neighbor Quality (*QNNQ*). Given a query q, let $\mathcal{G}(q) = \{g_1, \ldots, g_{k+1}\}$ be its top-$(k+1)$ nearest neighbors in the database, and let $\mathcal{N}(g_1) = \{g_1, n_1 \ldots, n_k\}$ be the top-$(k+1)$ nearest neighbors of g_1. Therefore, the *QNNQ@k* is formally defined as:

$$QNNQ@k = \frac{\mathcal{G}(q) \setminus \{g_1\} \cap \mathcal{N}(g_1) \setminus \{g_1\}}{k} \quad .$$

A higher *QNNQ@k* value indicates better clusterability. For example, if all queries originate from the database, their nearest neighbors

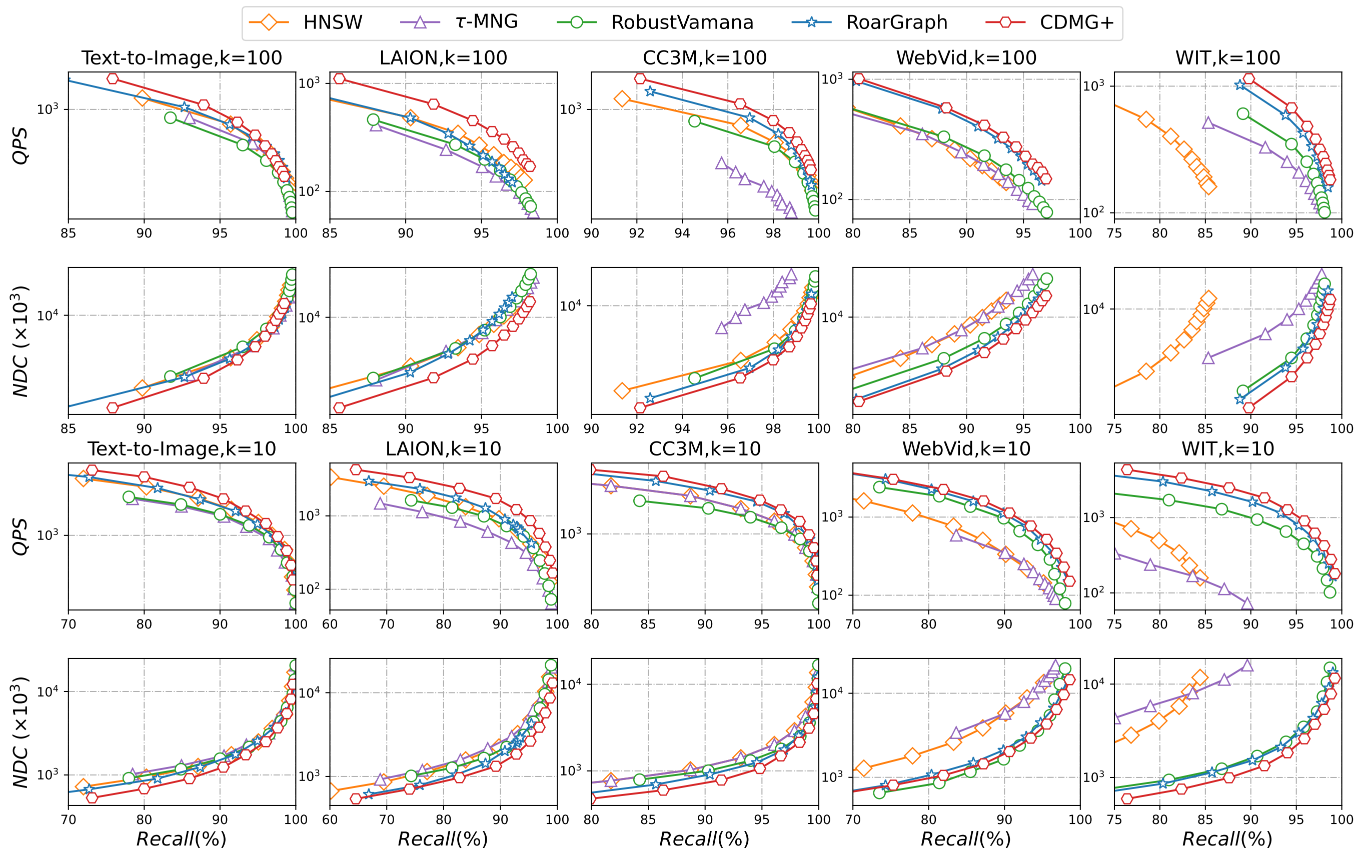


Figure 8: Efficiency (*QPS* and *NDC*) and effectiveness (*Recall@k*) evaluation of all algorithms across five OOD datasets: For the first and third rows, the top-right is better, while for the second and fourth rows, the bottom-right is better.

$\mathcal{G}$ will match $\mathcal{N}$, resulting in *QNNQ@k* = 1. However, when the query differs from the database, *QNNQ@k* < 1.

Implementation setup. All experiments are conducted on an Ubuntu 20.04 server equipped with Intel(R) Xeon(R) Gold 5218 CPU (2.30GHz) and 125GB of memory. The Eigen library [15] enhances matrix operations, while OpenMP enables parallel index construction execution using 20 threads.

4.2 Efficiency and Effectiveness Evaluation

Figure 8 presents the performance comparison of the proposed CDMG+ against state-of-the-art ID-oriented (HNSW and τ-MNG) and OOD-oriented (RobustVamana and RoarGraph) graph-based methods. We evaluate accuracy using *Recall@100* (rows 1 and 2) and *Recall@10* (rows 3 and 4) and measure search efficiency with *QPS* and *NDC*. In the figures illustrating *Recall@k* vs. *QPS* (rows 1 and 3), a curve positioned towards the top-right indicates better performance. Conversely, in those depicting *Recall@k* v.s. *NDC* (rows 2 and 4), a position toward the bottom-right is preferable.

