

On the Effectiveness of Graph Reordering for Accelerating Approximate Nearest Neighbor Search on GPU

Yutaro Oguri

The University of Tokyo
Tokyo, Japan
oguri@hal.t.u-tokyo.ac.jp

Mai Nishimura

OMRON SINIC X Corporation
Tokyo, Japan
mai.nishimura@sinicx.com

Yusuke Matsui

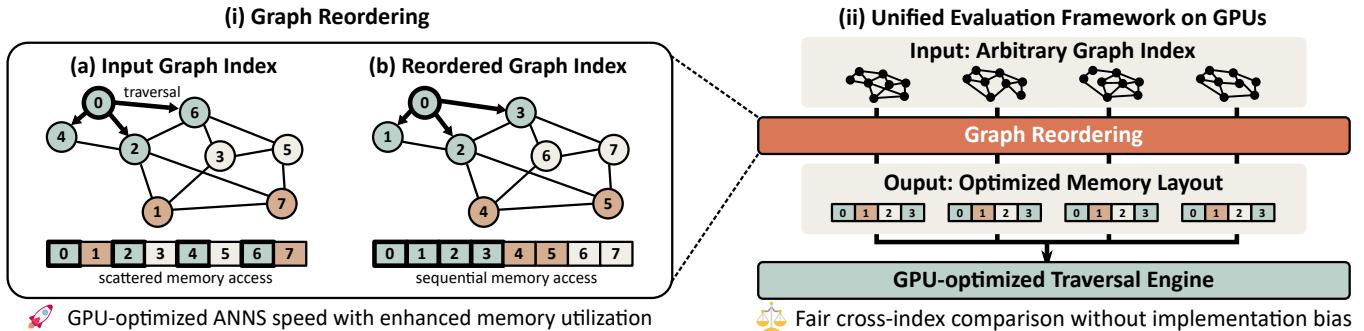
The University of Tokyo
Tokyo, Japan
matsui@hal.t.u-tokyo.ac.jp

Figure 1: (i) Graph Reordering transforms scattered memory access patterns into sequential ones by relocating co-accessed nodes to consecutive memory regions. (ii) Unified Execution Framework processes arbitrary graph topologies through optimized memory layouts and standardized GPU search algorithms, enabling fair performance evaluation across different graph indices.

Abstract

We present the first systematic investigation of graph reordering effects for graph-based Approximate Nearest Neighbor Search (ANNS) on a GPU. While graph-based ANNS has become the dominant paradigm for modern AI applications, recent approaches focus on algorithmic innovations while neglecting memory layout considerations that significantly affect execution time. Our unified evaluation framework enables comprehensive evaluation of diverse reordering strategies across different graph indices through a graph adapter that converts arbitrary graph topologies into a common representation and a GPU-optimized graph traversal engine. We conduct a comprehensive analysis across diverse datasets and state-of-the-art graph indices, introducing analysis metrics that quantify the relationship between structural properties and memory layout effectiveness. Our GPU-targeted reordering achieves up to 15% QPS improvements while preserving search accuracy, demonstrating that memory layout optimization operates orthogonally to existing algorithmic innovations. We will release all code upon publication to facilitate reproducibility and foster further research.

Keywords

Vector Search, Graph Reordering, Graph-based Index, GPU

1 Introduction

Approximate Nearest Neighbor Search (ANNS) addresses the fundamental challenge of efficiently retrieving the most similar vectors to queries from a large-scale database [9]. Graph-based approaches, in particular, dominate GPU-accelerated ANNS due to superior accuracy-throughput trade-offs [20, 31], becoming foundational infrastructure for retrieval systems [35], vector databases [33, 45],

and modern AI applications such as Retrieval-Augmented Generation (RAG) [35, 50]. Graph-based ANNS, in principle, depends on constructing effective graph representations that approximate high-dimensional proximity relationships, where structural properties directly determine search performance. Despite extensive topology optimization research [11, 31, 41], a critical gap persists between algorithmic design and GPU execution efficiency. GPU performance critically depends on coalesced memory access patterns, yet current approaches focus on graph topological optimization while neglecting its memory layout. This leaves a fundamental question underexplored – “Given a constructed graph index, how should we arrange vertices in memory to alleviate memory access bottlenecks and thereby maximize ANNS search efficiency on GPU?”

Our ultimate vision is a platform that automatically optimizes memory layouts for arbitrary graph indices and datasets, enabling them to achieve near-peak search throughput on a GPU. This presents several fundamental challenges that make it non-trivial. At its core, the problem requires identifying node groups that will be accessed concurrently or sequentially during query execution and co-locating them in consecutive memory regions. However, access patterns are query-dependent and vary with traversal entry points, making comprehensive pattern prediction computationally intractable. As a first step toward this vision, we aim to explore memory layout optimization based on graph topology characteristics. Yet current graph indices are implemented independently, each with its own data structures and traversal algorithms, making systematic evaluation difficult. No unified framework exists to isolate memory layout effectiveness from algorithmic differences, making a fair comparison of graph index quality versus memory optimization impossible.

To address these challenges, we develop a systematic evaluation framework with two key components. As illustrated in Fig. 1, (i) we explore the effectiveness of various graph reordering strategies across different indices and (ii) establish a unified evaluation framework that separates graph topology design from search implementation for fair comparison. Graph reordering provides the most direct approach to memory layout optimization, as vertex IDs map one-to-one to physical memory positions in standard graph representations. Our framework converts arbitrary graph topologies into a common representation and evaluates them using CAGRA’s state-of-the-art GPU search implementation [31], enabling the pure comparison of graph topologies without implementation factors.

Built upon our framework, we conduct a comprehensive analysis across multiple datasets and graph indices (CAGRA [31], NSG [11], Vamana [41], NN-Descent [8]) to evaluate reordering effectiveness. We introduce graph analysis metrics including community structure [47] to quantify how structural properties influence memory layout optimization. This analysis provides important insights into memory layout based on topology characteristics and informs future graph index design.

Our work represents the first systematic evaluation of memory layout optimization for graph-based ANNS on a GPU. Critically, reordering operates orthogonally to algorithmic design, enabling performance gains without altering search accuracy or graph structures, making our approach universally applicable across any existing graph-based ANNS methods with diverse graph topologies. In summary, our contributions are fourfold:

- **A unified ANNS evaluation framework on GPU** that systematically evaluates arbitrary graph indices with GPU-optimized memory layouts and a unified search algorithm, establishing the first common platform for fair cross-index comparison.
- **First analysis of pure graph index performance on GPU**, isolating topological characteristics from implementation differences and providing valuable insights across classical and modern datasets, considering AI-application workloads underexplored in prior ANNS research.
- **Comprehensive empirical analysis indicating universal reordering effectiveness** across diverse datasets and graph indices, including quantitative correlation analysis between graph community structure and memory layout optimization.
- **First demonstration of GPU-targeted memory reordering** for graph-based ANNS, achieving up to 15% QPS improvements through topology-agnostic optimization that unlocks the latent performance while preserving the search accuracy.

