RED-ANNS: An RDMA-Enabled Distributed Framework for Graph-Based Approximate Nearest Neighbor Search

Yue Chen
Fudan University
yuechen23@m.fudan.edu.cn

Kai Zhang*
Fudan University
zhangk@fudan.edu.cn

Sipeng Chen
Fudan University
sipengchen22@m.fudan.edu.cn

Shihai Xiao
Huawei Technologies Co., Ltd
xiaoshihai@huawei.com

Xiaomin Zou
Huawei Technologies Co., Ltd
zouxiaomin1@huawei.com

Ren Ren
Huawei Technologies Co., Ltd
renren1@huawei.com

Yinan Jing
Fudan University
jingyn@fudan.edu.cn

X. Sean Wang
Fudan University
xywangcs@fudan.edu.cn

Li Cao
Huawei Technologies Co., Ltd
xyzcaoli@outlook.com

Mingxiang Wan
Huawei Technologies Co., Ltd
wanmingxiang1@huawei.com

ABSTRACT

Unstructured data, such as text and images, are converted into high-dimensional vectors to capture their semantics for effective data retrieval. Approximate Nearest Neighbor Search (ANNS) over these vectors has become a fundamental technique in many domains, including retrieval-augmented generation and recommendation systems. With an ever-increasing volume of data, existing distributed solutions typically segment data across multiple machine nodes, handling query processing in a MapReduce-style approach. However, this approach suffers from reduced indexing efficiency and increased computational overhead, resulting in limited performance enhancement despite investing several times more resources.

In this work, we propose RED-ANNS, a distributed ANNS approach on an RDMA network. The core idea is to maintain a logically full graph across a shared memory space of multiple nodes and utilize Remote Direct Memory Access (RDMA) to search the distributed graph, thereby avoiding the reduction in indexing efficiency caused by segmentation. The key to making this approach effective is to address the overhead associated with remote accesses. We reduce remote access frequency through locality-aware data placement and affinity-based query scheduling, while we hide remote access latency with a dependency-relaxed best-first search algorithm. Extensive experiments demonstrate that RED-ANNS achieves a performance improvement of up to 2.5× over MapReduce-style approaches and up to 5.3× over open source vector databases.

PVLDB Reference Format:
Yue Chen, Kai Zhang, Sipeng Chen, Shihai Xiao, Xiaomin Zou, Ren Ren, Yinan Jing, X. Sean Wang, Li Cao, and Mingxiang Wan. RED-ANNS: An RDMA-Enabled Distributed Framework for Graph-Based Approximate Nearest Neighbor Search. PVLDB, 19(3): 399 - 412, 2025.
doi:10.14778/3778092.3778101

*Dr. Kai Zhang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 3 ISSN 2150-8097.
doi:10.14778/3778092.3778101

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/cheenyuee/RED-ANNS.git.

1 INTRODUCTION

Encoding unstructured data into high-dimensional vector representations using embedding models has become a critical way of efficient data retrieval and management [36, 53]. Extensive research has been conducted to improve vector retrieval performance by designing efficient Approximate Nearest Neighbor Search (ANNS) indexes, which have been widely used in search engines [7, 8, 54], recommendation systems [9, 34], and intelligent question answering [30, 39]. Moreover, popular large language models (LLMs) often utilize the retrieval-augmented generation (RAG) framework to access external knowledge libraries, retrieving contextually relevant information to enhance model outputs with vector search [2, 32, 46, 56].

As application scenarios become increasingly diverse and complex, the volume of vectors has been expanding at an unprecedented rate. For example, a monitoring system would segment each frame of the camera video stream and embed each object that appeared in the image frame. This results in searching billions of embedding vectors to locate a specific object or incident. For such workloads, the required storage often reaches the terabyte scale, exceeding the memory capacity of conventional servers [42]. Some persistent storage-based ANNS approaches only require small memory space for large datasets, but suffer 4.2-6.4× performance degradation due to associated I/O costs [8, 22, 24]. As business scenarios continue to evolve, they pose more stringent requirements for query latency, which can only be met with memory-based ANNS solutions.

To perform large-scale in-memory vector searches, existing solutions adopt a segmentation-based approach for data storage and utilize a MapReduce-style execution framework [55] to perform ANNS

in distributed environments. Specifically, this approach partitions the dataset into multiple segments and distributes the segments across multiple nodes, with indexes built offline for each segment independently [14, 19, 47, 55]. During query processing, queries are broadcast to all nodes in the cluster. Each node searches its local segments and retrieves a local top-k result for each segment (map). The local results from all nodes are then aggregated to produce the final top-k result (reduce). This approach can leverage the memory space and computational resources of multiple nodes for parallel execution. However, the introduction of segmentation significantly reduces indexing efficiency and increases computational overhead during query execution, resulting in a sharp drop in overall system throughput. For example, even though graph-based ANNS indexes exhibit superior performance over other indexes and have been widely adopted for high-performance ANNS [33, 48], partitioning the dataset into four segments and building a separate graph index for each can incur a 2.5× performance degradation compared to searching on a single graph index [52]. This is primarily due to the fact that there is no linear reduction in search hops on a subgraph compared to a full graph. In our experiments, searching on a $\frac{1}{4}$ subgraph would only lead to 40% fewer hops. In other words, 4× more computational resources are used to achieve only 1.6× performance improvement, which is extremely inefficient for scaling out.

In this work, we propose RED-ANNS, an RDMA-enabled distributed framework that significantly enhances resource efficiency and scalability for graph-based ANNS. To avoid the reduction in indexing efficiency of existing distributed approaches, RED-ANNS fully leverages the advantages of a distributed full graph index, which is placed across multiple machine nodes, with edges indicating local access or remote access. With the RDMA network, processes on a node can directly access a part of the graph on another node. Compared to existing approaches, the full graph search enables a lower computation overhead while maintaining the same retrieval accuracy. However, the primary challenge to achieving potential performance gains is reducing the frequency and overhead of remote accesses involved in the search process. To address this, RED-ANNS reduces remote accesses by designing locality-aware data placement and affinity-based query scheduling, and based on the characteristics of ANNS and the RDMA network, RED-ANNS proposes a dependency-relaxed best-first search algorithm that incorporates data prefetching to hide remote access latency. Through the proposed techniques, RED-ANNS effectively performs ANNS in distributed settings with significantly enhanced resource efficiency.

The main contributions of this work are summarized as follows:

- We propose a distributed graph-based ANNS framework for large-scale vector search, which delivers significantly improved computational resource efficiency.
- We design locality-aware data placement and affinity-based query scheduling strategies to enhance data access locality, thereby reducing remote data accesses in ANNS.
- We propose a dependency-relaxed best-first search algorithm that introduces data prefetching to hide remote data access latency in distributed graph search.
- We implement a prototype of RED-ANNS. Experimental evaluations demonstrate that RED-ANNS achieves up to 2.5× performance improvement over existing distributed solutions.

RED-ANNS is an efficient distributed in-memory ANNS solution with superior scalability, expected to yield significantly improved resource efficiency for large-scale vector search.

2 BACKGROUND

2.1 Approximate Nearest Neighbor Search

Definition 2.1. Given a dataset of N vectors with dimension D, denoted as $\mathcal{X} = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^D$, a query vector $q \in \mathbb{R}^D$, and a distance function $\delta(\cdot, \cdot)$, the top-k ($k \leq N$) NNS problem is to find a subset $S_k \subseteq \mathcal{X}$ containing the k closest vectors to q, defined as:

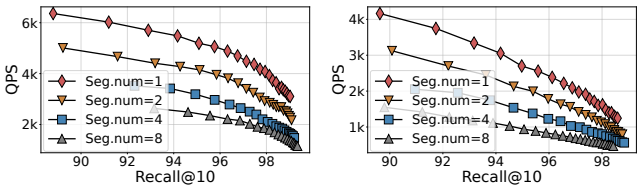
$$S_k = \arg \min_{S \subseteq \mathcal{X}, |S|=k} \sum_{x \in S} \delta(x, q) \tag{1}$$

Nearest Neighbor Search (NNS) is used to identify the top-k feature vectors in a database that are closest to a given query vector, based on predefined distance metrics such as the Euclidean distance, as shown in Definition 2.1. However, exhaustive search for the exact top-k nearest neighbors is computationally expensive, especially for large-scale and high-dimensional data due to the challenges imposed by the curse of dimensionality [21, 42]. To improve retrieval performance, extensive research has been conducted to design efficient Approximate Nearest Neighbor Search (ANNS) indexes, which achieve significant performance improvements by tolerating a small loss in accuracy. These studies mainly include graph-based indexes [16, 24, 33, 35], inverted file indexing [4, 13, 41, 44, 51], and quantization [3, 18, 20, 25, 26, 50]. The search performance of ANNS will be evaluated through the speed-vs-recall tradeoff, using throughput and latency to assess retrieval speed, and recall rate to evaluate retrieval accuracy. Given an approximate top-k result set S'_k and the exact result set S_k, the ANNS recall is defined as $Recall@k = |S'_k \cap S_k|/|S_k|$.

Extensive research indicates that graph-based ANNS methods achieve state-of-the-art performance on a variety of datasets, such as HNSW [33] and Vamana [24]. The core idea of graph-based methods is to represent feature vectors as vertices in a graph and to establish edges between them based on distance relationships, forming a neighborhood graph. During the search phase, the Best-First Search (BFS) algorithm [48] traverses the graph to locate potential result vertices. The traversal starts from an entry point and employs a greedy strategy to iteratively select the closest unchecked vertex to the query from the candidate pool for neighbor expansion, updating the pool with newly visited neighbors. This process continues until the convergence condition is met, that is, when no unchecked candidates remain. Since graphs are built on neighbor relationships, they typically require fewer vector distance calculations compared to other solutions [48], leading to higher query performance.