The results demonstrate that our proposed algorithm consistently outperforms existing methods across all datasets. At *Recall@100* = 90%, CDMG+ achieves *QPS* improvements of 1.3×, 1.76×, 1.53×, and 2.2× over baseline on Text-to-Image, LAION, CC3M, and WebVid, respectively. In terms of *NDC*, CDMG+ is 9.3%, 26.8%, 31.7%, and 47.6% *NDC* less than baseline. At Recall@10 = 95%, CDMG+ delivers *QPS* gains of 1.07×, 2.1×, 1.9×, and 3.6×, with corresponding *NDC* reductions of 12.9%, 44.6%, 26%, and 67.7%. Notably, HNSW fails to reach *Recall@100* = 90% and *Recall@10*=95% on WIT dataset.

Among the baseline methods, ID-oriented graph-based methods perform significantly worse than OOD-oriented methods, especially on datasets such as WebVid, WIT, and CC3M, which exhibit the highest OOD degrees (see Table 3). Although RobustVamana and RoarGraph are specifically designed for OOD queries, their improvements remain limited, particularly on Text-to-Image, LAION, and CC3M. This is mainly because they rely on heuristic optimization, often leading to unstable and suboptimal performance. In contrast, CDMG+ exhibits strong adaptability across all OOD datasets. It is also worth noting that while HNSW is originally designed for ID query scenarios, it achieves competitive results on certain datasets with low OOD degree, likely due to its hierarchical structure that facilitates better entry point selection.

4.3 Index Cost

We evaluated the index size and construction time of existing algorithms, with the results shown in Figure 9. We observed that CDMG+ almost has the smallest index size compared to other methods, particularly on Text-to-Image, LAION, and WIT datasets. This allows CDMG+ to better adapt to resource-constrained environments. In contrast, HNSW and τ-MNG incur larger index sizes, this is mainly because HNSW has an additional hierarchical structure for accessing entry points, while τ-MNG retains more

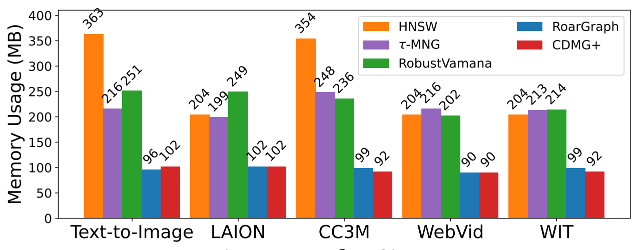


Figure 9: Index Size.

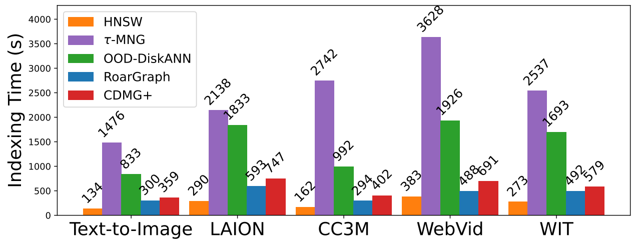


Figure 10: Indexing Time.

edges due to the edge pruning process enabled by query relaxation. The index size of RobustVamana is 2.24×-2.57× larger than that of CDMG+. This is mainly because RobustVamana retains numerous edges from the database distribution and augments them with additional query-derived edges. For example, the average out-degree of RobustVamana increases from 22 to 57 after edge stitching. The results of indexing time are shown in Figure 10, CDMG+ also occupies one of the leading positions, which reduces 56.9%-65.8% indexing time compared to the RobustVamana. Since each vertex in CDMG+ corresponds to two vectors from different distributions, its naive construction incurs a higher overhead than methods like HNSW and RoarGraph. However, with the introduction of the cross-distribution distance and implementation optimizations, CDMG+ achieves only slightly higher construction time while significantly outperforming existing methods in index quality.

4.4 Naviability and Clusterability

In this section, we conduct a fine-grained comparison of two key properties of graph indexes: navigability and clusterability. These properties respectively reflect the difficulty for greedy search to converge to the query's nearest neighbor and the difficulty of retrieving additional relevant results once the nearest neighbor has been found. We conduct a comparison of the navigability and clusterability of CDMG+ against baseline methods, as detailed below:

Navigability. We compare the number of hops *HOP* required by the optimized and original algorithms to achieve the target *Recall@100* on the Text-to-Image, LAION, and CC3M datasets. To ensure a fair comparison, we control the out-degree of the compared algorithms to be approximately the same. Specifically, we select RoarGraph as the baseline algorithm, and the average out-degree of CDMG+ and RoarGraph are 21 compared with 23, 25 compared with 25, and 23 compared with 24 on the three datasets, respectively. As shown in Figure 11, CDMG+ reaches the target *Recall* with fewer hops than RoarGraph. For example, on the LAION dataset, the *HOP* of RoarGraph is 1.25× that of CDMG+ under 95% Recall. This result demonstrates that CDMG+ effectively optimizes the graph structure, enabling faster convergence to query-relevant regions.

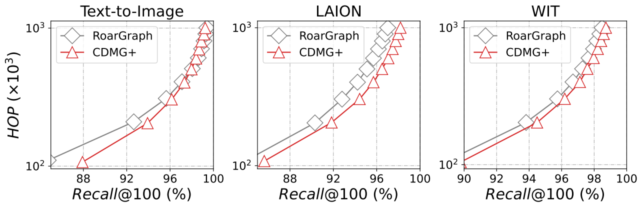


Figure 11: Hop statistics of RoarGraph and CDMG+.