2 Related Work

2.1 ANNS on GPU

Approximate Nearest Neighbor Search (ANNS) has evolved through distinct paradigms on CPU architectures, encompassing tree-based structures [10, 13, 38], locality-sensitive hashing (LSH) [1, 12, 49], inverted file indexing (IVF) [4] with Product Quantization(PQ) [16, 19], and graph-based approaches [11, 41]. While effective for many practical applications, CPU-based approaches face fundamental limitations in memory bandwidth and parallelism.

Breaking through these limitations, GPU-based approaches have revolutionized ANNS performance. Faiss-GPU [18] first demonstrated substantial speed-ups through a parallel k -selection algorithm. Subsequently, graph-based GPU algorithms have emerged as the dominant paradigm [31, 43]. However, fully harnessing massive parallelism and high memory bandwidth requires sophisticated implementation optimizations. Our work focuses not on algorithmic innovation, but rather on memory layout optimization to unlock GPU hardware performance without modifying the graph topology.

2.2 Graph-based ANNS

Graph-based ANNS algorithms can be systematically classified along two primary dimensions – *Graph Construction* and *Search*.

Graph Construction. Graph construction methods produce either multi-layer hierarchical structures (HNSW [23]) or single-layer flat indices (NSG [11], DiskANN [41]) with varying connectivity patterns and degree constraints. While GPU implementations use formats such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) for graph representation, access pattern-aware node ordering strategies remain largely underexplored.

Search Strategy. Search strategies divide into hierarchical approaches [23] and single-layer methods [11, 41] employing greedy beam search with sophisticated entry point selection [27–29]. GPU-optimized variants such as CAGRA [31] deliberately eliminate hierarchy and adopt fixed out-degree designs to enable uniform parallel processing that maximizes GPU throughput.

Our framework adopts CAGRA’s single-layer search strategy as it best exploits GPU parallelism, enabling direct performance comparison of graph indices that were previously evaluated in isolation. Graph reordering operates orthogonally to existing graph indices, extracting performance improvements while preserving the search accuracy.

2.3 Graph Reordering

Graph reordering techniques optimize memory access patterns by reassigning vertex indices to improve spatial locality. These methods are systematically categorized into four primary approaches: community-aware algorithms [21], degree/hub-based strategies [3, 52], BFS-based approaches [7], and window-based approaches [48]. While numerous graph reordering techniques have been proposed, their validation scope has been primarily limited to traditional graph algorithm benchmarks such as PageRank [32] and Strongly Connected Component (SCC) detection [37] on CPUs. Recent work explores the applicability of graph reordering to CPU-based ANNS [5] and Graph Neural Networks (GNNs) [26], demonstrating promising results. Specifically, the work reports that the graph reordering accelerates the CPU-based ANNS by 10 - 40% [5]. Despite proven effectiveness in CPU-centric contexts, graph reordering effectiveness for GPU-accelerated ANNS systems remains fundamentally unexplored. Our work provides the first systematic evaluation of graph reordering strategies for GPU-based ANNS performance across multiple index types, with a novel focus on analyzing the relationship between graph index community structure and data characteristics for reordering effectiveness.

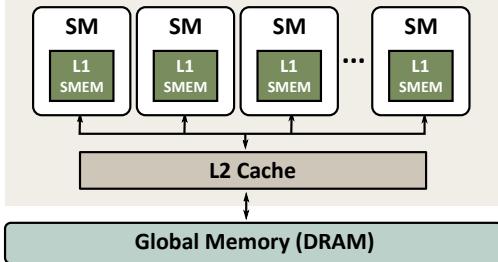


Figure 2: GPU Memory Hierarchy. Global memory provides high capacity but low bandwidth, while L1 cache and shared memory (SMEM) offer high bandwidth with limited capacity.

3 Preliminaries

3.1 Approximate Nearest Neighbor Search (ANNS)

Consider a database containing N d -dimensional vectors $\mathcal{D} = \{x_1, x_2, \dots, x_N\} \subset \mathbb{R}^d$. Given a query vector $q \in \mathbb{R}^d$, the fundamental vector search problem seeks to identify the most similar vector in the database,

$$n = \underset{i \in \{1, 2, \dots, N\}}{\operatorname{argmin}} \|q - x_i\|_2^2. \quad (1)$$

This brute-force approach necessitates evaluating all N vectors, leading to $\mathcal{O}(Nd)$ computational complexity that becomes impractical as datasets scale to millions or billions of vectors.

To address this scalability challenge, ANNS employs index structures [19, 23, 33] that transform the feature space into efficiently searchable representations. ANNS pursues the trade-off between search efficiency and accuracy by approximating Eq. (1). Instead of guaranteeing the exact closest vector, ANNS algorithms aim to find vectors that are close enough to the ground-truth nearest neighbor vector. In particular, graph-based ANNS methods achieve fast search by introducing graph-structured search indices that enable efficient navigation through the vector space [2, 46].

3.2 Graph-based Index

Graph-based index models the vector space as a connectivity graph $G = (\mathcal{V}, \mathcal{E})$, where each vertex $v_i \in \mathcal{V}$ represents a d -dimensional database vector $x_i \in \mathcal{D}$. Each edge $(v_i, v_j) \in \mathcal{E}$ is constructed based on algorithmic heuristics designed to enhance subsequent search efficiency [11, 41]. The search is performed by traversing the graph structure. The fundamental strategy involves two key components: (1) constructing a well-structured graph index that enables efficient exploration, and (2) performing beam search-based traversal [11, 31] to navigate toward the query's nearest neighbors.

3.3 Graph Reordering for ANNS

Given a graph index $G = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V} = \{v_1, v_2, \dots, v_N\}$ and their current memory layout, graph reordering seeks to find an optimal permutation π that reassigns each vertex v_i to a new index $\pi(i)$, i.e., relocating the vertex v_i from memory location l_i to $l_{\pi(i)}$.

Objective of Reordering. The objective is to assign close IDs to vertices frequently accessed together during traversal, ensuring

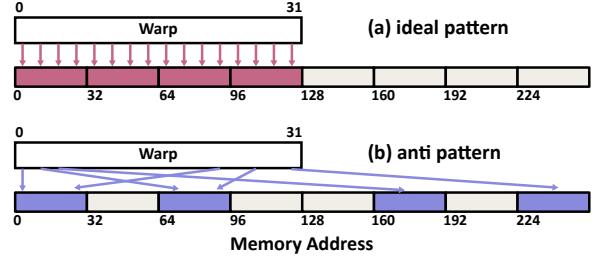


Figure 3: Top: Coalesced access where threads access consecutive memory addresses. Bottom: Worst access patterns with scattered memory requests.

consecutive memory storage that maximizes spatial locality. In standard graph representations, vertex IDs directly index arrays, establishing a one-to-one mapping between ID ordering and physical memory layout. Thus, optimizing vertex assignment directly translates to memory layout optimization. Figure 1 (i) illustrates this effect: consecutive tiled boxes represent memory regions with vertex IDs inside. During traversal from entry point v_0 , vertices at identical hop distances are accessed simultaneously, while distant vertices have lower co-access probability. The transition from (a) to (b) demonstrates how reordering collocates frequently accessed vertices in consecutive memory locations.