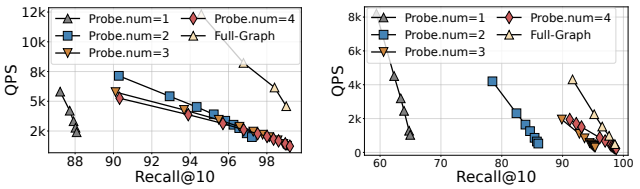
2.2 Limitation of Existing Distributed Solutions

Existing distributed ANNS solutions partition database vectors into segments across multiple nodes and build graph indexes for each segment separately [1, 12, 19, 31, 47, 49], enabling sub-graph parallel search (sub-GPS) [52] within a MapReduce-style execution framework. For example, the popular vector database Milvus [19, 47] partitions data into segments of up to 1 GB by default. Queries are then dispatched to nodes that hold segments of the searched collection. After each node performs ANNS on its local segments,



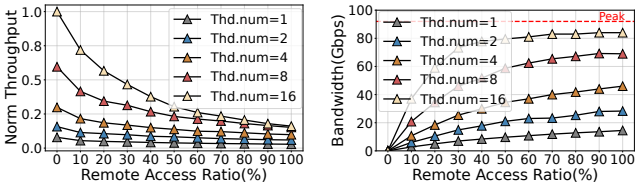
(a) DEEP100M (b) MS-Turing100M

Figure 1: Performance impact by graph segmentation



(a) DEEP10M (b) MS-Turing10M

Figure 2: Performance and accuracy impact with varying number of probed segments



(a) Throughput performance (b) Bandwidth consumption

Figure 3: Performance impact on full-graph based distributed ANNS approach with varying ratio of remote accesses

the results are aggregated to derive the global top-k results. To control the total number of segments, smaller segments can be merged into larger ones and index rebuilding is performed [47].

For the sub-GPS solutions, maintaining separate graph indexes for each segment allows local query execution with minimal inter-node communication overhead (except for the final aggregation). However, this method fragments the ideal connected graph into multiple isolated subgraphs, resulting in the loss of neighbor information between subgraphs and reduced indexing efficiency [52]. During query execution, multiple BFS traversals are required to find the nearest neighbors due to lack of connectivity, increasing computational overhead, especially when segments are small and numerous. Figure 1 compares the performance of Milvus on 100-million scale datasets in a four-node setup (hardware configuration detailed in §7.1), using one, two, four and eight segments per node, with each segment indexed by the built-in HNSW [47]. As the number of segments per node increases from 1 to 4, the performance degrades by an average of 2.1× at 0.95 recall, and further declines to 2.9× with eight segments, demonstrating a reduced efficiency due to smaller and more numerous segments. Distance calculation statistics show that four subgraphs require an average of 2.4× more distance calculations, increasing to 3.5× with eight segments, illustrating why sub-GPS solutions suffer performance degradation, a phenomenon also verified in existing work [52].

The above experiments demonstrate that performing ANNS in memory segments across all nodes would result in significant performance degradation. Some research proposes only probing a portion of segments based on the clustering property of the nearest neighbors [6, 11, 45]. However, our experiments show that this strategy can be detrimental to query accuracy. Figure 2 presents the single node performance on ten-million scale dataset when the dataset is partitioned into four segments using the balanced k-means algorithm [10], with an HNSW [33] index built for each segment. On the DEEP dataset, probing the three most relevant segments can achieve a recall of above 0.99. However, by probing two segments, the recall drops to a maximum of 0.97. In contrast, on the MS-Turing dataset, the accuracy loss is more pronounced due to its more dispersed groundtruth distribution. Achieving a recall of up to 0.99 requires searching all segments, whereas the recall only reaches 0.95 and 0.86 with probing three and two segments, respectively. In production environments that generally have strict requirements for high accuracy, searching only for partial segments can barely reach the goal. Moreover, the figure also demonstrates the performance advantage of full-graph based ANNS, which consistently delivers both high throughput and high recall.

3 RED-ANNS: AN OVERVIEW

3.1 Core Idea

Given the limitations faced by existing distributed solutions in performance and scalability, we propose RED-ANNS, an RDMA-enabled distributed ANNS approach. Instead of segmenting the vectors among distributed nodes and performing MapReduce-style queries, RED-ANNS adopts a full-graph parallel search (full-GPS) strategy by maintaining a logical full graph over the entire database vectors in distributed memory to fully leverage the connectivity of graph index for improved retrieval efficiency. To enable fine-grained cross-node data access in full-GPS, RED-ANNS utilizes a high-speed RDMA network as its underlying communication infrastructure. RDMA technology enables computers to directly access authorized memory regions on remote nodes, offering a low-latency data access mechanism for distributed systems. RED-ANNS builds a unified memory space across multiple nodes with RDMA, which sets the foundation for a distributed full-GPS implementation.

However, since full-GPS is conducted across the entire graph, it inevitably incurs remote accesses as the graph is placed among multiple nodes. And although the RDMA network provides relatively low remote access latency, excessive remote access would significantly degrade performance and counteract the advantage brought by the full-GPS design. This is because the remote memory access overhead of RDMA is still around 100× higher than that of local memory access [27], which would become a bottleneck in ANNS. To examine the performance impact of remote access, we build a Vamana [24] index on the DEEP100M dataset and deploy a full copy of the data on each of the four nodes (configuration detailed in §7.1). During query execution, we artificially control the ratio of data fetched from local versus remote nodes to simulate performance under different levels of remote access. Figure 3(a)

presents the performance degradation under different remote access ratios, while Figure 3(b) shows the corresponding consumption of network bandwidth. Considering a random access pattern with 75% remote accesses in the four-node setup, with a single thread the performance drops by 60%, indicating that the system is latency-bound at low load. When scaling to 16 threads, performance further drops by 80% as the network approaches saturation, demonstrating that the system becomes bandwidth-bound under high load. These degradations highlight the critical importance of addressing remote access challenges in distributed full-GPS implementations.

3.2 Design Methodology in RED-ANNS

To achieve a better trade-off between sub-GPS, which suffers from computation inefficiency, and full-GPS, which suffers from remote access overhead, RED-ANNS aims to optimize data access without compromising computation efficiency by addressing the remote access overhead in full-GPS in two ways: minimizing the remote access frequency and hiding the access latency. The main techniques proposed in RED-ANNS to achieve the goal are as follows.

1) **Locality-aware data placement.** An ANNS exhibits the property of gradually converging near the target region, and the base vectors it accesses tend to be spatially clustered around the query vector. This property presents an opportunity to reduce remote access by optimizing data placement. However, simply partitioning the base vectors into n clusters for n machine nodes would result in suboptimal performance due to its inconsistency with the graph-building procedure. In RED-ANNS, we reduce remote access of graph traversal by exploiting graph locality and vector similarity. We use similarity as the weights of graph edges and partition the graph with the goal of minimizing cross-node edges and colocating highly similar vertices. Additionally, hot vectors on the boundary across partitions are duplicated to further reduce remote accesses.

2) **Affinity-based query scheduling.** With incoming queries, RED-ANNS routes queries to nodes that have better localities, aligning query execution with optimized data placement. To achieve this, we build an anchor set of vectors to evaluate which node is more appropriate. For an online query, RED-ANNS first identifies primary and secondary candidate nodes and assigns the query to the primary candidate for execution. Targeting the potential load imbalance caused by query assignment and execution latency, RED-ANNS further incorporates an affinity-based work stealing mechanism. When a node has processed all its assigned queries, it steals from the nodes with pending queries, prioritizing the ones whose secondary candidate is itself. This strategy addresses the load imbalance while mitigating locality loss caused by stealing.

3) **Dependency-relaxed best-first search.** Prefetching is a common way to hide memory access latency in ANNS. However, the high remote access latency in RED-ANNS renders the existing prefetching mechanism largely ineffective. To address this, we propose relaxing the best-first dependencies in the graph search order to enable efficient data prefetching. Specifically, for multiple search branches involved in the search process (that is, the candidate pool maintains several vertices to be expanded), RED-ANNS computes neighbor expansions on the current branch while asynchronously fetching the neighbors for the potential following expansion, creating an overlap between computation and data loading. We also

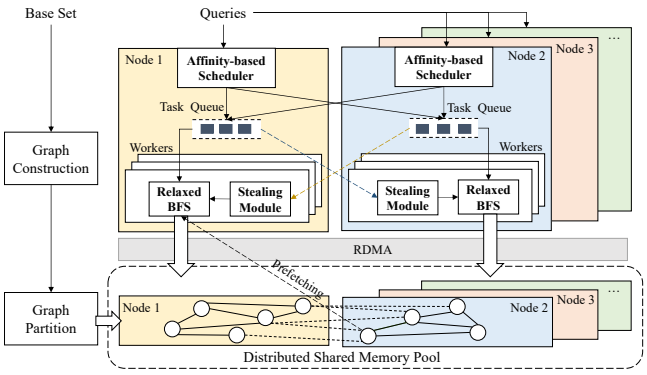


Figure 4: The architecture of RED-ANNS

employ a pruning scheme to reduce unnecessary prefetches. Due to the connectivity of the neighborhood graph, RED-ANNS with the relaxed BFS order can always navigate to the target region with the groundtruths, while a much higher performance is achieved.

3.3 The Architecture of RED-ANNS

Figure 4 illustrates the architecture of RED-ANNS, which operates on a cluster of servers connected via an RDMA network. After the graph is constructed and partitioned offline, the full graph is maintained across all nodes in the cluster, where queries are scheduled and executed in real time based on this full graph.