Table 4: Clusterability Statistics (*QNNQ@100*)

Method ↓	Text-to-Image	LAION	CC3M	WEBVID	WIT
ID-ori	0.4568	0.4167	0.5394	0.3929	0.4190
OOD-ori	0.2634	0.2006	0.2744	0.1545	0.2106
Bipartite	0.2607	0.2142	0.3314	0.1692	0.2721
Ours	<u>**0.2919**</u>	<u>**0.2505**</u>	0.3175	<u>**0.1843**</u>	0.2460

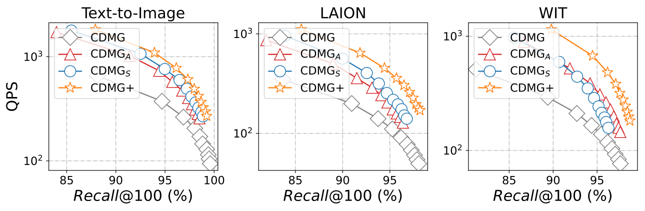


Figure 12: Ablation comparisons.

Clusterability. For clustering quality, we construct KNNG, Bipartite Graph, and a CDMG-optimized KNNG, avoiding the edge-pruning influence and ensuring the neighbors are selected only based on distance. We use *QNNQ@100* to measure the clustering quality of query structures in these graphs. As illustrated in Table 4, in KNNG, the *QNNQ* scores of ID-ori and OOD-ori represent in-distribution and out-of-distribution queries, respectively, with OOD-ori being significantly lower since query nodes do not belong to the database. As a result, *QNNQ@100* < 1. The CDMG-optimized version improves *QNNQ@100* quality across all cases, with respective increases of 11%, 25%, 16%, 19%, and 17% compared to the unoptimized version.

4.5 Ablation Study

We mainly compare the impact of key components in CDMG+, with the trade-offs between *Recall@100* and *QPS* presented in Figure 12. Specifically, we present the performance comparison between CDMG and CDMG+, as well as the differences among CDMG$_A$, CDMG$_S$, and CDMG+, which are constructed using asymmetric, symmetric, and cross-distribution distances, respectively.

The naive CDMG simply merges the out-edges obtained from both distributions without an effective comparison process, resulting in a significantly higher out-degree. For example, on the Text-to-Image dataset, CDMG exhibits a 1.3× increase in out-degree compared to CDMG+. In contrast, CDMG+ adopts cross-distribution distance to jointly compute the edges from both distributions, effectively constraining the out-degree by the parameter o, and thereby reducing the distance computations during the search process.

We compare the impact of three distance computation strategies on search performance. Specifically, for a vertex v_i and its candidate neighbor v_j, we construct CDMG$_A$ using asymmetric distance $\delta(q_i, x_j)$, CDMG$_S$ using symmetric distance $\delta(q_i, x_j) + \delta(q_j, x_i)$,

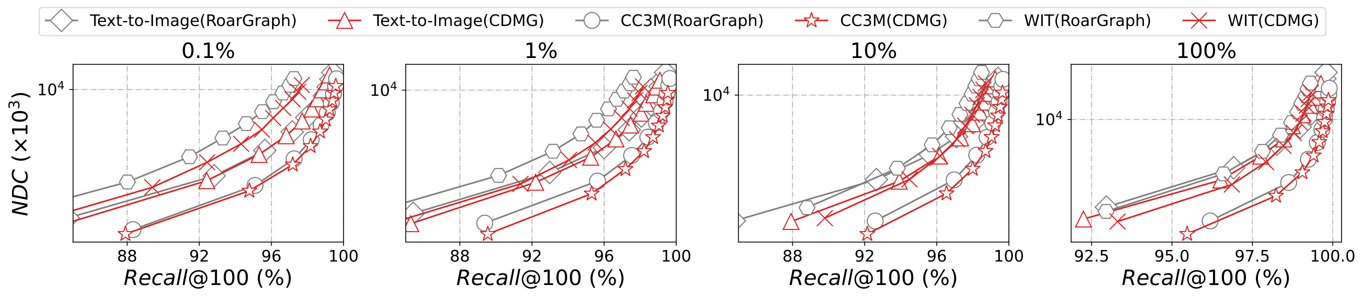


Figure 13: Performance comparison across different query set sizes for index construction.

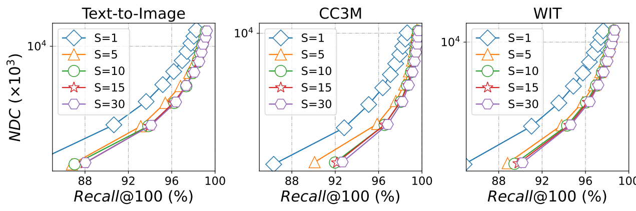


Figure 14: Impacts from different parameter S.