Graph Traversal and Reordering. Graph-based ANNS algorithms predominantly employ beam search, which maintains a dynamic list of candidate vertices and iteratively explores their neighborhoods to approach the query's nearest neighbors. During each iteration, the algorithm accesses neighbors of current candidates, computes distances to the query, and updates the candidate list. This generates irregular memory access patterns since topologically connected vertices are often stored in non-contiguous memory locations, resulting in poor cache locality. Graph reordering mitigates this problem by relocating frequently co-accessed vertices to adjacent memory positions, transforming scattered memory requests into sequential accesses that improve cache utilization and memory coalescing on GPUs. Since most graph-based ANNS methods [11, 31, 41] employ similar beam search-based traversal patterns, reordering effectiveness in this study extends broadly across different algorithms with comparable vertex access behaviors.

3.4 Memory Hierarchy of GPU

Modern GPU architectures¹ employ a fundamentally different memory hierarchy design compared to traditional CPUs, optimized for high-throughput parallel computation rather than single-thread performance. While CPUs prioritize cache efficiency and latency minimization for sequential processing, a GPU maximizes memory bandwidth utilization to serve thousands of concurrent threads simultaneously. As depicted in Fig. 2, the GPU memory hierarchy consists of multiple levels with distinct characteristics. *Global memory* provides the largest storage capacity accessible to all processing

¹In this paper, we focus on NVIDIA GPU architectures and CUDA programming model. While the general principles apply to other GPU vendors, specific optimizations and memory hierarchy details may vary across different architectures.

units but exhibits high access latency. *L2 cache* serves as an intermediate layer offering moderate capacity with improved access speed. *L1 cache* along with shared memory delivers the fastest access times, but with limited storage space.

Figure 3 depicts typical examples of memory access patterns. NVIDIA GPUs execute threads in groups of 32 called warps, where memory requests from threads within a warp can be coalesced into fewer transactions. Ideally, when threads access consecutive memory addresses, multiple individual requests are combined into single coalesced transactions, maximizing bandwidth utilization.

In GPU-based graph traversal, memory layout optimization by reordering may improve performance via two main factors: (i) *coalesced access* to global memory, and (ii) *better cache locality* in L1 or L2. These effects are often entangled in highly optimized kernels, making it difficult to isolate their individual contributions. Thus, instead of low-level profiling, we take an empirical approach by measuring the combined impact of reordering by evaluating high-level metrics such as QPS. This forms the basis of our performance analysis.

4 Evaluation Framework

4.1 Overview

Figure 4 depicts an overview of our framework for reordering effectiveness. The evaluation framework consists of (i) a unified search execution engine, (ii) a general graph adapter, and (iii) a fine-grained index analyzer.

Our framework addresses the challenge that existing graph indices are distributed across heterogeneous implementations with distinct data structures and traversal algorithms, preventing systematic cross-index comparison of reordering effectiveness. For this, we introduce a unified search execution engine that unifies the traversal algorithms across different graph indices and adopts a state-of-the-art GPU-optimized search implementation. Specifically, we use CAGRA [31]’s traversal implementation provided by NVIDIA’s cuVS library [6], which harnesses advanced techniques such as software warp splitting and forgettable hash table management to efficiently utilize GPU resources and extract massive parallelism. The search kernel of the traversal engine takes a run-time parameter L , which determines the size of the priority queue used to store intermediate results during graph traversal. This design allows us to compare graph-based indices such as NSG and Vamana, which are originally designed for CPU execution, under a consistent and fair GPU setting.

The graph adapter takes a pre-constructed graph index in either adjacency list or CSR format as inputs and outputs a Faiss’s GPU-compatible index representation [9] without modifying the original graph topology. We support de facto standard formats in ANNS, including Faiss [9] and DiskANN [41], and we can extend the framework to arbitrary indices by implementing a component that extracts adjacency information. Our graph adapter allows easy integration of a new index by supplying minimal custom code. The index analyzer evaluates structural properties such as community structure and other metrics that influence memory layout and traversal performance, providing critical insights for reordering strategies and index design.

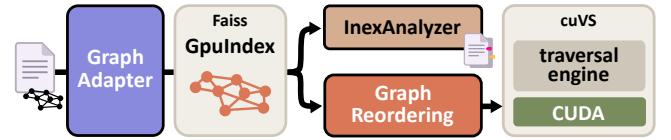


Figure 4: Our framework transforms arbitrary graph indices into a standardized Faiss [9] GpuIndex via Graph Adapter. IndexAnalyzer extracts structural properties, and cuVS [6] traversal engine enables fair cross-index comparison on a GPU.

Using our framework, we can systematically benchmark each graph index G across multiple reordering functions π and datasets \mathcal{D} . Beyond traditional ANNS efficiency metrics, our evaluation incorporates structural analysis to provide a multi-faceted assessment of reordering effectiveness. By correlating graph characteristics such as community structure with memory layout performance, we identify fundamental principles underlying effective optimization and provide empirically grounded insights for future index design.

4.2 Graph-based Indices

We employ the following graph-based indices with a single-layer graph structure.²

- **CAGRA** [31]: CAGRA constructs a fixed out-degree directional graph through a two-stage process. We first build an initial k-NN graph using NN-descent [8], then optimize it via rank-based edge reordering and reverse edge addition to improve node reachability and reduce strongly connected components. cuVS [6, 31] provides a highly optimized implementation of CAGRA for GPU.
- **NSG** [11]: Navitaing Spreading-out Graph (NSG) is a graph that approximates Monotic Relative Neighborhood Graph (MRNG) with a navigating node, designed to ensure connectivity, reduce out-degree, and shorten search paths. NSG is known to perform as well as or better than HNSW [25, 30]. We use the NSG implementation provided in Faiss [9].
- **Vamana** [41]: The graph-based index used in DiskANN [41], originally designed for SSD-resident indices with a tunable parameter to minimize graph diameter and reduce sequential disk reads. Note that this paper handles Vamana as an on-memory index. We use the implementation provided in the DiskANN repository [40].
- **NN-Descent** [8]: NN-Descent starts from a random k -NN approximation and repeatedly improves it by exploring each point’s neighbors, updating the graph when closer neighbors are found. Through optimizations such as local join, incremental search, sampling, and early termination, NN-Descent achieves high recall with low time complexity. NSG adopts NN-Descent as the base kNN graph construction [11]. We use the NN-Descent implementation provided in Faiss [9].

CAGRA is a state-of-the-art index designed for GPUs. NSG and Vamana (DiskANN) provide one of the best accuracy-speed trade-offs on a CPU. NN-Descent does not outperform other indices in general, but we adopt it because NN-Descent provides a simple

²We exclude hierarchical indices like HNSW [23] and focus exclusively on single-layer indices as adopted by CAGRA [31], which maximize GPU parallelism and throughput.

k NN graph structure and serves as a base graph for other advanced indices like NSG and CAGRA. We port these CPU-designed graph structures to the GPU for comparative analysis using our evaluation framework.