Building distributed graph. RED-ANNS first constructs a graph index on the entire database vectors in an offline phase. Then we perform graph partitioning on the constructed graph and place partitions on the corresponding nodes to build a distributed graph. To facilitate data access on the distributed graph, an address mapping table is built to map each `vertex_id` to its physical storage location in the shared memory pool, represented as a tuple `(node_id, local_id)`, where `node_id` represents the node storing the vertex, and `local_id` indicates the offset of the vertex within that specific node. To avoid the overhead of table lookups during graph traversal, we further inline the physical location information directly into the neighbor list in the graph. Specifically, each vertex is uniquely identified by the two-dimensional location tuple `(node_id, local_id)`, and every `vertex_id` in the neighbor list is replaced with this tuple. When accessing a vertex, if its `node_id` corresponds to the local node, the data can be accessed directly; otherwise, a remote access work request is issued via the underlying RDMA interface to fetch the data from the corresponding remote node. Additionally, RED-ANNS adds a duplication mechanism to store frequently accessed remote vertices locally, which reduces remote accesses. It is important to note that although the data are physically partitioned across machines, the system still logically maintains a unified graph structure.

Query execution workflow. During online query processing, RED-ANNS manages the scheduling and execution of queries in the cluster. As shown in the figure, queries are routed to the corresponding physical node for execution with *Affinity-based Scheduler*, which chooses the node leading to fewer remote accesses. In query execution, the corresponding node performs the ANNS algorithm

and accesses the distributed graph in the search process. Specifically, for each hop that ANNS searches along the graph, the *Relaxed BFS* module performs query execution with relaxed execution dependency, where the mechanism initiates multiple remote accesses to prefetch potential candidate neighbors in the search path. This relaxed BFS aims at effectively pipelining the high-cost remote I/O with distance calculation. When the task queue of a node becomes empty, the *Stealing Module* fetches queries from heavily loaded nodes to its node, and the work stealing also prioritizes queries with a higher affinity for this node. With the set of techniques, queries are executed efficiently on the cluster with a locality-aware balanced load. This leads to fewer remote accesses and a mitigation of the remote access penalty, which is the key to improving overall resource efficiency for a full-GPS design.

4 DATA PLACEMENT AND QUERY SCHEDULING

To reduce remote access during query execution, we distribute data across multiple machine nodes based on graph locality and schedule queries to the node with the highest affinity.

4.1 Locality-Aware Data Placement

For efficient query execution, we first partition and distribute the base data across multiple machine nodes, providing the opportunity to reduce remote accesses. Clustering algorithms [7, 8, 10] offer a straightforward method for partitioning the vector space into multiple Voronoi regions, grouping nearby vectors together, and assigning them to the same node. However, the clustering mechanism inherently differs from that used in graph index construction, which can cause data accesses during the search to frequently alternate between local and remote nodes due to the graph connectivity.

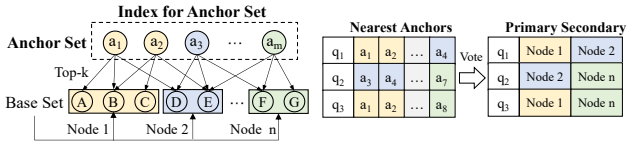
Locality-aware graph partitioning. To improve search efficiency in the distributed graph, we employ a locality-aware graph partitioning strategy that considers both graph structure and vector similarity for efficient data placement. Our graph partitioning has three goals: 1) minimizing cross-node edges without breaking the intrinsic graph structure to enhance data access locality, 2) ensuring a balanced distribution of vertices across physical nodes to facilitate load balancing, and 3) co-locating highly similar vectors at partition boundaries on the same node for higher access efficiency. To preserve the locality while maintaining a balanced load distribution, we adopt a balanced graph partitioning algorithm METIS [28] as the base algorithm to divide the graph into balanced partitions. Considering that METIS cannot understand the vector similarities at partition boundaries, we further assign edge weights based on normalized vector similarity, so that short edges representing high similarity are prioritized to be preserved locally, whereas longer edges can be compromised for cross-node connections. Specifically, for the graph $G(V, E)$, where V and E denote the sets of vertices and edges, respectively, the edge weight between two vertices $u \in V$ and $v \in V$ is defined as:

$$\text{Weight}(u, v) = 1 - \frac{dist(u, v) - dist_{\min}}{dist_{\max} - dist_{\min}} \tag{2}$$

where, $dist(u, v)$ represents the vector distance between the two vertices, and $dist_{\max}$ and $dist_{\min}$ refer to the maximum and minimum distances, respectively, between all connected vertex pairs in the graph. The optimization goal of partitioning the graph into $n \in \mathbb{Z}^+$ parts is formulated as:

$$\min_{\mathcal{P}} \sum_{\substack{(u,v)\in E \\ u\in P_i,\, v\in P_j,\; i\neq j}} \text{Weight}(u, v) \tag{3}$$

where, P_i and P_j represent different partitions of V, with each partition satisfying $P_i \approx \frac{|V|}{n},\ \forall i = 1, \ldots, n$. Finally, the vertices are divided into n balanced parts, each of which is placed as a data shard on one of the n machine nodes.



(a) The anchor set for scheduling **(b) Voting for nodes**

Figure 5: Affinity-based query assignment

Duplication mechanism. To enhance the local availability of vertices near partition boundaries across multiple relevant nodes, we duplicate frequently accessed boundary vertices in local DRAM. To achieve this, we perform offline profiling using a query training set generated by randomly sampling from real queries—for example, historical queries in real-world applications—to capture the data access characteristics. During profiling, we track all remotely accessed vertices and their access frequencies, allowing us to identify hot vertices and duplicate them from their original remote nodes to a local cache with a predefined capacity. During online query processing, when a remote vertex is required, RED-ANNS first checks the local cache. If the vertex is found, remote access is avoided.

4.2 Affinity-Based Query Scheduling

Affinity-based query scheduling consists of two modules. In a given data placement, the query assignment module routes each query to the node with highest affinity, improving data access locality. To address potential load imbalance, the work-stealing module provides runtime load balancing, improving resource utilization.

Affinity-based query assignment. To improve data access locality, we consider the affinity between each query and the data shards distributed across different physical nodes. Since the graph traversal during the search process occurs primarily around the region containing the query's top-k nearest neighbors, we define the affinity between a query and a data shard as the proportion of the query's top-k nearest neighbors that reside in this shard. Specifically, given a dataset $\mathcal{X}$ partitioned into n disjoint shards $P_1, P_2, \ldots, P_n$ such that $\mathcal{X} = \bigcup_{i=1}^{n} P_i$, a query q, and its top-k nearest neighbors S_k within $\mathcal{X}$, the affinity value between query q and shard P_i is defined as:

$$\text{Affinity}(q, P_i) = \frac{|S_k \cap P_i|}{|S_k|} \tag{4}$$

Ideally, each query is assigned to the node that hosts the data shard with the highest affinity, so that most of the vertices accessed during the search process reside locally.

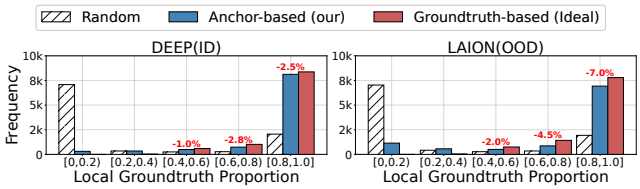


Figure 6: Effectiveness analysis of affinity-based query assignment with in-distribution and out-of-distribution workloads. The x-axis represents the proportion of a query's top-100 neighbors that are located on the assigned node, while the y-axis shows the query frequency (out of 10k queries).

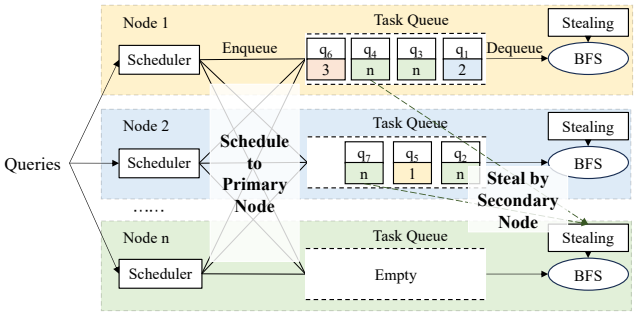


Figure 7: Affinity-based work stealing

However, the actual groundtruth of an online query's top-k nearest neighbors is unknown, making it impossible to compute the exact affinity between the query and the data shards. To enable affinity-based query assignment, we construct a set of anchor vectors that serve as a bridge to help identify the node expected to host the most affine shard for query execution. As shown in Figure 5(a), given a data placement, we first prepare a sample set of base points as an anchor set $\mathcal{A} = \{a_1, \ldots, a_m\}$. For each anchor vector, we identify its top-k nearest neighbors from the base set and compute its affinity for each shard. Then, each anchor is associated with the node that hosts the most affine shard, as formalized by:

$$\text{Anchor}(a_i) = \arg\max_j \text{Affinity}(a_i, P_j) \tag{5}$$

For example, when most nearest neighbors for a_1 are located in Node 1, $\text{Anchor}(a_1) = 1$. As shown in Figure 5(b), to identify the node that is expected to host the most affine shard for an online query vector q, we first retrieve its top-k nearest neighbors from the anchor set $\mathcal{A}$, denoted by $a_1, \ldots, a_k$ (accelerated by an additional graph index over $\mathcal{A}$), and then determine the *Primary Node* using a voting mechanism as follows:

$$\text{PrimaryNode}(q) = \text{mode}(\{\text{Anchor}(a_1), \ldots, \text{Anchor}(a_k)\}) \tag{6}$$

where the *mode* operator calculates the node id that appears the most times. Finally, the query is assigned to the *Primary Node* for execution by pushing it into the task queue of that node.