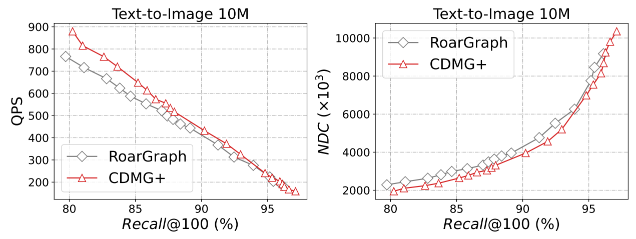


Figure 15: 10M-scale efficiency and effectiveness evaluation.

and CDMG+ using cross-distribution distance $\delta(x_i + q_i, x_j + q_j)$. As shown in Figure 12, asymmetric distance performs the worst on the Text-to-Image and LAION datasets, as it captures only partial clusterability and fails to maintain monotonicity. In comparison, symmetric distance improves performance by better preserving monotonicity. However, on the WIT dataset, it underperforms the asymmetric variant—likely due to its vulnerability in cases where vectors have similar database vectors but queries pointing in different directions. Cross-distribution distance introduces a corrective term and ensures both clusterability and navigability, consistently achieving the best search performance across all datasets.

4.6 Parameter Sensitivity

In this section, we examine the impact of different query set sizes $|Q|$ and the parameter of query aggregation S on performance.

Query set size. We evaluate search performance under varying query set sizes—0.1%, 1%, 10%, and 100% of the database—on three datasets: Text-to-Image, CC3M, and WIT. As shown in Figure 13, we assess the trade-off between *Recall@100* and *NDC*. CDMG+ consistently enhances search performance across all query set sizes and outperforms the baseline, RoarGraph. Using 1% of the dataset strikes a favorable balance between efficiency and accuracy.

Parameter S. We analyze the performance impact of the parameter S in query aggregation. As shown in Figure 14, we compare different settings: S = 1, 5, 10, 15, and 30. Search performance improves as S increases, as averaging vectors helps mitigate the impact of smaller training sets and yields better query representations. S = 15 is a suitable choice across all datasets. However, as S increases, the improvement in search performance gradually diminishes. Compared to S=15, larger S values provide only a slight gain at low recall levels, while the construction cost increases significantly.

4.7 Scalability

We further evaluate the performance of CDMG+ on a larger dataset, Text-to-Image10M. As shown in Figure 15, the left and right plots illustrate the trade-offs between *Recall@100* and *QPS*, and *Recall@100* and *NDC*, respectively. CDMG+ surpasses state-of-the-art methods and demonstrates excellent scalability on this larger dataset.

5 DISCUSSION

Further Analysis of ID Datasets. Although ID queries have been extensively studied, some difficult ones [25, 72] still fall into local optima during search. This stems from the assumption of uniform data distributions in existing methods, which overlook real-world patterns like skewness and long tails. We view such hard queries as an OOD problem relative to the database, and future work will explore optimizations for non-uniform ID scenarios.

Streaming Algorithms. In real-world applications, efficiently supporting dynamic storage in graph indexing remains an open challenge. Specifically, data distributions in the database often evolve over time, leading to OOD effects between pre-deployed databases and new queries [82]. Additionally, when similar data is removed in batches, effectively inserting new nodes while maintaining strong clusterability and navigability is another direction for future work.

Hybrid Optimization. A promising direction for further work is to jointly optimize entry point selection [46], dimensionality reduction [60, 61], quantization [26], and hardware-specific acceleration [21] under OOD scenarios, in conjunction with CDMG+.

6 CONCLUSION

This work presents CDMG+, a theoretically grounded (with CDMG as its theoretical counterpart) and computationally efficient graph index designed for OOD queries. By enforcing cross-distribution monotonicity, CDMG guarantees reliable and efficient search paths even under significant distributional shifts between database and query vectors. Theoretical analysis establishes the optimality of CDMG in search time complexity for OOD scenarios, while the practical refinements in CDMG+ ensure scalable and robust performance. Extensive experiments demonstrate the superiority of our approach, achieving substantial gains in speed and accuracy over existing methods across diverse benchmarks.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *IEEE Data Eng. Bull.* 46, 3 (2023), 89–105.

[3] Artem Babenko and Dmitry Baranchuk. 2021. Text-to-Image Dataset for Billion-Scale Similarity Search. Retrieved August 23, 2023, from https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search.

[4] Max Bain, Arsha Nagrani, Gül Varol, and Andrew Zisserman. 2021. Frozen in time: A joint video and image encoder for end-to-end retrieval. In *Proceedings of the IEEE/CVF international conference on computer vision.* 1728–1738.

[5] Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I Zeki Yalniz. 2023. Dedrift: Robust similarity search under content drift. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 11026–11035.

[6] Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. 2019. Learning to route in similarity graphs. In *International Conference on Machine Learning.* PMLR, 475–484.

[7] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X Sean Wang. 2024. Roargraph: A projected bipartite graph for efficient cross-modal approximate nearest neighbor search. *arXiv preprint arXiv:2408.08933* (2024).

[8] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[9] Yijie Dang, Nan Jiang, Hao Hu, Zhuoxiao Ji, and Wenyin Zhang. 2018. Image classification based on quantum K-Nearest-Neighbor algorithm. *Quantum Information Processing* 17 (2018), 1–18.

[10] Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. 2024. Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits. *arXiv preprint arXiv:2405.18680* (2024).

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

[12] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and code: Fast indexing with graphs and compact regression codes. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 3646–3654.

[13] Karima Echihabi. 2020. High-dimensional vector similarity search: from time series to deep network embeddings. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 2829–2832.

[14] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3198–3201.

[15] Eigen Library. 2009. Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. https://eigen.tuxfamily.org/index.php?title=Main_Page. [Online; accessed 01-June-2024].

[16] Cole Foster, Berk Sevilmis, and Benjamin Kimia. 2025. Generalized relative neighborhood graph (GRNG) for similarity search. *Pattern Recognition Letters* 188 (2025), 103–110.

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

[18] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[19] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[20] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. 2022. Manu: a cloud native vector database management system. *arXiv preprint arXiv:2206.13843* (2022).

[21] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning.* PMLR, 3887–3896.