4.3 Reordering Algorithms

To study the effect of memory layout on GPU-based ANNS performance, we evaluate a diverse set of graph reordering algorithms. These algorithms were selected to cover a wide range of strategies, including topology-aware heuristics, degree-based sorting, and a BFS-based approach. We note that all methods operate only on the order of graph node placement in memory without changing the graph topology itself. The following algorithms are considered:

- **Degree Sort** [3]: A simple degree-based sorting strategy that orders nodes based on their degree values and assigns new consecutive IDs. In our experiments, we consider two variants: one using in-degree (**Indegree Sort**) and the other using out-degree (**Outdegree Sort**).
- **Hub Sort** [52]: A clustering strategy that identifies high-degree hub nodes above a predefined threshold. These hubs are grouped together while preserving their relative order, which aims to exploit spatial locality during graph traversal.
- **GOrder** [48]: A window-based strategy that maximizes graph locality by optimizing a cost function which quantifies shared neighbors and edge connectivity within sliding windows. We use a window size $w = 10$, which is a similar value used in prior work [5].
- **RCM** [7]: Reverse Cuthill-McKee (RCM) algorithm that performs BFS traversal with degree-sorted queue insertion to minimize graph bandwidth and improve cache efficiency.
- **Random**: A uniformly random permutation of the node IDs. This serves as a control to assess whether the benefits of other reorderings come from meaningful structural changes rather than incidental reshuffling.

4.4 Datasets

Our framework executes and evaluates the graph-based indices on (i) traditional ANN benchmark datasets and (ii) modern high-dimensional and cross-modal datasets representing contemporary AI applications, including text-to-image search and LLMs. Reordering effectiveness depends heavily on vector dimensionality, *i.e.*, how many neighboring node features can fit within the GPU’s parallel execution unit (warp), making traditional low-dimensional datasets insufficient for comprehensive evaluation.

Considering that these contemporary AI applications, like cross-modal retrieval and RAG, are based on GPU, we also employ the modern datasets, which are uncommon in the previous ANNS research.

(i) Traditional benchmarks include:

- **SIFT** [19]: The local SIFT descriptors of images.
- **GIST** [19]: The global color GIST descriptors of images.
- **Deep** [51]: The image embeddings generated by GoogLeNet [42].

(ii) Modern benchmarks include:

Table 1: Dataset descriptions. ‘Dim’ means the dimensionality of vectors.

Dataset	#Vectors	Dim.	#Query	Metric
SIFT [19]	1M	128	10,000	L2
GIST [19]	1M	960	1,000	L2
Deep [51]	1M, 10M	96	10,000	L2
Yandex T2I [36]	1M	200	100,000	Inner Product
OpenAI Embed. [39]	1M	1536	10,000	L2
Wikipedia [53]	1M, 10M	768	1,000	Inner Product
BioASQ [53]	1M, 10M	1024	3,106	Inner Product
C4 [53]	5M	1536	1,000	Inner Product

- **Yandex Text-to-Image (Yandex T2I)** [36]: The database is composed of image embeddings generated by the Se-ResNext-101 [14] model, and the query set is text embeddings generated by a variant of DSSM [15] model. The database and the query set have different statistical characteristics, which are known to exhibit poor performance in ANNS [17].
- **OpenAI Embedding (OpenAI Embed.)** [24, 29, 39]: The text embeddings of WikiText generated by OpenAI’s model.
- **Wikipedia** [53]: The text embeddings of the Wikipedia corpus generated by Cohere V2 model.
- **BioASQ** [53]: Consists of the text embeddings of the BioASQ [22, 44] question answering corpus generated by OpenAI’s model.
- **C4** [53]: Consists of the text embeddings of the Colossal Clean Crawled Corpus (C4) [34] generated by Cohere V3 model.

4.5 Evaluation Metrics

We evaluate reordering effectiveness from two perspectives: *ANNS performance* and *graph structural properties*. For ANNS performance, we measure accuracy and search speed to assess the impact of memory layout optimization. For structural properties, we employ quantitative metrics to capture the inherent characteristics of graph-based indices and their relationship to reordering effectiveness.

4.5.1 Metrics for evaluating ANNS performance.

Recall@k. Accuracy is measured by Recall@ k , defined as the average fraction of ground-truth top- k neighbors recovered over the query set Q . For each query $q \in Q$, let $R^*(q)$ be the ground-truth top- k nearest neighbors and $\widehat{R}(q)$ the top- k neighbors returned by the algorithm. The Recall@ k for the query set Q is

$$\text{Recall}@k = \frac{1}{|Q|} \sum_{q \in Q} \frac{|R^*(q) \cap \widehat{R}(q)|}{k}. \quad (2)$$

QPS. We measure the speed of the search by Queries Per Second (QPS). It is a throughput metric: the number of queries completed per unit wall-clock time. A higher QPS indicates that the system processes more queries per second under identical conditions.

4.5.2 Metrics for analyzing the index structure.

Local Clustering Coefficient: To quantify the community strength in a graph $G(\mathcal{V}, \mathcal{E})$, we use the Local Clustering Coefficient (LCC) [47].

Local Clustering Coefficient (LCC) For a vertex v with degree $k_v \geq 2$, we define the LCC for the vertex v as

$$C_v = \frac{2 T_v}{k_v(k_v - 1)}, \quad (3)$$

where T_v is the number of triangles that include v (set $C_v = 0$ if $k_v < 2$). $C_v \in [0, 1]$. The larger value indicates that the neighbors of v are more densely interconnected.

Average Local Clustering Coefficient: Given a vertex set \mathcal{V} of a graph G , we define the average LCC as

$$\langle C \rangle = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} C_v. \quad (4)$$

The larger value indicates that G has stronger community structures.

5 Experimental Results

Our experimental evaluation focuses on addressing the following research questions:

- Q1** How do different graph index topologies perform under identical search algorithms?
- Q2** Does graph reordering improve the traversal speed for each graph index?
- Q3** Can reordering effectiveness be estimated from quantitative graph characteristics?
- Q4** How does the vector dimensionality affect reordering effectiveness?

We conduct all experiments on a single NVIDIA RTX 6000 Ada (48GB) GPU and Intel Xeon w5-3435x CPU (16 cores), both modern high-performance processors from the recent generation. We use the maximum degree $K = 32$ for all graph indices. We adopt the default values for the other build parameters defined in each index's implementation. We control the single runtime parameter L . We run each configuration five times and take the average of the resulting metrics.

5.1 Evaluation of Graph-based indices without Reordering (Q1)

Using our evaluation framework, we systematically benchmark the graph-based indices introduced in Sec. 4.2 across 12 datasets on a single GPU. We evaluate performance using Recall@10 and QPS. We test on 16 runtime parameters $L \in \{20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, 120, 140, 160, 180\}$. To the best of our knowledge, this is the first work to run NSG, Vamana, and NN-Descent on a GPU while preserving their original graph topologies.

Figure 5 demonstrates performance trade-off curves between the two metrics. Overall, as illustrated in Fig. 5 (a–c) and (d–l), NN-Descent consistently underperforms compared to the other indices, except on the Deep1M dataset. Figure 5 (c) indicates that NSG performs worst on Deep 1M. Interestingly, NSG and Vamana, although originally designed for CPU execution, perform as well as or better than the GPU-optimized CAGRA index on all datasets except BioASQ (Fig. 5 (j–k)). This tendency is independent of the vector dimension d or the number of database vectors N . Specifically, as demonstrated in Fig. 5 (h), NSG outperforms CAGRA on Wikipedia 1M dataset by 1.2 times.