Figure 6 shows the effectiveness of query assignment achieved with a lightweight 30K-scale anchor set over a 100M-scale base set distributed over four nodes. Our query assignment enables most queries to maintain a high proportion of their top-100 neighbors locally, except for those near partition boundaries, where the top-100 neighbors inherently span multiple nodes. In addition, the anchor vectors also help RED-ANNS in adaptively selecting local entry points during the search process. For example, when q_1 chooses Node 1 as the primary scheduling node in Figure 5(a), the groundtruths of the nearest anchor vector in that node are enqueued into the candidate pool to start the search procedure (that is, A and B in the base set). In this way, the query avoids searching from a fixed or remote entry point but instead attempts to approach the region containing its results using adaptive local entry points.

Affinity-based work-stealing mechanism. While we prefer to execute each query on the node with the highest affinity, in practice, query assignments across nodes may be imbalanced, and queries differ in their convergence difficulty (e.g., varying hop counts), resulting in uneven computational workloads [45]. As a result, some nodes may finish their assigned query tasks earlier than others. Even though the regular work-stealing mechanism can provide runtime load balancing by allowing idle nodes to take on queries with higher data affinity to other busy nodes, it comes at the cost of increased remote access due to queries being executed on suboptimal nodes. The affinity-based work-stealing mechanism helps mitigate the increase in remote accesses. To achieve this, as shown in Figure 5(b), we also select a runner-up candidate as the *Secondary Node* when voting during query assignment, and store this information together with the query task. Then, as shown in Figure 7, when load imbalance occurs, the work-stealing module on an idle node attempts to balance the load by selecting a query task from the tail of the overloaded task queue, prioritizing tasks whose *Secondary Node* matches itself, and migrates it to execute the query. If no such matching query is found, it directly selects the task at the tail of the queue.

5 DEPENDENCY-RELAXED BEST-FIRST SEARCH

To hide the network latency associated with remote access, we introduce a dependency-relaxed BFS algorithm with an efficient data prefetching mechanism that enables the overlap of computation and I/O for distributed query execution in RED-ANNS.

5.1 Why Existing Prefetching Mechanisms Fail

Existing memory-based indexing systems improve memory access efficiency by leveraging C++ prefetching instructions [15, 16, 33]. When a vertex from the candidate pool is selected for neighbor expansion, the distances between the query and all of its neighbors must be computed. In typical graph construction algorithms, the degree of vertex ranges from 32 to 128. During batched neighbor distance computations, a prefetch instruction is issued prior to each computation to proactively load the data required for the next calculation into the CPU cache. This prefetching mechanism has proven effective in memory-based ANNS indexes, such as HNSW[33].

However, although this prefetching mechanism is effective in single-node memory-based ANNS indexes, it becomes less effective in distributed settings. In RDMA networks, a read can introduce a round-trip latency of several microseconds, while a distance computation takes only tens of nanoseconds. The latency of loading vector data from remote nodes is two orders of magnitude higher

than the computation time. Consequently, only prefetching the next neighbor, as in the existing prefetching mechanism, cannot achieve an effective temporal overlap between data loading and computation, and remote accesses cause the CPU to idle during waiting I/O, resulting in a significant performance degradation.

5.2 Execution Dependency Analysis

In the BFS search process, all operations form a sequential chain composed of multiple hops, with each hop expanding the neighbors of a candidate vertex. As shown in Figure 8(a), the execution of a single hop consists of several fine-grained subtasks[23, 48]: (i) *Graph Read* that reads the neighbor list of the selected vertex, (ii) *Graph Traverse* that scans the neighbors of the vertex, (iii) *Vector Read* that reads the feature vector of each neighbor, (iv) *Distance Calculation* that calculates the distance between the query and each neighbor, (v) *Candidate Update and Next Vertex Select* that updates the candidate pool and chooses the candidate with the lowest distance to the query for the next hop. There are two types of dependencies in the process. Within each hop, data dependencies require subsequent subtasks to wait until the previous ones are completed before execution can proceed, which we call intra-hop data dependency. Between adjacent hops along the search path, the BFS must wait for the completion of the neighbor expansion at the previous hop to compute the best candidates for the next hop, which is referred to as an inter-hop requirement of the best-first greedy strategy. The combination of intra-hop data dependencies and inter-hop strategy requirements creates a tightly coupled execution dependency.

The chained execution dependencies severely impact search performance in distributed settings. As shown in Figure 8(a), given the inherently high I/O latency of read operations due to network transmission, the sequential nature of the execution limits the concurrency between computation and I/O, resulting in low CPU utilization and high query latency. An optimized memory layout can reduce the round-trip waiting time required for each hop, as shown in Figure 8(b). Since vertices involved in distance computations may be selected to expand its neighbors further in future iterations, colocating each vertex's vector data and neighbor list in memory allows both the vector and its associated neighbor list to be fetched together as a single vertex data unit (referred to as *Vertex Read*) through a single RDMA read. However, the latency of *Vertex Read* still causes significant performance degradation and cannot be effectively hidden due to the chained execution dependencies.

5.3 Dependency-Relaxed Best-First Search

To enable concurrency between computation and I/O, we relax the best-first requirement in BFS. We first introduce how one-hop relaxed BFS works and then extend it to multi-hop relaxed BFS. To reduce network traffic, we also incorporate a quantization-based pruning technique that reduces unnecessary data prefetching.

One-hop dependency-relaxed BFS. Figure 8(c) illustrates the one-hop relaxed dependency BFS. Instead of waiting for all neighbor expansions in the current hop to complete, we begin prefetching the data required for the next hop's neighbor expansion in advance. Specifically, before starting *Distance Calculation* and *Candidate Update* in the i-th iteration, we preselect a vertex from the current candidate pool for neighbor expansion in the $(i+1)$-th iteration

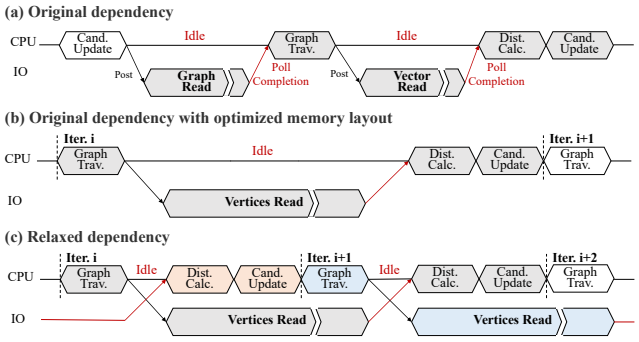


Figure 8: Dependency-relaxed prefetching mechanism. Operations with the same color represent subtasks belonging to the same hop.

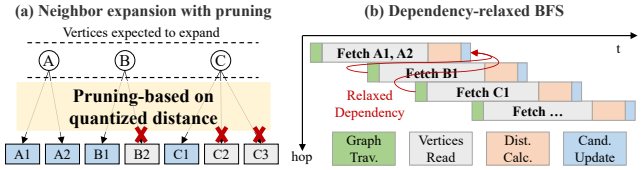


Figure 9: Dependency-relaxed BFS with pruning

and perform *Graph Traverse* to issue asynchronous I/O requests for fetching data of its neighbors, overlapping the computation time of the i-th iteration with the I/O time for the $(i+1)$-th iteration and thus reducing the idle time in the $(i+1)$-th iteration. However, since the prefetch step occurs before the *Candidate Update* in the i-th iteration, the vertex chosen for neighbor expansion in the $(i+1)$-th iteration is independent of the updated queue; it is selected based on the old candidates. Therefore, this scheme relaxes the original best-first requirement in BFS, potentially resulting in a slightly longer search path in the graph than that of the original BFS algorithm.

Multi-hop dependency-relaxed BFS. One-hop relaxed BFS helps to hide part of the network latency compared to the no-prefetching scenario, but the computation time of a single-hop neighbor expansion is still insufficient to fully mask the communication delay. Leveraging the multiple search branches of BFS (that is, the candidate pool contains several vertices to be expanded), we extend one-hop relaxed BFS to an n-hop relaxed BFS strategy. For example, given current candidate vertices A, B, and C in the candidate pool as illustrated in Figure 9(a), we issue asynchronous I/O requests for neighbors of B and C in advance before performing the *Distance Calculation* and *Candidate Update* for the neighbors of A, as shown in Figure 9(b). This allows the selection of expansion vertices for the subsequent two hops to be independent of the *Candidate Update* results of hop A, thus enabling 2-hop relaxed data prefetching to hide more network latency by prefetching earlier and allowing a higher degree of overlap.

Considering that only a small fraction of the expanded neighbors are actually added to the candidate pool, while most are discarded due to being far from the query, a large amount of unnecessary data transfer occurs during the search process. Moreover, breaking

the best-first execution dependency further amplifies this effect. To reduce network traffic, we incorporate product quantization [25] to prune remote neighbors that are unlikely to contribute to the search. Specifically, before the search begins, each machine node stores the compressed quantized codes of all vectors. During the search, for each remote neighbor, we first compute an approximate distance between the query and the remote neighbor based on local quantized codes to estimate the likelihood of being added to the candidate pool. Only neighbors whose quantized distance is less than ϵ times the maximum distance in the current candidate pool are fetched for subsequent precise distance calculation. The parameter ϵ controls the pruning threshold, preventing over-pruning that could degrade search accuracy by removing essential neighbors. As shown in Figure 9, after pruning the neighbors, we only expand vertices B and C with their neighbors B1 and C1, respectively.