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

[23] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 5713–5722.

[24] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2553–2561.

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

[26] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. 2022. Ood-diskann: Efficient and scalable graph anns for out-of-distribution queries. *arXiv preprint arXiv:2211.12850* (2022).

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

[28] Taewon Jeong and Heeyoung Kim. 2020. Ood-maml: Meta-learning for few-shot out-of-distribution detection and classification. *Advances in Neural Information Processing Systems* 33 (2020), 3907–3916.

[29] Leonid V Kantorovich. 1960. Mathematical methods of organizing and planning production. *Management science* 6, 4 (1960), 366–422.

[30] Douwe Kiela and Léon Bottou. 2014. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP).* 36–45.

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

[32] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. 2020. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), 2880–2894.

[33] Mr Ramesh Krishnamaneni and AN Murthy. 2021. Advancing Drug Dealing Detection Using Neural Embedding and Nearest Neighbour Searching Techniques. *International Journal on Recent and Innovation Trends in Computing and Communication* 9, 7 (2021), 19–23.

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

[35] Victor Weixin Liang, Yuhui Zhang, Yongchan Kwon, Serena Yeung, and James Y Zou. 2022. Mind the gap: Understanding the modality gap in multi-modal contrastive representation learning. *Advances in Neural Information Processing Systems* 35 (2022), 17612–17625.

[36] Weizhe Lin, Jinghong Chen, Jingbiao Mei, Alexandru Coca, and Bill Byrne. 2023. Fine-grained late-interaction multi-modal retrieval for retrieval augmented visual question answering. *Advances in Neural Information Processing Systems* 36 (2023), 22820–22840.

[37] Jie Liu, Xiao Yan, Xinyan Dai, Zhirong Li, James Cheng, and Ming-Chang Yang. 2020. Understanding and improving proximity graph based maximum inner product search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 139–146.

[38] Yuchen Liu, Junnan Zhu, Jiajun Zhang, and Chengqing Zong. 2020. Bridging the modality gap for speech-to-text translation. *arXiv preprint arXiv:2010.14920* (2020).

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

[40] Junshui Ma, Robert P Sheridan, Andy Liaw, George E Dahl, and Vladimir Svetnik. 2015. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling* 55, 2 (2015), 263–274.

[41] Zi-Ao Ma, Tian Lan, Rong-Cheng Tu, Yong Hu, Heyan Huang, and Xian-Ling Mao. 2024. Multi-modal Retrieval Augmented Multi-modal Generation: A Benchmark, Evaluate Metrics and Strong Baselines. *arXiv preprint arXiv:2411.16365* (2024).

[42] Prasanta Chandra Mahalanobis. 2018. On the generalized distance in statistics. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)* 80 (2018), S1–S7.

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

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

[45] Stanley Milgram. 1967. The small world problem. *Psychology today* 2, 1 (1967), 60–67.

[46] Yutaro Oguri and Yusuke Matsui. 2024. Theoretical and Empirical Analysis of Adaptive Entry Point Selection for Graph-based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2402.04713* (2024).

[47] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE).* IEEE, 4236–4247.

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

[49] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[50] Samira Pouyanfar, Yimin Yang, Shu-Ching Chen, Mei-Ling Shyu, and SS Iyengar. 2018. Multimedia big data analytics: A survey. *ACM computing surveys (CSUR)* 51, 1 (2018), 1–34.

[51] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PmLR, 8748–8763.

[52] Andrey V Savchenko. 2017. Maximum-likelihood approximate nearest neighbor method in real-time image recognition. *Pattern Recognition* 61 (2017), 459–469.

[53] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. 2021. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114* (2021).

[54] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning. In *Proceedings of ACL*.

[55] Peiyang Shi, Michael C Welle, Mårten Björkman, and Danica Kragic. 2023. Towards understanding the modality gap in CLIP. In *ICLR 2023 Workshop on Multimodal Representation Learning: Perks and Pitfalls*.

[56] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. 2022. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*. PMLR, 177–189.

[57] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613* (2021).

[58] Krishna Srinivasan, Karthik Raman, Jiecao Chen, Michael Bendersky, and Marc Najork. 2021. Wit: Wikipedia-based image text dataset for multimodal multilingual machine learning. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 2443–2449.

[59] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1165–1174.

[60] Mariano Tepper, Ishwar Singh Bhati, Cecilia Aguerrebere, Mark Hildebrand, and Ted Willke. 2023. LeanVec: Search your vectors faster by making them fit. *arXiv preprint arXiv:2312.16335* (2023).

[61] Mariano Tepper, Ishwar Singh Bhati, Cecilia Aguerrebere, and Ted Willke. 2024. GleanVec: Accelerating vector search with minimalist nonlinear dimensionality reduction. *arXiv preprint arXiv:2410.22347* (2024).

[62] Sergios Theodoridis and Konstantinos Koutroumbas. 2006. *Pattern recognition*. Elsevier.

[63] Bing Tian, Haikun Liu, Yuhang Tang, Shihai Xiao, Zhuohui Duan, Xiaofei Liao, Xuecang Zhang, Junhua Zhu, and Yu Zhang. 2024. FusionANNS: An Efficient CPU/GPU Cooperative Processing Architecture for Billion-scale Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2409.16576* (2024).

[64] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 39–54.

[65] Leonid Nisonovich Vaserstein. 1969. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii* 5, 3 (1969), 64–72.

[66] Hongya Wang, Zhizheng Wang, Wei Wang, Yingyuan Xiao, Zeng Zhao, and Kaixiang Yang. 2020. A note on graph-based nearest neighbor search. *arXiv preprint arXiv:2012.11083* (2020).