On traditional datasets such as SIFT, GIST, and Deep, NSG and Vamana outperform CAGRA in the medium recall regions, particularly when Recall@10 < 0.9. Among modern datasets, we observe similar trends for OpenAI Embedding, Wikipedia, and C4, which consist of natural language corpora represented as very high-dimensional embeddings. In contrast, on the QA-focused dataset BioASQ, CAGRA consistently achieves the best performance with a large margin at both 1M and 10M scales.

5.2 Reordering comprehensive evaluation on real world dataset (Q2)

We apply the reordering algorithms introduced in Sec. 4.3 to four graph-based indices and evaluated the QPS improvement across 12 datasets compared to the baseline without reordering. We use the same runtime parameter set as Sec. 5.1. Figure 6 summarizes the results. For each index, we highlight two cases where reordering is effective and two where it is not. In some combinations of dataset and index, we observe up to 15% QPS improvement without compromising recall.

Figure 6 (a–d) illustrate the results for CAGRA. As illustrated in Fig. 6 (a), reordering achieves 10% speed-up at most. Figure 6b shows that the configuration with Wikipedia 10M dataset shows a limited but uniform improvement. On the other hand, as shown in Fig. 6 (c–d), on SIFT 1M and BioASQ 1M, most configurations show little to no speed-up and even degradation of QPS by 8–10% in high-recall regions. For NSG, Fig. 6 (e–f) show that reordering is effective on Deep 1M excluding the case with GOrder algorithm and on Yandex T2I for all methods, yielding up to 15% improvement. However, Fig. 6 (g–h) illustrate that the speed-up is negligible or even negative on modern datasets like OpenAI Embedding 1M and C4 5M. Figure 6 (i–j) and (m–n) demonstrate the positive case for Vamana and NN-Descent. They show significant QPS improvements on Deep1M and Yandex T2I 1M, reaching up to 15%. On the other hand, as illustrated on Fig. 6 (k–l) and (o–p), Wikipedia dataset shows no substantial speed-up for either the 1M or 10M scale. In addition, as shown in Fig. 7, we find that in some dataset-index combinations, Vamana and NN-Descent exhibit remarkably similar speed-up characteristics under reordering. This trend is particularly notable for BioASQ 1M / 10M and C4 5M.

Among the reordering algorithms, GOrder exhibits unstable performance across many cases. In particular, we observe steep drops in speed-up in high-recall regions. For the other algorithms, we find no consistent performance trends across datasets and indices.

5.3 Community characteristics of graphs and reordering (Q3)

To investigate whether the structural property of the graph affects the effectiveness of memory reordering, we evaluate the correlation between average LCC and the average speed-up achieved by reordering. We measure the two metrics for each (index, dataset) pair. Figure 8 describes the relationship between the speed-up and LCC. We observe no strong correlation between the two. Some indices with high LCC (e.g., Vamana and BioASQ 10M) exhibit almost no speed-up, while others with low LCC achieve notable improvements (e.g., NN-Descent and BioASQ 1M).

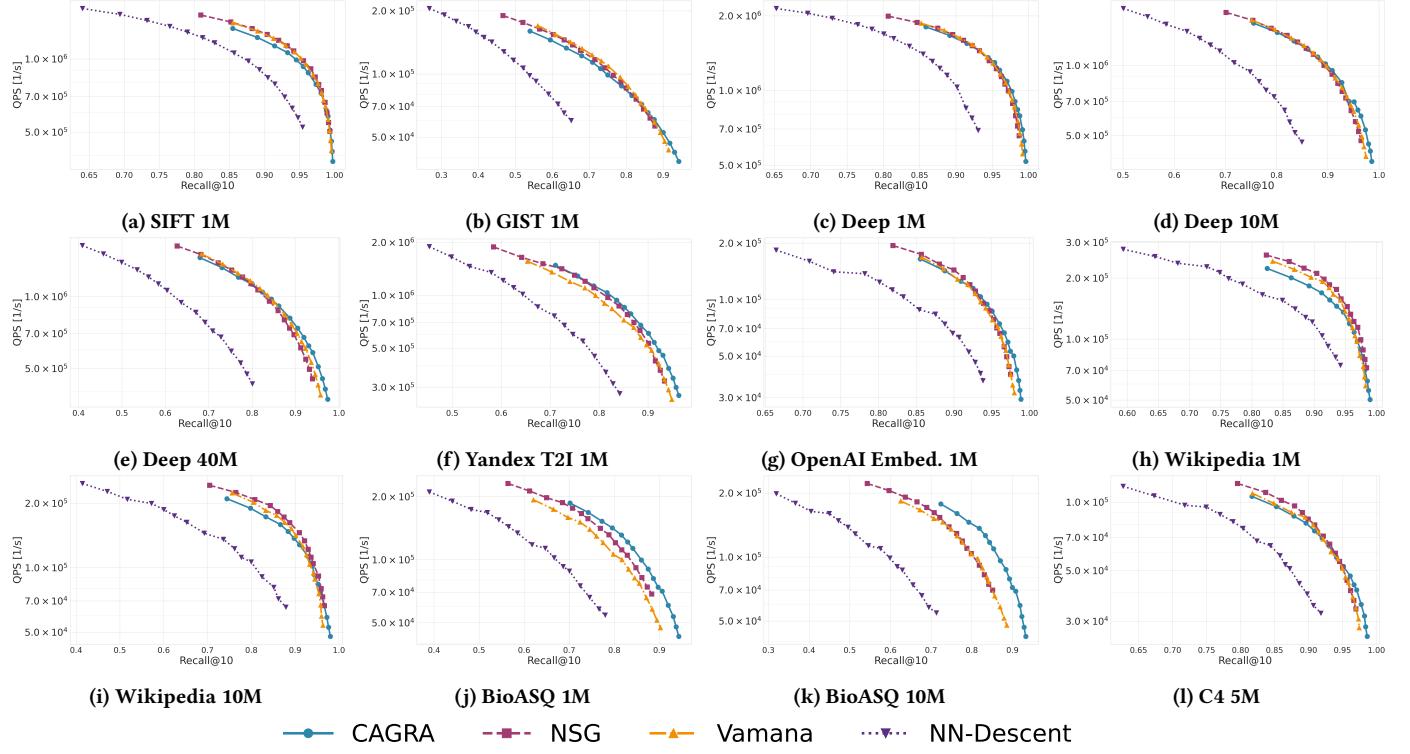


Figure 5: Recall-QPS trade-off curves for four graph-based indices (CAGRA, NSG, Vamana, NN-Descent) across 12 datasets evaluated under our framework with a unified, GPU-optimized traversal implementation.

Table 2: The correlation between vector dimensionality and reordering effectiveness. We use Spearman’s rank correlation coefficient r_s .