5.4 Algorithm Explanation

Our dependency-relaxed BFS algorithm is shown in algorithm 1, incorporating a relaxation parameter n and a pruning threshold ϵ. Relaxed-BFS first initializes a work request queue to temporarily store ongoing RDMA requests, and then adds the entry points to the candidate pool *(Line 1-3)*. *ExpandNeighbors* operation computes distances for a batch of neighbor vectors, updates the candidate pool accordingly, and temporarily maintains the neighbor lists of newly enqueued vertices for use in subsequent neighbor expansions. Relaxed BFS selects the vertex closest to the query from the candidate pool in each iteration, and processes local and remote neighbors separately *(Line 5-7)*. For remote neighbors, the Relaxed BFS first prunes the neighbor set *(Lines 8-12)*, and then issues a *PostRead* operation to asynchronously fetch the necessary vertex data of neighbors, including both the vectors and their neighbor list *(Line 13)*. In a given iteration, computations for remote neighbors are delayed by n iterations to allow sufficient time for I/O completion *(Line 19)*. In contrast, local neighbors are processed immediately to update the candidate pool *(Line 14)*, mitigating the negative impact of longer search paths caused by relaxed dependency. Within each iteration, the algorithm first issues asynchronous I/O requests for remote neighbors before computing local neighbors *(Line 13-14)*. This ordering ensures that even when the relaxation parameter $n = 0$, some latency can still be hidden by adjusting the computation sequence of neighbor vectors. Notably, when updating the candidate pool, newly enqueued vertices may be selected for future neighbor expansion in subsequent iterations. To accommodate this, their neighbor lists, as part of the vertex data, are temporarily stored in the candidate pool. In later iterations, these neighbor lists can be directly retrieved for graph traversal, thereby avoiding redundant data fetches for the neighbor list. For vertices that are not enqueued, their data is immediately discarded, as it will no longer be used.

Algorithm 1: Relaxed Best-First Search

Input: Graph G, entry points P, query q, candidate pool size l, result size k, distance function $\delta(\cdot,\cdot)$, relaxed hop n, pruning factor ϵ
Output: Approximate nearest neighbors of q

```
1  candidatePool S ← ∅
2  workRequestQueue Q ← ∅
3  ExpandNeighbors(S, l, δ(·,·), q, P)
4  while S.HasUnexpandedVertex() do
5      u ← S.ClosestUnexpandedVertex()
6      u.Expanded(true)
7      lB, rB ← divide N_out(u) into local/remote batch
8      for v ∈ rB do
9          if PQ.Compute(v, q) > ε * S.Back().distance then
10             rB.Remove(v) // prune remote neighbors
11         end
12     end
13     PostRead(rB), Q.Push(rB) // async read data
14     ExpandNeighbors(S, l, δ(·,·), q, lB)
15     if n > 0 then
16         n ← n − 1, continue
17     end
18     rB ← Q.Pop(), PollCompletion(rB) // wait I/O
19     ExpandNeighbors(S, l, δ(·,·), q, rB)
20 end
21 while !Q.Empty() do
22     rB ← Q.Pop(), PollCompletion(rB) // wait I/O
23     ExpandNeighbors(S, l, δ(·,·), q, rB)
24 end
25 return the closest k vertices in S
```

6 IMPLEMENTATION

We have implemented a prototype of RED-ANNS in C++ 17, utilizing the MPI framework for inter-node coordination and the RDMA Verbs programming interface for remote access. In our software and hardware stack, we use RoCE-supported NICs and Mellanox Ethernet switch to build the communication network. Through one-sided read operations, RED-ANNS utilizes RDMA's CPU offload capability to pull vertex data from remote memory into local memory for neighbor expansion. To efficiently handle Work Requests [27], we also make full use of RDMA's features, as detailed below.

(1) *Doorbell Batching:* We employ batch submission to improve the request efficiency during neighbor expansion. When posting multiple Work Requests to the `ibv_post_send()` verbs, submitting them as a linked list in a single call—rather than making multiple separate calls for each individual request—improves performance by allowing the low-level driver to optimize the process.
(2) *Selective Signaling:* We implement selective signaling in the Send Queue to reduce PCIe transactions and NIC processing overhead. By marking requests as unsignaled using `send_flags`, the system avoids generating a Completion Queue Entry (CQE) for every Work Queue Entry (WQE), and only the final request in a batched requests is marked as signaled to generate a CQE.
(3) *Concurrent Processing:* We enable the concurrent processing of multiple requests by configuring the parameters `max_rd_atomic` and `max_dest_rd_atomic` of the Queue Pair, allowing RDMA reads to be handled more efficiently.

Table 1: Datasets in experiments

Dataset	Dist.	Dim.	Scale	Query Emb.	Base Emb.	Type
DEEP [5]	L2	96	100M	Image	Image	ID
MS-Turing [53]	L2	100	100M	Text	Text	ID
Text2Image [42]	IP	200	100M	Text	Image	OOD
LAION [38]	L2	512	100M	Text	Image	OOD

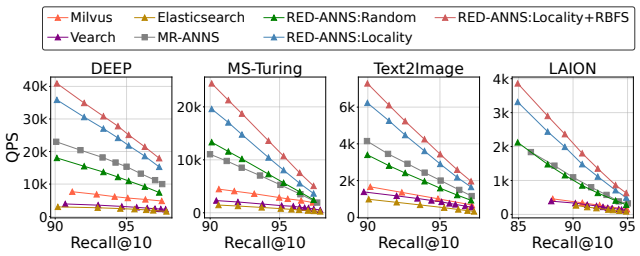


Figure 10: Performance comparison with different recalls

7 EVALUATION

7.1 Evaluation Setup

Hardware configuration. All evaluations were conducted on a rack-scale cluster consisting of four machines. Two machines are equipped with dual 10-core Intel(R) Xeon(R) Silver 4210R CPUs and 128 GB DRAM, and the other two with dual 16-core Intel(R) Xeon(R) Gold 5218 CPUs and 128 GB DRAM. All machines run Ubuntu 20.04 with Mellanox OFED v5.8-4.1.5.0 and are equipped with a ConnectX-5 MCX516A 100Gbps RoCE NIC (PCIe 3.0 x16), connected to a Mellanox SN2100 100GbE switch.

Workloads. We selected multiple representative datasets covering both In-Distribution (ID) and Out-of-Distribution (OOD) workloads. Table 1 summarizes the datasets, including DEEP, MS-Turing (MS-T), Text2Image (T2I) and LAION, which are widely recognized in academia and industry as benchmark datasets for ANNS. Among them, DEEP, MS-T and T2I were obtained from the BigANN Benchmark [42]. Each dataset consists of a collection of database vectors, query vectors, and the corresponding groundtruth. We standardize the experiments to be conducted on a data scale of 100M.

Comparing targets. We selected the popular distributed vector databases to compare RED-ANNS with mainstream industry implementations, including Milvus [47], Vearch [31], and Elasticsearch [1]. Moreover, to fairly and unbiased demonstrate the superiority of full-GPS over MapReduce-style execution, we also implemented a lightweight baseline, referred to as ***MR-ANNS***, which shares the same codebase with RED-ANNS (e.g., the AVX512 accelerated computation library) but follows MapReduce-style execution. MR-ANNS maintains only one subgraph per node to represent the best achievable performance for MapReduce-style methods.

We also evaluated several variants of RED-ANNS to assess the improvement of each proposed design component. Specifically, ***Random*** represents RED-ANNS using the full-GPS strategy with random data placement and query scheduling. ***Locality*** incorporates locality-aware data placement and affinity-based query scheduling. ***RBFS*** executes queries using the dependency-relaxed best-first search instead of the conventional BFS approach.

Parameter setting. In Milvus, Vearch and Elasticsearch, we use their HNSW index [33], with parameters $M = 64, efConstruction = 100$, and use the optimized segment parameter $maxSize = 20$GB to reduce the creation of numerous small segments in Milvus. For both MR-ANNS and RED-ANNS, we use the Vamana index [24] with parameters $R = 64, L = 100$. In RED-ANNS, the randomly sampled query training set is set to 1M (1% of the baseset size), the cache capacity for duplication is set to 4M (4% of the baseset size), the affinity-based query scheduling module uses a 30K base sampling set as the anchor set, and the relaxed hop count n is set to three.

7.2 Overall Performance

We evaluate performance in terms of retrieval speed, measured by Queries Per Second (QPS), and retrieval accuracy, measured by Recall@k, in the four-node cluster with eight threads per node.

Performance under different values of recall. Figure 10 shows performance under different recall levels, with k fixed at 10, following previous work [24, 33, 42]. To achieve higher recall, the (R)BFS algorithm requires a larger candidate pool size to allow more neighbor expansions, increasing the likelihood of retrieving the true top-k nearest neighbors. However, this results in more distance calculations, reducing QPS. Compared to MR-ANNS, RED-ANNS achieves performance improvements of up to 1.7-1.8×, 2.1-2.4×, 1.9-2.0× and 2.0-2.2× on the DEEP, MS-T, T2I and LAION datasets, respectively, at Recall@10≥0.9. Among the three industry implementations, Milvus delivers the best performance. RED-ANNS outperforms Milvus by 4.0–5.3×, and surpasses Vearch and Elasticsearch by 5.1–9.6× across the four datasets, at Recall@10≥0.9.