[67] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[68] Kaiye Wang, Qiyue Yin, Wei Wang, Shu Wu, and Liang Wang. 2016. A comprehensive survey on cross-modal retrieval. *arXiv preprint arXiv:1607.06215* (2016).

[69] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2024. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2024).

[70] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.

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

[72] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *arXiv preprint arXiv:2408.13899* (2024).

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

[74] Yi Wu, Edward Y Chang, Kevin Chen-Chuan Chang, and John R Smith. 2004. Optimal multimodal fusion for multimedia data analysis. In *Proceedings of the 12th annual ACM international conference on Multimedia*. 572–579.

[75] Xiaoliang Xu, Mengzhao Wang, Yuxiang Wang, and Dingcheng Ma. 2021. Two-stage routing with optimized guided search and greedy algorithm on proximity graph. *Knowledge-Based Systems* 229 (2021), 107305.

[76] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–26.

[77] Ming Yang, Yuzheng Cai, and Weiguo Zheng. 2024. CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search. *Advances in Neural Information Processing Systems* 37 (2024), 103076–103100.

[78] Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. 2024. Revisiting the index construction of proximity graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2410.01231* (2024).

[79] Shi Yu, Chaoyue Tang, Bokai Xu, Junbo Cui, Junhao Ran, Yukun Yan, Zhenghao Liu, Shuo Wang, Xu Han, Zhiyuan Liu, et al. 2024. Visrag: Vision-based retrieval-augmented generation on multi-modality documents. *arXiv preprint arXiv:2410.10594* (2024).

[80] Yuanhang Yu, Dong Wen, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2022. GPU-accelerated proximity graph approximate nearest Neighbor search and construction. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 552–564.

[81] Qiang Yue, Xiaoliang Xu, Yuxiang Wang, Yikun Tao, and Xuliyuan Luo. 2024. Routing-Guided Learned Product Quantization for Graph-Based Approximate Nearest Neighbor Search. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4870–4883.

[82] Huayi Zhang, Lei Cao, Yizhou Yan, Samuel Madden, and Elke A Rundensteiner. 2020. Continuously adaptive similarity search. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2601–2616.

[83] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 377–395.

[84] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*. PMLR, 838–846.

[85] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.

[86] Liangli Zhen, Peng Hu, Xu Wang, and Dezhong Peng. 2019. Deep supervised cross-modal retrieval. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10394–10403.

[87] Dantong Zhu and Minjia Zhang. 2021. Understanding and Generalizing Monotonic Proximity Graphs for Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2107.13052* (2021).

APPENDIX

A. PROOF FOR LEMMA 2

Since BuildCDMG (Algorithm 3) obtains edges by invoking RobustPruning (Algorithm 1) twice and then merging the results, we begin by proving that the expected out-degree from a single invocation of RobustPruning is $O(\ln|V|)$, where $|V|$ is the number of vertices.

Case 1: We first consider the scenario where $\delta(x_i, x_j) \le 3\tau$ and show that the expected out-degree is $O(\ln|V|)$. Because RobustPruning connects each point to others within the ball $B(x_i, 3\tau)$, the expected out-degree equals the expected number of points in this ball. Assuming points are uniformly distributed in the space and the point density in this space is $g = O(\ln|V|)$, the expected number of points within the ball $B(x_i, 3\tau)$ is $O(\ln|V|)$, as τ is a constant.

Case 2: For $\delta(x_i, x_j) > 3\tau$, vertex v_i links to v_j if and only if $\delta(x_j, x^*) + 3\tau > \delta(x_i, x_j)$. The probability that x^* falls within the region $Vol(lune(x_i, x_j) \setminus B(x_j, \delta(x_i, x_j) - 3\tau))$ is given by

$$\begin{aligned} P(x_i, x_j) &= \frac{Vol(lune(x_i, x_j) \setminus B(x_j, \delta(x_i, x_j) - 3\tau))}{Vol(lune(x_i, x_j))} \\ &< 1 - (1 - \frac{3\tau}{\delta(x_i, x_j)})^D \quad . \end{aligned} \tag{6}$$

Using Bernoulli's inequality, the expected out-degree becomes:

$$\begin{aligned} P &= \sum_{x_j \in \{V \setminus x_i\}} 1 - (1 - \frac{3\tau}{\delta(x_i, x_j)})^D \\ &< D \cdot 3\tau \sum_{x_j \in \{V \setminus x_i\}} \frac{1}{\delta(x_i, x_j)} \quad . \end{aligned} \tag{7}$$

For continuous distributions, let R denote the maximum distance between any two points in the space. Then:

$$P \le D \cdot 3\tau \int_{3\tau}^{R} \frac{1}{x} dx \le D \cdot 3\tau (\ln R - \ln 3\tau) \quad , \tag{8}$$

which gives the expected out-degree is $O(\ln R)$. Let the volume of the entire space be bounded by $Vol(B(R)) \le \psi \cdot Vol_V \le \psi \cdot \frac{|V|}{g}$. Since $Vol(B(R)) = \frac{\pi^{D/2} R^D}{\Gamma(1+D/2)}$, let $b = \frac{\pi^{D/2}}{\Gamma(1+D/2)}$. Then:

$$R \le (\frac{\psi|V|}{gb})^{\frac{1}{D}} \quad . \tag{9}$$

Hence, the expected out-degree from a single invocation of RobustPruning is $O(\ln R) = O(\ln |V|^{1/D}) = O(\ln|V|)$. As BuildCDMG invokes RobustPruning twice and merges the results, the overall expected out-degree is $O(2 \cdot \ln|V|)$, which simplifies to $O(\ln|V|)$. □

B. PROOF FOR LEMMA 4

Given a database $\mathcal{X} \subset \mathbb{R}^D$ and a query $q \in \mathbb{R}^D$, both sharing the same underlying distribution, the query q and its nearest neighbor x_n satisfy that $\delta(q, x_n) < \tau$. For a query q and radius $r > 0$, let $B(q, r)$ denote a ball of radius r centered at q, and $Vol(B(q, r))$ represent the volume of $B(q, r)$.