Index	Max speed-up (r_s)	Average speed-up (r_s)
CAGRA	-0.26	+0.69
NSG	-0.79	-1.00
Vamana	-0.95	-0.91
NN-Descent	-0.91	-0.98

5.4 Evaluation on datasets with different dimensionalities (Q4)

To investigate how vector dimensionality influences the effectiveness of graph node reordering, we conduct experiments on randomly generated datasets with varying dimensionalities. Specifically, we use datasets with dimensionalities $d \in \{8, 16, 32, 64, 128, 256, 512, 1024\}$. We generate query sets from a random seed different from the one used for the database, since query distributions may differ from the database in real-world settings. For each configuration, we measure the maximum and average speed-up compared to the baseline without reordering.

Figure 9 plots the maximum and average QPS improvement as a function of vector dimensionality. Additionally, we compute Spearman’s rank correlation coefficients to quantify the strength

of monotonic relationships between dimensionality and speed-up effectiveness. Table 2 summarizes the results.

We find a consistent negative correlation between dimensionality and reordering effectiveness for NSG, Vamana, and NN-Descent. Specifically, the correlation coefficients for maximum QPS improvement range from -0.79 to -0.95, and the coefficients for the average QPS improvement range from -0.91 to -1.00. Both exhibit strong negative correlations. In contrast, CAGRA shows no consistent correlation. Interestingly, in some cases, the average speed-up even increases with higher d .

6 Lessons Learned

We summarize our findings by connecting them to the research questions posed in Sec. 5.

Topological differences between graphs are not the main factor affecting search throughput on a GPU (Q1). Figure 5 reveals that NSG and Vamana, despite being originally designed for CPU execution, consistently achieve recall - QPS trade-offs comparable to or even surpassing the GPU-optimized CAGRA across a diverse range of datasets. This trend holds across various vector dimensionalities and database sizes, encompassing both traditional benchmarks and modern high-dimensional embeddings. Interestingly, while the graph statistics of NSG and Vamana, such as the number of edges and connectivity structures, differ significantly from those of CAGRA, their recall-QPS trade-offs on GPU are often

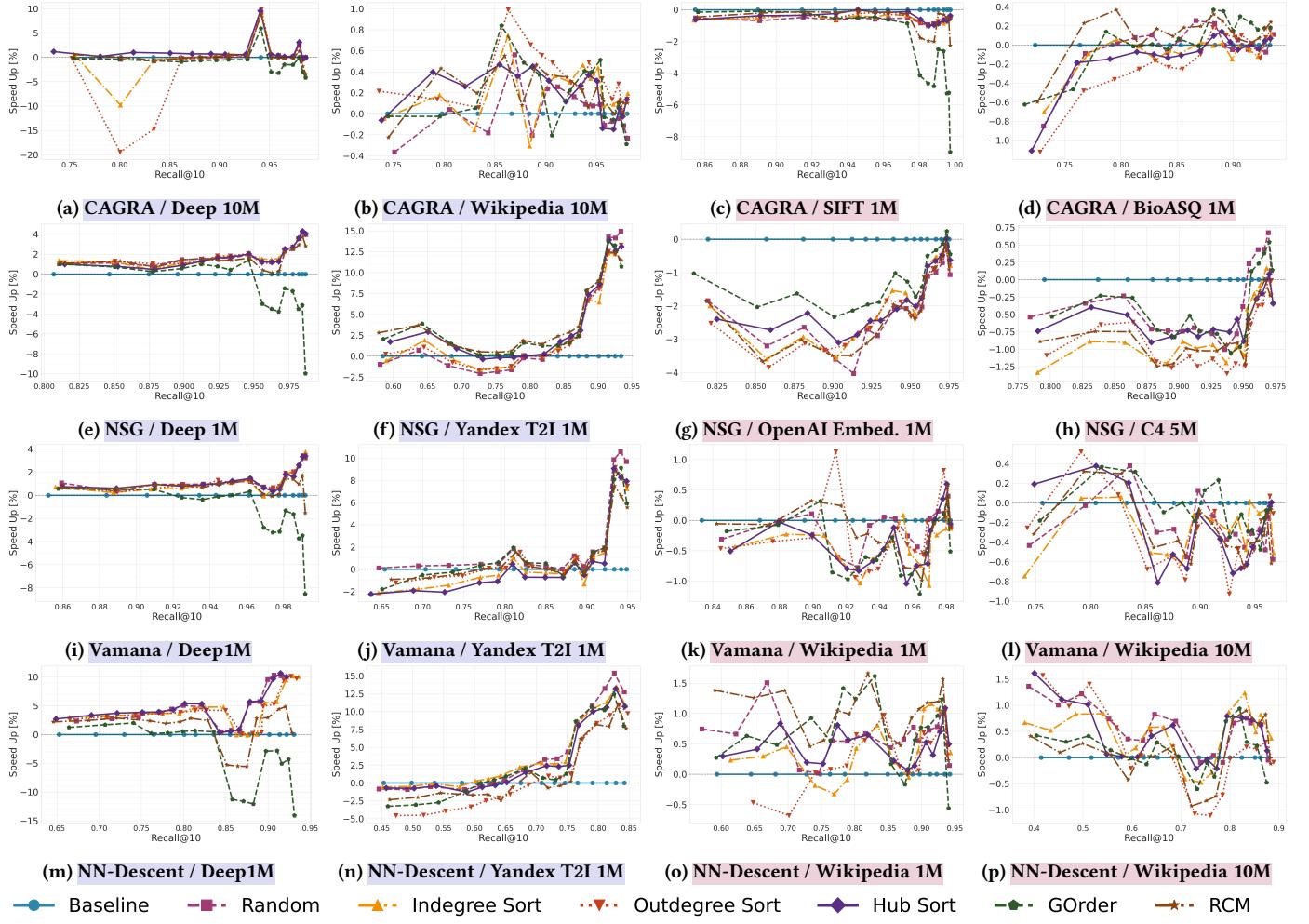


Figure 6: Performance impact of graph node reordering across various datasets and graph-based indices. Blue-labeled plots (left two columns) highlight configurations where reordering yielded substantial improvements in QPS, while red-labeled plots (right two columns) show cases with minimal or no improvement. Each plot shows the Recall - Speed-up trade-off with and without node reordering.

similar. This finding indicates that topological differences between graph indices are not the main factor affecting search throughput on a GPU. We hypothesize this is due to the fundamental architectural difference between CPU and GPU execution. In GPU architecture, nodes are traversed with massive parallelism during traversal, and memory access patterns become the primary performance bottleneck. In such scenarios, other factors, such as coalescing access and warp-level efficiency, may outweigh the influence of graph topology. These results suggest that performance prediction and optimization for GPU-based ANNS should account for low-level hardware layout interactions rather than focusing solely on the graph topology design.