Performance under different values of k. Figure 11 shows performance at varying values of k, with the recall fixed at 0.9. As k increases, the (R)BFS algorithm usually also needs to traverse more vertices to find more nearest neighbors, resulting in a decrease in QPS. However, on the LAION dataset that exhibits significant OOD characteristics, retrieving an accurate top-1 result is particularly challenging, resulting in a lower QPS at k=1 compared to larger values of k. In general, RED-ANNS achieves up to 1.6-1.9×, 2.0-2.4×, 1.8-2.0× and 1.9-2.1× performance improvement over MR-ANNS on the DEEP, MS-T, T2I and LAION datasets, respectively, across different values of k. Notably, on the DEEP dataset, we observe that for a MapReduce-style execution, searching for only the top-40 results on each subgraph is sufficient to ensure that the final aggregated top-100 results achieve a Recall@k of 0.9, eliminating the need to search for the full top-100 results on each subgraph. Because Milvus enforces retrieving at least 100 results from each subgraph, the resulting recall naturally exceeds the 0.9 threshold.

Analysis of improvement. As shown in Figure 10 and Figure 11, Compared to MR-ANNS, *Random* achieves at most a 20% performance improvement at Recall@10=0.9 by employing the full-GPS strategy, which provides minimal computational overhead. However, its performance remains limited due to excessive remote accesses. Compared to *Random*, *Locality* improves more performance by 42–75% due to enhanced data access locality. Furthermore, compared to *Locality*, *RBFS* helps hide the network latency for necessary remote access, leading to additional performance gains of 14–33%. In addition, we note that MR-ANNS outperforms Milvus

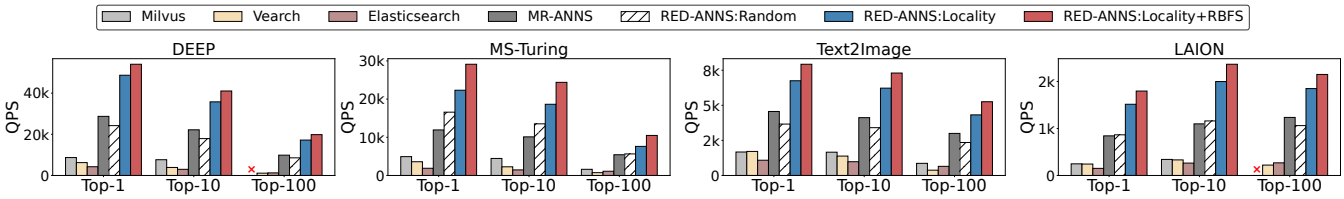


Figure 11: Performance comparison with Top-1, Top-10, and Top-100

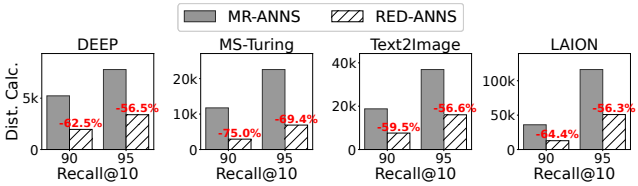


Figure 12: Distance computation overhead

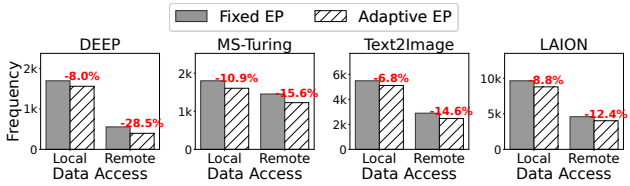


Figure 13: Performance implications of adaptive entry point

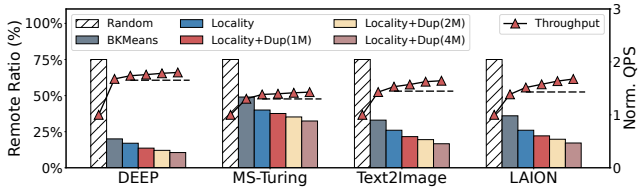


Figure 14: Proportion of remote accesses with different data placement schemes

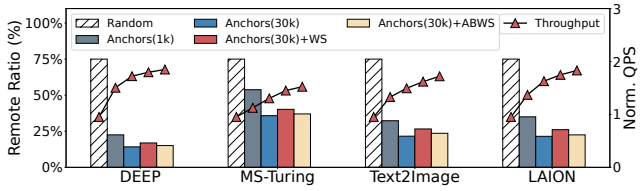


Figure 15: Proportion of remote accesses and corresponding throughput with different query scheduling schemes

because Milvus maintains multiple segments per node, which reduces indexing efficiency, and as a full-featured distributed vector database, Milvus incorporates a complete database architecture [19], inevitably introducing runtime overhead.

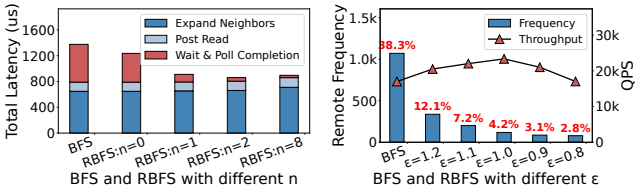
7.3 Effectiveness Analysis

7.3.1 Minimal Computational Overhead. Thanks to our full-GPS strategy, RED-ANNS achieves optimal indexing efficiency with minimal computational overhead. Figure 12 compares the distance computation overhead of RED-ANNS and MR-ANNS at Recall@10=0.9 and 0.95 in the four-node setting. At Recall@10=0.9, RED-ANNS reduces the number of distance computations by 62%, 75%, 59% and 64% on the DEEP, MS-T, T2I and LAION datasets, respectively. This significant reduction creates substantial opportunities for performance improvement compared to MapReduce-style approaches.

7.3.2 Enhanced Data Access Locality. We evaluate the impact of our data placement and query scheduling strategies on remote access separately by fixing one component while varying the other.

Locality-aware data placement. Figure 14 shows the proportion of remote accesses and throughput of RED-ANNS under different data placement schemes, including varying duplication scales. Under *Random*, the remote access ratio is typically 75% in a four-node setup. By applying locality-aware graph partitioning, the ratios are significantly reduced to 16%, 38%, 25%, and 24% on the DEEP, MS-T, T2I, and LAION datasets, respectively. After further duplicating 4M of frequently accessed remote vertices in the local cache, we leveraged the generalizability of offline profiling to achieve hit rates of 38%, 19%, 36%, and 33% for all remote access requests on the four datasets, respectively, thus reducing remote access ratios to 10%, 31%, 16% and 16%. However, the effectiveness of duplication is jointly influenced by cache lookup overhead, cache hit rate, and remote access cost. On the 100-dimensional MS-T dataset, where the hit rate is relatively low and the data transmission cost is small due to the lower dimensionality, duplication does not yield noticeable throughput gains. In contrast, on the 512-dimensional LAION dataset, where both the hit rate and the dimensionality are higher, duplication delivers an additional 11% performance improvement.

Affinity-based query scheduling. Figure 15 shows the proportion of remote accesses and throughput of RED-ANNS under different query scheduling schemes. Affinity-based query assignment with 1K anchors effectively routes queries to their most affine nodes, significantly reducing remote accesses compared to *Random*. Increasing anchor granularity to 30K enables a more accurate affinity estimation and further reduces remote accesses. Work-stealing (WS) increases the remote access proportion by 4–7% across the entire workload due to 10–15% of stolen queries losing locality. Nevertheless, it improves resource utilization through load balancing and achieves 6–12% higher QPS compared to assignment without WS. Affinity-based work-stealing (ABWS) recovers some of the



(a) Neighbor prefetching **(b) Neighbor pruning**

Figure 16: Performance implications of Relaxed BFS

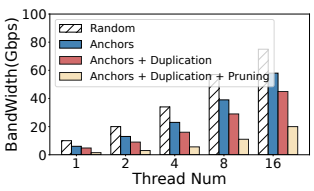


Figure 17: Network traffic

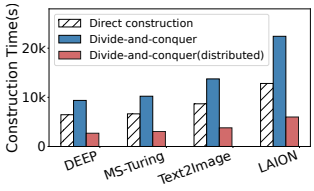


Figure 18: Construction time

lost locality, keeping the remote access proportion close to assignment without WS while boosting QPS by 11–18% through load balancing. The anchor set also assists in adaptively selecting local entry points during the search, as shown in Figure 13. By avoiding graph traversal from remote entry points, this mechanism reduces remote accesses by 12–28%. Meanwhile, since adaptive entry points shorten search paths, local access also decreases by 7–11%.

7.3.3 Hidden Remote Access Latency. RBFS hides remote access latency by relaxing the strict best-first greediness of conventional BFS, enabling network latency to overlap with CPU distance computations. Figure 16(a) decompose the average query latency of RBFS under different dependency relaxation levels on the MS-T dataset at Recall@10=0.9. The latency is broken down into four parts: *Expand Neighbors*, *Post Read*, *Wait I/O* and *Poll Completion*. Among them, *Expand Neighbors* part encompasses CPU overhead, such as *Graph Traversal*, *Distance Calculation* and *Candidate Update*. For RBFS, when $n = 0$ (i.e., no dependency relaxation), reordering the distance computations for neighbor expansion reduces CPU wait time, improving performance by 10%. At $n = 1$, dependency relaxation reduces CPU wait time by enabling greater overlap, leading to a noticeable drop in average query latency compared to conventional BFS. The latency reaches its lowest point at $n = 2$, where the network latency is effectively hidden, and the CPU wait time is reduced by 90%, leading to an overall 37% reduction in average query latency. However, as relaxed-hops increases further, the disruption of execution dependencies causes more vertex traversals, and the *Expand Neighbors* time exceeds that of BFS by 9% at $n = 8$.