Consider a graph G constructed as a τ-MNG on the database $\mathcal{X}$. There exists a monotonic path $P = [x_0, \ldots, x_{\ell-1}, x_n]$, where ℓ is the length of the monotonic path. We can construct a sequence balls centered at q: $B(q, \delta(q, x_0)), \ldots, B(q, \delta(q, x_{\ell-1})$.

Define $\eta_{q,i} = \frac{Vol(B(q, \delta(q, x_{i+1}))}{Vol(B(q, \delta(q, x_i))}$ for $i = 0, \ldots, \ell-2$. It follows:

$$\eta_{q,i} = (\frac{\delta(q, x_{i+1})}{\delta(q, x_i)})^D, i = 0, \ldots, \ell-2 \quad . \tag{10}$$

Due to the graph G satisfy that $\delta(q, x_i) - \delta(q, x_{i+1}) > \tau, i = 0, \ldots, \ell-2$. Let $R_q = max_{x \in \mathcal{X}} \delta(q, x)$. It follows that:

$$\eta_{q,i} \le (\frac{\delta(q, x_i) - \tau}{\delta(q, x_i)})^D \le (\frac{R_q - \tau}{R_q})^D \tag{11}$$

and

$$\begin{aligned} Vol(B(q, \delta(q, x_{\ell-1}))) &= Vol(B(q, \delta(q, x_0)))\eta_{q,0} \ldots \eta_{q,\ell-2} \\ &\le Vol(B(q, R_q))\left(\left(\frac{R_q - \tau}{R_q}\right)^D\right)^\ell \quad . \end{aligned} \tag{12}$$

Recall that R denotes the largest distance between any two base vectors in the database $\mathcal{X}$. Since $R_q \le R + \tau$, we obtain:

$$\frac{Vol(B(q, \delta(q, x_{\ell-1}))}{Vol(B(q, R_q))} \le \left(\left(\frac{R}{R+\tau}\right)^D\right)^\ell \quad . \tag{13}$$

Let $\hat{\eta} = \left(\frac{R}{R+\tau}\right)^D$. Since $\hat{\eta} < 1$, we have:

$$\ell \le log_{\hat{\eta}} \frac{Vol(B(q, \delta(q, x_{\ell-1})))}{Vol(B(q, R_q))} = D \cdot log_{\hat{\eta}} \frac{\delta(q, x_{\ell-1})}{R+\tau} \quad . \tag{14}$$

The expectation of ℓ over all possible q is

$$\mathbb{E}[\ell] \le \frac{\mathbb{E}[\ln(\delta(q, x_{\ell-1}))] - \ln(R+\tau)}{\ln R - \ln(R+\tau)} \tag{15}$$

Define $f(R) = \frac{\mathbb{E}[\ln \delta(q, x_{\ell-1})] - \ln(R+\tau)}{\ln R - \ln(R+\tau)}$. Then, $\mathbb{E}[\ell] \le f(R)$. The derivative of $f(R)$ w.r.t. R is

$$f'(R) = \frac{\frac{1}{R+\tau}(\ln(R+\tau) - \ln R)}{(\ln R - \ln(R+\tau))^2} + \frac{(\ln(R+\tau) - \mathbb{E}[\ln \delta(q, x_{\ell-1})])(\frac{1}{R} - \frac{1}{R+\tau}))}{(\ln R - \ln(R+\tau))^2} \quad . \tag{16}$$

Since $R + \tau > \delta(q, x_{\ell-1}), f'(R) > 0$, implying that f is an increasing function. Due to the base vectors are uniformly distributed with density g. We have $g \cdot V_{\mathcal{X}} = |V|$. Since g increases with $|V|$, it must satisfy $g \ge g_0$ for some constant g_0. Therefore, $Vol(B(R)) \le \frac{|V|}{g} \le \frac{|V|}{g_0}$. The volume of ball $Vol(B(R))$ is given by:

$$Vol(B(R)) = \frac{\pi^{D/2} R^D}{\Gamma(1+D/2)} \quad , \tag{17}$$

where Γ is the Gamma function. Let $b = \frac{\pi^{D/2}}{\Gamma(1+D/2)}$. Then, $R \le (\frac{|V|}{g_0 b})^{1/D}$. Define $r(|V|) = (\frac{|V|}{g_0 b})^{1/D}$, since f is an increasing function, we have the following

$$\mathbb{E}[\ell] \le f(r(|V|)) = \frac{\mathbb{E}[\ln \delta(q, x_{\ell-1})] - \ln(r(|V|) + \tau))}{\ln r(|V|) - \ln(r(|V|) + \tau)} \quad . \tag{18}$$

Since $\delta(q, x_{\ell-1}) \ge \delta(q, x_n)$. We analyze two cases: (i) If $\tau \ge \frac{\sqrt{D}}{2}(\frac{2}{g})^{1/D}$, $\mathbb{E}[\ln \delta(q, x_n)] \ge \ln(\frac{1}{2}(\frac{2}{\ln n})^{\frac{1}{m}})$. (ii) If $\tau \le \frac{\sqrt{D}}{2}(\frac{2}{g})^{1/D}$, $\mathbb{E}[\ln \delta(q, x_n)] \ge \ln(\tau/2)$. Therefore,

$$\mathbb{E}[\ell] \le O(\frac{(r(|V|) + \tau)\ln(r(|V|) + \tau)}{\tau}) \quad . \tag{19}$$