Reordering improves GPU search speed but data-dependent (Q2). Our experiments demonstrate that graph reordering can significantly improve traversal speed on GPU, with up to 15% QPS

gain observed in some index-dataset combinations without loss in recall. This confirms that reordering is a promising, non-intrusive optimization technique for graph-based ANNS on a GPU. While promising, the reordering effectiveness is highly dataset-dependent. Specific datasets such as Deep10M and Yandex T2I consistently benefit from reordering. Others like SIFT or Wikipedia show marginal or even negative improvements. This highlights that reordering effectiveness is not guaranteed across all settings. Each reordering method exhibits varying levels of stability for graph-based ANNS. GOrder often degrades performance in high-recall regions, while Hub Sort, Degree Sort, and RCM are more consistent but not universally superior. This indicates that the choice of reordering algorithm should consider both graph structure and dataset characteristics. Note that speed-ups are sensitive to runtime parameters, particularly the size of the priority queue L used in graph traversal. In some configurations, the impact of reordering varies drastically

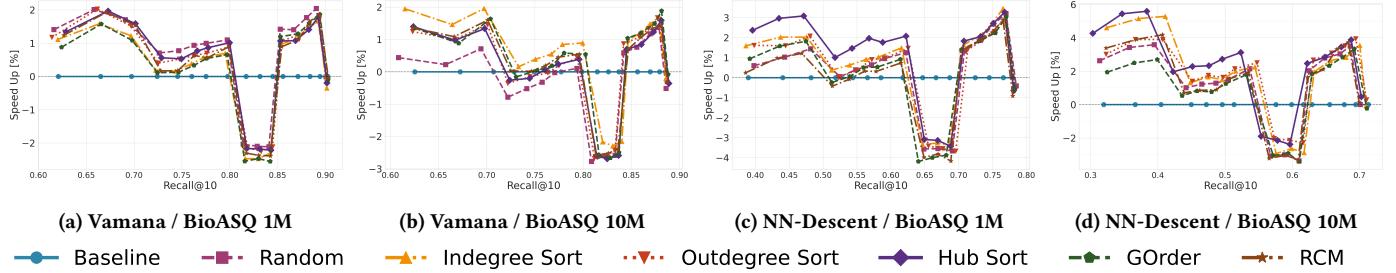


Figure 7: Speed-up characteristics of different reordering algorithms on BioASQ 1M and 10M datasets, using Vamana and NN-Descent indices. Each line shows the QPS improvement over the baseline across different recall levels. We can observe that the patterns of curves are similar between the two different indices.

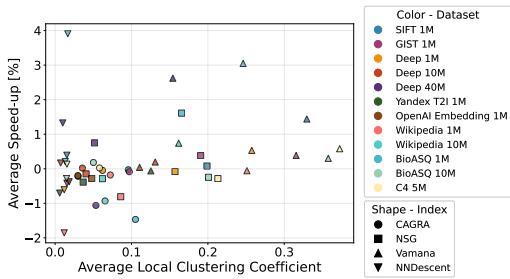


Figure 8: The relationship between the average Local Clustering Coefficient and the average speed-up for each combination of a dataset and an index.

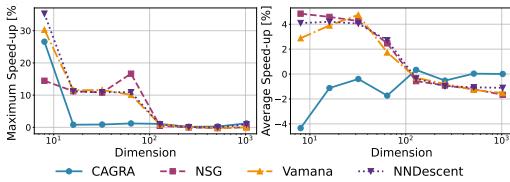


Figure 9: The maximum and average speed-up for randomly generated datasets with various dimensionalities.

depending on this setting, suggesting complex interactions between memory layout and GPU runtime behavior. This indicates that performance is influenced not only by static graph structure, but also by dynamic traversal characteristics and hardware-specific factors such as memory coalescing.

Reordering effectiveness does not necessarily correlate with strong community structure (Q3). While prior reordering studies targeting classical graph algorithms or GNNs suggest that stronger community structures improve effectiveness [26], our graph-based ANNS on GPU workloads yield a different insight. We find no clear correlation between average LCC and QPS improvement. Unlike conventional graph workloads, graph-based ANNS associates each node with a high-dimensional vector. Our results indicate that conventional structural metrics like community strength do not

reliably predict reordering effectiveness in GPU-based ANNS, due to these fundamental differences in problem settings.

The vector dimensionality affects reordering effectiveness (Q4). We observe that the effectiveness of graph reordering is strongly influenced by the dimensionality of the vector space. In our experiments with synthetic datasets ranging from $d = 8$ to 1024, we find a consistent negative correlation between dimensionality and QPS improvement for NSG, Vamana, and NN-Descent. As the dimensionality increases, the speed-up benefit of reordering decreases across these indices. As shown in Tab. 2, this trend is supported by high-magnitude negative correlation coefficients. In contrast, CAGRA shows no such trend; in some cases, the average speed-up even increases with higher d . These findings suggest that reordering is generally more effective in low-dimensional settings for CPU-oriented graph indices, whereas GPU-native indices like CAGRA maintain stable performance regardless of vector dimensionality.

7 Conclusion and Future Work

This work presents the first comprehensive evaluation of graph reordering effectiveness for graph-based ANNS on a GPU. Our unified evaluation framework eliminates implementation bias across different graph indices, enabling fair matrix evaluation of reordering algorithms across heterogeneous graph topologies. Our experimental results demonstrate up to 15% QPS improvements for certain indices while preserving search accuracy.

While our work focuses on cross-index evaluation of reordering effectiveness, our framework also serves as an execution platform for GPU-optimized graph-based ANNS. We expect this work to provide a sound basis for future graph index research and GPU-based ANNS system development. Although we did not conduct a detailed analysis of GPU architecture-specific performance factors, our study deliberately limits its scope to architecture-independent insights to maintain generality. Our framework establishes a foundation for automated memory layout optimization and serves as a universal execution platform for graph-based ANNS research. Future work includes developing intelligent reordering algorithm selection based on graph properties and advancing novel index topologies optimized for GPU memory hierarchies.