7.3.4 Pruned Remote Access Frequency. RBFS reduces unnecessary data transfer by pruning remote neighbors based on a local quantized distance estimation, boosting throughput under constrained network resources. Figure 16(b) shows the remote access frequency and throughput of RBFS under different pruning parameters ϵ on the MS-T dataset at Recall@10=0.9. The remote access proportion among all data accesses is also indicated above each bar. A smaller ϵ corresponds to a more aggressive pruning strategy, which reduces remote accesses, but may negatively impact search accuracy due to filtering necessary neighbors. At $\epsilon = 1.2$, pruning reduces remote accesses by 68% and improves QPS compared to conventional BFS. The optimal QPS is achieved at $\epsilon = 1.0$, where the remote access frequency drops by 89%. However, when ϵ falls below 1.0, pruning becomes too aggressive, filtering essential remote neighbors and degrading search accuracy. Maintaining the target recall under such conditions requires a larger candidate pool size, which lengthens the search path and consequently lowers QPS.

7.3.5 Resource Tradeoff. In RED-ANNS, memory is strategically utilized to reduce remote accesses during search, thus reducing network overhead. Specifically, the anchor set used for affinity estimation contains 30K vectors, accounting for only 0.03% of the 100M-scale dataset and resulting in negligible memory overhead. Furthermore, each node duplicates up to 4% hot vertices in its local cache, adding 16% memory overhead across four nodes (i.e., $4\% \times 4$). We also employ product quantization with a 6.25% compression ratio to prune remote neighbors, storing the codebook of remote vectors for each node, which introduces an additional 18.75% memory overhead (i.e., $6.25\% \times 3$). Figure 17 presents the network consumption under different optimization techniques, with varying numbers of search threads per node on the MS-T dataset. With 8 threads per node, after introducing an anchor set for affinity-based query scheduling, network traffic drops to 68% of *Random*. Adding duplication to increase the local availability of remote vertices further reduces traffic to 49%, and incorporating neighbor pruning reduces it to 17%, significantly alleviating bandwidth consumption.

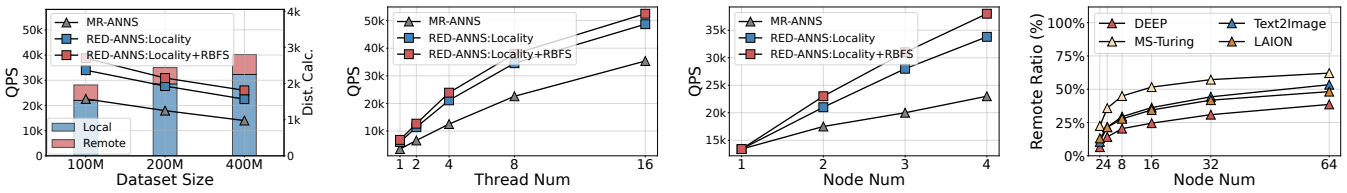
7.4 Scalability

We evaluated the scalability of RED-ANNS and compared it with MR-ANNS in the same memory budget, that is, without additional duplication or codebook storage, with k fixed at 10.

Bigger dataset. Figure 19(a) shows the performance and distance computation overhead of RED-ANNS across different dataset sizes on DEEP dataset at Recall@10=0.9 in the four-node setting. As the dataset size increases, throughput gradually decreases due to higher computation costs; however, the proportion of remote accesses remains stable at 18–21%. Compared to MR-ANNS, RED-ANNS achieves performance improvements of 1.7×, 1.6×, and 1.6× on the 100M, 200M, and 400M scales, respectively.

Scale up. Figure 19(b) shows the performance under varying numbers of search threads on DEEP dataset at Recall@10=0.9. Compared to MR-ANNS, RED-ANNS achieves speedups of 1.9×, 1.7×, and 1.5× with 1, 8, and 16 threads per node, respectively, with greater gains observed at fewer threads. This is because contention for network resources affects RED-ANNS performance as concurrency increases; however, its optimized data locality still significantly reduces remote accesses, delivering strong overall speedups.

Scale out. Figure 19(c) shows the performance under different numbers of nodes on DEEP dataset at Recall@10=0.9. Compared to MR-ANNS, RED-ANNS achieves up to 1.3×, 1.5×, and 1.7× improvements with 2, 3, and 4 nodes, respectively, providing strong scalability because its computational overhead remains constant owing to the full graph index. In contrast, MR-ANNS incurs increasing computational costs due to the growing number of subgraphs



(a) Bigger dataset **(b) Scale up** **(c) Scale out** **(d) Remote access in scale out**

Figure 19: Scalability evaluation

with more nodes. We also conduct pseudo-distributed experiments to study remote access behavior at larger node scales by launching more RED-ANNS instances, where any cross-instance data access is treated as a remote access, logically corresponding to RDMA communication in real multi-node deployments. Figure 19(d) shows that the remote access ratio, on the four datasets at Recall@10=0.9, increases with the number of nodes; however, even with 32 nodes, more than half of the data accesses remain local, indicating the locality on larger node scales.

8 DISCUSSION

Graph construction. To construct the full graph index, we use DiskANN [24] for scalable graph construction on a single node, supporting two construction strategies: direct construction and divide-and-conquer. For datasets that fit entirely in memory, a Vamana index can be built directly on the dataset. For larger datasets, the divide-and-conquer approach first divides the dataset into k clusters, introducing overlaps between clusters by assigning each vector to its top-ℓ nearest clusters (typically ℓ=2). Vamana indices are then constructed independently and sequentially for each cluster. Finally, the full graph index is obtained by merging the edge sets of all clusters, where vectors shared across multiple clusters act as bridges to establish inter-cluster connectivity and yield a connected full graph. In distributed settings, the construction tasks per-cluster can also be naturally parallelized across multiple nodes [29, 40], improving efficiency. Figure 18 shows the construction time on our 100M-scale datasets using different construction approaches. On a single node with 16 threads, the divide-and-conquer strategy incurs a longer construction time because cluster overlaps increase the total data volume processed. However, when per-cluster construction is parallelized across four nodes, the construction time is significantly reduced, with the average dropping below two hours.

Graph updates. We construct a full graph index over the entire database vectors in distributed memory. This indexing paradigm is consistent with the existing single-node index setting [24, 43]. For insertions, the graph index first needs to locate the insertion position in the graph by searching for the nearest neighbors of the new vector and then adds edges between them. For deletions, the old vector is removed, and its former neighbors are reconnected to restore graph connectivity. In the distributed setting, to enable efficient search during insertion, we treat the inserted vector as a query vector and schedule it to an appropriate node. To correctly handle edge modifications during insertion and deletion, we adopt distributed lock-based concurrency control to protect access to each vertex's neighbor set, supporting concurrent updates across multiple nodes. As the underlying data evolve through insertions and deletions, the distribution of base vectors may change. In such cases, lightweight reprocessing becomes necessary. For example, the anchor set used for query scheduling should be resampled from the updated base set to reflect the new distribution.

9 RELATED WORK

Large-scale ANNS solutions. To address the memory challenge in large-scale ANNS, existing methods mainly fall into three categories. 1) *Compression-based methods* leverage quantization techniques to compress raw vectors, such as PQ [25], SQ [57], and RaBitQ [17], allowing the datasets to fit in memory, where distance estimation is conducted through compressed codes. 2) *Persistent storage–based methods* offload large datasets to SSDs/NVM to alleviate DRAM pressure. For example, DiskANN [24] stores the graph on SSDs and dynamically loads data during search. HM-ANN [37], as a heterogeneous memory design, stores the graph in Intel Optane (high capacity) while maintaining high-degree hub vertices in DRAM (low latency) to accelerate access. 3) *Distributed memory–based methods* adopt divide-and-conquer strategies to scale storage across multiple nodes. However, existing logical segmentation implementations, such as Milvus [47], Vearch [31], and Elasticsearch [1], suffer from degraded query performance due to escalating computational overhead as the number of segments increases. In contrast, we propose RED-ANNS, which maintains a globally shared full-graph index in the unified memory space of multiple nodes, providing optimal computation efficiency and strong scalability.

10 CONCLUSION

Distributed ANNS is a fundamental infrastructure for retrieving large volumes of unstructured data. In this paper, we identify resource inefficiencies in existing solutions and propose RED-ANNS, a framework that leverages RDMA networks to build a shared memory of a cluster and adopts a full graph index to support efficient distributed ANNS. To reduce remote accesses during search, we propose locality-aware data placement and affinity-based query scheduling, enhancing data access locality. To hide network latency for remote accesses, we propose a dependency-relaxed best-first search algorithm which incorporates an effective data prefetching. Extensive experimental results demonstrate that RED-ANNS delivers outstanding performance and scalability on various datasets.

ACKNOWLEDGMENTS

This work was supported by the Project of Key R&D Program of Shandong Province (Grant No. 2024CXGC010113).

REFERENCES

[1] 2020. Elasticsearch: An open source, distributed, RESTful search engine. https://github.com/elastic/elasticsearch

[2] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. 2023. Retrieval-based Language Models and Applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, ACL 2023, Toronto, Canada, July 9-14, 2023*, Yun-Nung Vivian Chen, Margot Mieskes, and Siva Reddy (Eds.). Association for Computational Linguistics, 41–46. https://doi.org/10.18653/V1/2023.ACL-TUTORIALS.6

[3] Artem Babenko and Victor S. Lempitsky. 2014. Additive Quantization for Extreme Vector Compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 931–938. https://doi.org/10.1109/CVPR.2014.124

[4] Artem Babenko and Victor S. Lempitsky. 2015. The Inverted Multi-Index. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 6 (2015), 1247–1260. https://doi.org/10.1109/TPAMI.2014.2361319

[5] Artem Babenko and Victor S. Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2055–2063. https://doi.org/10.1109/CVPR.2016.226

[6] Qi Chen. 2021. Research talk: SPTAG++: Fast hundreds of billions-scale vector search with millisecond response time. https://www.microsoft.com/en-us/research/video/research-talk-sptag-fast-hundreds-of-billions-scale-vector-search-with-millisecond-response-time/.