Since τ is a constant and independent of $|V|$, it follows that

$$\mathbb{E}[\ell] = O(r(|V|) \ln r(|V|)) = O(|V|^{\frac{1}{D}} \ln |V|) \tag{20}$$

Due to the expected out-degree of MRNG is $O(\ln|V|)$, therefore the search time complexity is $O(|V|^{\frac{1}{D}} (\ln|V|)^2)$. □

C. DATASET

We provide a detailed introduction to the five modern large-scale OOD datasets as follows:

- **Text-to-Image**: The Text-to-Image[3] is a cross-domain dataset derived from a visual search engine. The database vectors are image embeddings generated by the Se-ResNext-101 model, while the query vectors are extracted form user-specified textual queries using a variant of the DSSM model.
- **LAION**: The LAION[4] is a widely used text-image dataset. Both images and text are embedded using CLIP-ViT-B/32.
- **WIT**: The WIT dataset[5] is a dataset derived from a text-to-image application. The image-text pairs are extracted from Wikipedia pages. Database vectors are encoded using CLIP-ViT-B/32, while queries are generated with CLIP-ViT-B32-multilingual-v1.
- **WebVid**: The WebVid[6] is a caption-video dataset sourced from stock footage sites. The embeddings are encoded using CLIP-ViT-B/32.
- **CC3M**: The CC3M[7] is a dataset of image-text pairs designed for image captioning systems. Database vectors are encoded using ViT-B/16, while attached text are transformed using BERT.

In our experiments, all vectors across the datasets are normalized, and the ground truth is computed based on cosine similarity.

D. BASELINES & PARAMETER SETTINGS

We compare CDMG with several state-of-the-art graph-based methods, with their detailed descriptions and experimental parameter settings provided as follows.

- **HNSW**: A well-known hierarchical graph-based search method. We set the out-degree parameter to $M = 32$ after testing values from 8 to 48, and the $efConstruction$ is set to 500.
- **τ-MNG**: An optimized version of NSG with additional connections for improved recall. It shares parameters $R = 64, C = 500, L = 500$ with NSG, and we fine-tune τ from 0.01 to 0.3, setting $\tau = 0.01$ for optimal performance.
- **RobustVamana**: A graph index specifically designed for OOD queries. We set $R = 64$, $L = 500$, and $\alpha = 1.0$ after tuning α from 1.0 to 1.2. Unless otherwise specified, the size of the queries is 10% of the database.
- **RoarGraph**: RoarGraph is an efficient graph index for OOD queries. We configure $N_q = 100, M = 35$ and $L = 500$ following the official instructions. The size of the queries is the same as in RobustVamana, representing 10% of the database.
- **CDMG** (proposed): We configure $S = 15$ during query aggregation, with $M = 35$ and $L = 500$ for graph index construction. The size of the queries is 10% of the database.

E. RELATED WORK

Graph-based Vector Similarity Search. Numerous algorithms have been proposed to address the vector similarity search issue, broadly divided into four types: graph-based, quantization-based, hash-based, and tree-based methods. Among these, graph-based methods have gained a strong position in the trade-off between search latency and accuracy. This advantage is due to two key properties of graph indexes: (i) they exhibit superior navigability, enabling rapid graph traversal to locate a small set of database vectors close to the query, and (ii) they effectively capture and express clusterability for search results. Existing graph indexes are derived from base graphs with strong theoretical guarantees, e.g., DG, RNG. However, they were not originally designed for vector similarity search problems and typically suffer from high construction and search complexities. Current graph indexes enhance these base graphs in various ways to better suit retrieval tasks. For instance, HNSW improves DG and RNG with a hierarchical structure and a heuristic edge selection process, maintaining a hierarchical structure based on neighbor distances, which achieves nearly logarithmic search time complexity. NSG introduces MRNG, ensuring that a monotonous path exists between any two vertices in the resulting graph. Vamana and τ-MNG further enhance the flexibility of edge selection by adding a query relaxation parameter.

OOD-oriented Graph-based Methods. Mainstream graph-based methods are designed for ID queries, resulting in limited search performance on OOD vector similarity search. To mitigate this limitation, two state-of-the-art methods have been proposed: RobustVamana [26] and RoarGraph [7]. RobustVamana integrates a subset of the query set during construction and establishes connections between database vectors that co-occur as out-neighbors of an indexed query. RoarGraph constructs the graph index under the guidance of query distribution. It follows three sequential processes. First, a query-base graph is constructed as the foundation, where queries are treated as bridges to connect their scattered ground truths. Then, the query-base graph is projected onto the base vectors, preserving only the relationships of the base vectors identified by the queries. Finally, neighbors of each base vector are supplemented to enhance connectivity. However, both methods depend on ID-oriented candidate acquisition and neighbor selection, which do not align with OOD queries. Recent studies have also investigated the OOD vector similarity search problem through approaches such as product quantization (e.g., APQ), dimensionality reduction (e.g., LeanVec and GleanVec), and entry point selection. These algorithms are orthogonal to the graph index.

[3]https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search
[4]https://laion.ai/blog/laion-400-open-dataset/
[5]https://github.com/IntelLabs/VectorSearchDatasets
[6]https://github.com/m-bain/webvid
[7]https://github.com/google-research-datasets/conceptual-captions