References

- [1] Alexandre Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) (*NIPS’15*). MIT Press, Cambridge, MA, USA, 1225–1233.
- [2] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data* 3, 1, Article 43 (Feb. 2025), 31 pages. doi:10.1145/3709693
- [3] Vignesh Balaji and Brandon Lucia. 2018. When is Graph Reordering an Optimization? Studying the Effect of Lightweight Graph Reordering Across Applications and Input Graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 203–214. doi:10.1109/IISWC.2018.8573478
- [4] Dmitry Baranckuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part XII* (Munich, Germany). Springer-Verlag, Berlin, Heidelberg, 209–224. doi:10.1007/978-3-030-01258-8_13
- [5] Benjamin Coleman, Santiago Segarra, Alex Smola, and Anshumali Shrivastava. 2022. Graph reordering for cache-efficient near neighbor search. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) (*NIPS’22*). Curran Associates Inc., Red Hook, NY, USA, Article 2789, 13 pages.
- [6] NVIDIA Corporation. 2023. cuVS: Vector Search and Clustering on the GPU. <https://github.com/rapidsai/cuvs>.
- [7] E. Cuthill and J. McKee. 1969. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference (ACM ’69)*. Association for Computing Machinery, New York, NY, USA, 157–172. doi:10.1145/800195.805928
- [8] 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* (Hyderabad, India) (*WWW’11*). Association for Computing Machinery, New York, NY, USA, 577–586. doi:10.1145/1963405.1963487
- [9] Matthijs Douze, Alexandre Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvassy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).
- [10] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules against data series similarity search. *Proceedings VLDB Endowment* 15, 10 (June 2022), 2005–2018.
- [11] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graphs. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.
- [12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB ’99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 518–529.
- [13] Han Hu, Jiye Qiu, Hongzhi Wang, Bin Liang, and Songling Zou. 2024. DIDS: Double Indices and Double Summarizations for fast similarity search. *Proceedings VLDB Endowment* 17, 9 (May 2024), 2198–2211.
- [14] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-Excitation Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7132–7141. doi:10.1109/CVPR.2018.00745
- [15] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) (*CIKM ’13*). Association for Computing Machinery, New York, NY, USA, 2333–2338. doi:10.1145/2505515.2505665
- [16] Iris A. M. Huijben, Matthijs Douze, Matthew Muckley, Ruud J. G. Van Sloun, and Jakob Verbeek. 2024. Residual quantization with implicit neural codebooks. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) (*ICML’24*). JMLR.org, Article 830, 18 pages.
- [17] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harshavardhan Simhadri, and Sheshansh Agrawal. 2022. OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. *arXiv* (2022).
- [18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
- [19] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.
- [20] V Karthik, Saim Khan, Somesh Singh, Harsha Vardhan Simhadri, and Jyothi Vedurada. 2025. BANG: Billion-Scale Approximate Nearest Neighbour Search using a Single GPU. *IEEE Transactions on Big Data* (2025).
- [21] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.
- [22] Anastasia Krithara, Anastasios Nentidis, Konstantinos Bougiatotis, and Georgios Paliouras. 2023. BioASQ-QA: A manually curated corpus for Biomedical Question Answering. *Sci. Data* 10, 1 (March 2023), 170.
- [23] 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.
- [24] Yusuke Matsui. 2025. LotusFilter: Fast Diverse Nearest Neighbor Search via a Learned Cutoff Table. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 30430–30439.
- [25] Yusuke Matsui, Martin Aumüller, and Han Xiao. 2023. CVPR2023 Tutorial on Neural Search in Action.
- [26] Nikolai Merkel, Pierre Toussing, Ruben Mayer, and Hans-Arno Jacobsen. 2024. Can graph reordering speed up graph neural network training? An experimental study. *Proceedings VLDB Endowment* 18, 2 (Oct. 2024), 293–307.
- [27] Jiongkang Ni, Xiaoliang Xu, Yuxiang Wang, Can Li, Jiajie Yao, Shihai Xiao, and Xuecang Zhang. 2023. DiskANN++: Efficient Page-based Search over Isomorphic Mapped Graph Index using Query-sensitivity Entry Vertex. *arXiv* (2023).
- [28] Yutaro Oguri and Yusuke Matsui. 2023. General and Practical Tuning Method for Off-the-Shelf Graph-Based Index: SISAP Indexing Challenge Report by Team UTokyo. In *Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña, Spain, October 9–11, 2023, Proceedings* (Coruña, Spain). Springer-Verlag, Berlin, Heidelberg, 273–281. doi:10.1007/978-3-031-46994-7_23
- [29] Yutaro Oguri and Yusuke Matsui. 2024. Theoretical and empirical analysis of adaptive entry point selection for graph-based approximate nearest neighbor search. *arXiv* (2024).
- [30] Naoki Ono and Yusuke Matsui. 2023. Relative NN-descent: A fast index construction for graph-based approximate nearest neighbor search. In *Proceedings of the 31st ACM International Conference on Multimedia*. ACM, New York, NY, USA, 1659–1667.
- [31] 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.
- [32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank Citation Ranking : Bringing Order to the Web. In *The Web Conference*. <https://api.semanticscholar.org/CorpusID:1508503>
- [33] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *VLDB J.* 33, 5 (Sept. 2024), 1591–1615.
- [34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. <http://jmlr.org/papers/v21/20-074.html>
- [35] David Rau, Shuai Wang, Hervé Déjean, Stéphane Clinchant, and Jaap Kamps. 2025. Context Embeddings for Efficient Answer Generation in Retrieval-Augmented Generation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining* (Hannover, Germany) (*WSDM ’25*). Association for Computing Machinery, New York, NY, USA, 493–502. doi:10.1145/3701551.3703527
- [36] Yandex Research. 2021. Benchmarks for Billion-Scale Similarity Search. <https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search>.
- [37] Micha Sharir. 1981. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications* 7, 1 (1981), 67–72.
- [38] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.
- [39] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalene Dobson Manohar, Dmitry Baranckuk, Edo Liberty, Frank Liu, Ben Landrum, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weigu Zheng, Zihao Wan, Jin Yin, and Ben Huang. 2024. Results of the Big ANN: NeurIPS’23 competition. *arXiv* (2024). arXiv:2409.17424 [cs.IR]. <https://arxiv.org/abs/2409.17424>
- [40] Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, Gopal Srinivas, Suhas Jayaram Subramanya, Andrija Antonijevic, Dax Pryce, David Kaczyński, Shane Williams, Siddarth Gollapudi, Varun Sivashankar, Neel Karia, Aditi Singh, Shikhar Jaiswal, Neelam Mahapatro, Philip Adams, Bryan Tower, and Yash Patel. 2023. DiskANN: Graph-structured Indices for Scalable, Fast, Fresh and Filtered Approximate Nearest Neighbor Search. <https://github.com/Microsoft/DiskANN>
- [41] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2019. *DiskANN: fast accurate billion-point nearest neighbor search on a single node*. Curran Associates Inc., Red Hook, NY, USA.
- [42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9. doi:10.1109/CVPR.2015.7298594
- [43] Bing Tian, Haikun Liu, Yuhang Tang, Shihai Xiao, Zhuohui Duan, Xiaofei Liao, Xuecang Zhang, Junhua Zhu, and Yu Zhang. 2025. Fusionanns: An efficient cpu/gpu cooperative processing architecture for billion-scale approximate nearest neighbor search. In *USENIX*.

- [44] George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artières, Axel-Cyrille Ngonga Ngomo, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Palioras. 2015. An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics* 16, 1 (April 2015), 138.
- [45] Jiaoguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, 2614–2627.
- [46] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.
- [47] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *Nature* 393 (1998), 440–442. <https://api.semanticscholar.org/CorpusID:3034643>
- [48] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. 2016. Speedup Graph Processing by Graph Ordering. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) (*SIGMOD ’16*). Association for Computing Machinery, New York, NY, USA, 1813–1828. doi:10.1145/2882903.2915220
- [49] Jiuchi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (2024), 2241–2254.
- [50] Chenyuan Wu, Ninglu Shao, Zheng Liu, Shitao Xiao, Chaozhuo Li, Chen Zhang, Senzhang Wang, and Defu Lian. 2025. Lighter And Better: Towards Flexible Context Adaptation For Retrieval Augmented Generation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining* (Hanover, Germany) (*WSDM ’25*). Association for Computing Machinery, New York, NY, USA, 271–280. doi:10.1145/3701551.3703580
- [51] Artem Babenko Yandex and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2055–2063. doi:10.1109/CVPR.2016.226
- [52] Yunning Zhang, Vladimir Kiriansky, Charith Mendis, Saman Amarasinghe, and Matei Zaharia. 2017. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 293–302.
- [53] Zilliz. 2023. VectorDBBenchmark. <https://github.com/zilliztech/VectorDBBenchmark>.