[7] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search*. https://github.com/Microsoft/SPTAG

[8] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighborhood Search. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 5199–5212. https://proceedings.neurips.cc/paper/2021/hash/299dc35e747eb77177d9cea10a802da2-Abstract.html

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells (Eds.). ACM, 191–198. https://doi.org/10.1145/2959100.2959190

[10] Rieke de Maeyer, Sami Sieranoja, and Pasi Fränti. 2023. Balanced k-means revisited. (2023).

[11] Shiyuan Deng, Xiao Yan, Kelvin Kai Wing Ng, Chenyu Jiang, and James Cheng. 2019. Pyramid: A General Framework for Distributed Similarity Search on Large-scale Datasets. In *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye (Eds.). IEEE, 1066–1071. https://doi.org/10.1109/BIGDATA47090.2019.9006219

[12] Etienne Dilocker, Bob van Luijt, Byron Voorbach, Mohd Shukri Hasan, Abdel Rodriguez, Dirk Alexander Kulawiak, Marcin Antas, and Parker Duckworth. [n.d.]. *Weaviate*. https://github.com/weaviate/weaviate

[13] Mohamad Dolatshah, Ali Hadian, and Behrouz Minaei-Bidgoli. 2015. Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces. *CoRR* abs/1511.00628 (2015). arXiv:1511.00628 http://arxiv.org/abs/1511.00628

[14] Ishita Doshi, Dhritiman Das, Ashish Bhutani, Rajeev Kumar, Rushi Bhatt, and Niranjan Balasubramanian. 2021. LANNS: A Web-Scale Approximate Nearest Neighbor Lookup System. *Proc. VLDB Endow.* 15, 4 (2021), 850–858. https://doi.org/10.14778/3503585.3503594

[15] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024). https://doi.org/10.48550/ARXIV.2401.08281 arXiv:2401.08281

[16] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474. https://doi.org/10.14778/3303753.3303754

[17] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3 (2024), 167. https://doi.org/10.1145/3654970

[18] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*. IEEE Computer Society, 2946–2953. https://doi.org/10.1109/CVPR.2013.379

[19] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. *Proc. VLDB Endow.* 15, 12 (2022), 3548–3561. https://doi.org/10.14778/3554821.3554843

[20] 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 *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 3887–3896. http://proceedings.mlr.press/v119/guo20h.html

[21] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, Jeffrey Scott Vitter (Ed.). ACM, 604–613. https://doi.org/10.1145/276698.276876

[22] 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. *CoRR* abs/2211.12850 (2022). https://doi.org/10.48550/ARXIV.2211.12850 arXiv:2211.12850

[23] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS: Software-Hardware Collaborative Memory Disaggregation and Computation for Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, Julia Lawall and Dan Williams (Eds.). USENIX Association, 585–600. https://www.usenix.org/conference/atc23/presentation/jang

[24] 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. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[25] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[26] Yannis Kalantidis and Yannis Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2329–2336. https://doi.org/10.1109/CVPR.2014.298

[27] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*, Ajay Gulati and Hakim Weatherspoon (Eds.). USENIX Association, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[28] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392. https://doi.org/10.1137/S1064827595287997

[29] Sang-Hong Kim and Ha-Myung Park. 2023. Efficient Distributed Approximate k-Nearest Neighbor Graph Construction by Multiway Random Division Forest. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye (Eds.). ACM, 1097–1106. https://doi.org/10.1145/3580305.3599327

[30] Patrick S. H. Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. PAQ: 65 Million Probably-Asked Questions and What You Can Do With Them. *Trans. Assoc. Comput. Linguistics* 9 (2021), 1098–1115. https://doi.org/10.1162/TACL_A_00415

[31] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyun Ni, and Ning Wang. 2019. The Design and Implementation of a Real Time Visual Search System on JD E-commerce Platform. arXiv:1908.07389 [cs.IR]

[32] Haotian Liu, Kilho Son, Jianwei Yang, Ce Liu, Jianfeng Gao, Yong Jae Lee, and Chunyuan Li. 2023. Learning Customized Visual Models with Retrieval-Augmented Knowledge. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 15148–15158. https://doi.org/10.1109/CVPR52729.2023.01454

[33] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[34] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2311–2320. https://doi.org/10.1145/3394486.3403280

[35] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proc. ACM Manag. Data* 1, 1 (2023), 54:1–54:27. https://doi.org/10.1145/3588908

[36] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark,

Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, 8748–8763. http://proceedings.mlr.press/v139/radford21a.html

[37] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/788d986905533aba051261497ecffcbb-Abstract.html

[38] 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. *CoRR* abs/2111.02114 (2021). arXiv:2111.02114 https://arxiv.org/abs/2111.02114

[39] Min Joon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P. Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-Time Open-Domain Question Answering with Dense-Sparse Phrase Index. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 4430–4441. https://doi.org/10.18653/V1/P19-1436

[40] Yang Shi, Yiping Sun, Jiaolong Du, Xiaocheng Zhong, Zhiyong Wang, and Yao Hu. 2025. Scalable Overload-Aware Graph-Based Index Construction for 10-Billion-Scale Vector Similarity Search. In *Companion Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025 - 2 May 2025*, Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov (Eds.). ACM, 1303–1307. https://doi.org/10.1145/3701716.3715576

[41] Chanop Silpa-Anan and Richard I. Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society. https://doi.org/10.1109/CVPR.2008.4587638

[42] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2021. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *NeurIPS 2021 Competitions and Demonstrations Track, 6-14 December 2021, Online (Proceedings of Machine Learning Research)*, Douwe Kiela, Marco Ciccone, and Barbara Caputo (Eds.), Vol. 176. PMLR, 177–189. https://proceedings.mlr.press/v176/simhadri22a.html

[43] 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. *CoRR* abs/2105.09613 (2021). arXiv:2105.09613 https://arxiv.org/abs/2105.09613

[44] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*. IEEE Computer Society, 1470–1477. https://doi.org/10.1109/ICCV.2003.1238663

[45] Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. 2024. Scalable Billion-point Approximate Nearest Neighbor Search Using SmartSSDs. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*, Saurabh Bagchi and Yiying Zhang (Eds.). USENIX Association, 1135–1150. https://www.usenix.org/conference/atc24/presentation/tian

[46] Boxin Wang, Wei Ping, Lawrence McAfee, Peng Xu, Bo Li, Mohammad Shoeybi, and Bryan Catanzaro. 2024. InstructRetro: Instruction Tuning post Retrieval-Augmented Pretraining. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=PLAGGbssT8

[47] Jianguo 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 *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2614–2627. https://doi.org/10.1145/3448016.3457550

[48] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (2021), 1964–1978. https://doi.org/10.14778/3476249.3476255

[49] 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. *Proc. VLDB Endow.* 13, 12 (2020), 3152–3165. https://doi.org/10.14778/3415478.3415541

[50] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix X. Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5745–5755. https://proceedings.neurips.cc/paper/2017/hash/b6617980ce90f637e68c3ebe8b9be745-Abstract.html

[51] Peter N. Yianilos. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, Vijaya Ramachandran (Ed.). ACM/SIAM, 311–321. http://dl.acm.org/citation.cfm?id=313559.313789

[52] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, and Yu Wang. 2023. DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-based Approximate Nearest Neighbor Search. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023, Toronto, ON, Canada, 28 October 2023 - 1 November 2023*. ACM, 283–296. https://doi.org/10.1145/3613424.3614292

[53] Hongfei Zhang, Xia Song, Chenyan Xiong, Corby Rosset, Paul N. Bennett, Nick Craswell, and Saurabh Tiwary. 2019. Generic Intent Representation in Web Search. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 65–74. https://doi.org/10.1145/3331184.3331198

[54] Jianjin Zhang, Zheng Liu, Weihao Han, Shitao Xiao, Ruicheng Zheng, Yingxia Shao, Hao Sun, Hanqing Zhu, Premkumar Srinivasan, Weiwei Deng, Qi Zhang, and Xing Xie. 2022. Uni-Retriever: Towards Learning the Unified Embedding Based Retriever in Bing Sponsored Search. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, Aidong Zhang and Huzefa Rangwala (Eds.). ACM, 4493–4501. https://doi.org/10.1145/3534678.3539212

[55] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. 2023. Fast, Approximate Vector Queries on Very Large Unstructured Datasets. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, Mahesh Balakrishnan and Manya Ghobadi (Eds.). USENIX Association, 995–1011. https://www.usenix.org/conference/nsdi23/presentation/zhang-zili

[56] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna Qiu, and Lili Qiu. 2024. Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely. *CoRR* abs/2409.14924 (2024). https://doi.org/10.48550/ARXIV.2409.14924 arXiv:2409.14924

[57] Xiaoyao Zhong, Haotian Li, Jiabao Jin, Mingyu Yang, Deming Chu, Xiangyu Wang, Zhitao Shen, Wei Jia, George Gu, Yi Xie, Xuemin Lin, Heng Tao Shen, Jingkuan Song, and Peng Cheng. 2025. VSAG: An Optimized Search Framework for Graph-based Approximate Nearest Neighbor Search. *CoRR* abs/2503.17911 (2025). https://doi.org/10.48550/ARXIV.2503.17911 arXiv:2503.17911