Efficient Vector Search on Disaggregated Memory with d-HNSW

Yi Liu*
University of California Santa Cruz
Santa Cruz, CA, USA

Fei Fang*
University of California Santa Cruz
Santa Cruz, CA, USA

Chen Qian
University of California Santa Cruz
Santa Cruz, CA, USA

ABSTRACT

Efficient vector query processing is essential for powering large-scale AI applications, such as LLMs. However, existing solutions struggle with growing vector datasets that exceed the memory capacity of a single machine, leading to excessive data movement and resource underutilization in monolithic architectures.

We introduce d-HNSW, the first vector search engine for RDMA-based disaggregated memory systems. d-HNSW achieves high performance by supporting efficient data indexing with minimal network communication overhead. At its core, d-HNSW introduces a novel disaggregation of the HNSW graph-based vector index, leveraging the properties of greedy search to coordinate data transfers efficiently between the memory and compute pools. Specifically, d-HNSW incorporates three key techniques: (i) Representative index caching, which constructs a lightweight index from a sampled subset of the data and caches it in the compute pool to minimize frequent access to critical components of the hierarchical graph index; (ii) RDMA-friendly data layout, which optimizes data placement to reduce networking round trips for both vector queries and insertions; and (iii) Batched query-aware data loading, which mitigates bandwidth usage between memory and compute pools, addressing the limited cache capacity of compute nodes. The experimental results demonstrate that d-HNSW outperforms Naive d-HNSW implementation by up to 117× in query latency while maintaining a recall of 0.87 on the SIFT1M dataset.

CCS CONCEPTS

• **Information systems → Database design and models**.

* Equal contribution.

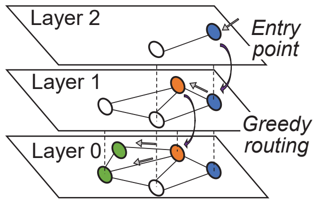


Figure 1: Graph-based vector search index: HNSW.

KEYWORDS

Disaggregated memory, vector database, RDMA, HNSW

ACM Reference Format:
Yi Liu, Fei Fang and Chen Qian. 2025. Efficient Vector Search on Disaggregated Memory with d-HNSW. In *17th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3736548.3737822

1 INTRODUCTION

Vector similarity search [24, 33, 34, 42, 45, 46] aims to identify the most similar vectors from a large dataset given a query vector. Approximate nearest neighbor (ANN) search has emerged as a practical alternative for exact top-k nearest neighbor search, offering significant speedups with minimal accuracy trade-offs. Vector databases leverage ANN search techniques to support efficient retrieval, making them essential for a wide range of ML applications [13, 14], including search engines [21] and recommendation systems [29]. In particular, they play a critical role in powering large language models (LLMs) and retrieval-augmented generation (RAG) systems [9] by enabling fast, high-dimensional similarity searches over massive embedding spaces. In RAG, a vector database retrieves semantically relevant documents based on the user prompt's embedding, allowing LLMs to generate responses with external knowledge rather than limited to the information encoded in their model parameters. As these AI-driven applications [7, 29] continue to grow, the demand for scalable and high-performance vector search solutions has become increasingly crucial.

Disaggregation [35] is gaining attention in cloud computing by separating storage and compute hardware resources, allowing them to scale independently for better flexibility and efficiency. Different resource pools are connected with high-speed networks to realize fast data transfer. For example, RDMA (Remote Direct Memory Access) [10] is one of the high-performance fabrics that enables direct memory access between remote machines, bypassing the CPU to reduce latency and improve throughput. Recent industry efforts, such as DeepSeek's 3FS [2, 4], have demonstrated the benefits of leveraging RDMA for AI training and inference, enabling high-speed remote memory access with low latency. Beyond LLM applications, modern vector datasets exhibit dynamic and heterogeneous resource requirements. For example, compute-intensive query workloads often fluctuate independently from storage demands as dimensions scale. Disaggregation can address this imbalance by enabling independent scaling of memory pools and compute instances. Inspired by this architectural shift [31, 35] and requirements, we are motivated to propose an RDMA-based disaggregated vector database designed to improve hardware resource utilization with high-throughput vector queries.

Among the various similarity search algorithms [6, 23, 33], graph-based approaches [5, 19] have demonstrated superior performance in both recall and latency. Hierarchical Navigable Small World (HNSW) [19] is a widely adopted graph-based index that balances vector search accuracy and efficiency. Thus, in this work, we introduce d-HNSW, a fast, RDMA-based vector similarity search engine designed for disaggregated memory system. d-HNSW aims to bridge the gap between high-performance vector similarity search and the emerging disaggregated architecture in datacenter, ensuring scalability and efficiency in handling high-throughput data queries.

The disaggregated memory pool provides abundant memory resources, allowing us to store both the HNSW index and all original floating-point vector data on it. Intuitively, the disaggregated compute instances handle data requests and reply on one-sided RDMA primitives to directly access the index and vectors, bypassing memory instances' CPUs. However, this approach presents several challenges. (i) The greedy algorithm [41] in HNSW navigates the index by comparing distances to the query vector along a search path, where each node on the graph represents a vector. The traversal path is unpredictable, and if we need to read vectors on each single step along the path via the network, the number of round trips required for a vector query becomes excessive. To mitigate this, we propose partitioning vectors into groups with an *representative index* and selectively reading only the partitions that most likely contain top-k candidates. (ii) All partitions are compactly serialized and written to remote registered memory. When a new vector is inserted, it needs to be stored in available memory while ensuring fast index access for different partitions. If we allocate a global memory space for inserted vectors, those belonging to the same partition will be scattered across fragmented memory regions, leading to high latency as the RDMA NIC needs to issue multiple network or PCIe round trips. To address this, we propose an *RDMA-friendly graph index layout* that enables efficient data queries while supporting dynamic vector insertions. (iii) Since we process vector queries in a batch [13] and there is limited cache DRAM space in the compute pool, we propose *query-aware data loading* to reduce partition data loading from the memory pool and save bandwidth by pruning duplicate partition transfers, thereby improving vector query throughput.

We implement a prototype of d-HNSW with 12K LoC and evaluate it against other RDMA-based approaches in terms of vector query recall, latency, and throughput across various datasets. The results show d-HNSW outperforms other baselines by up to 117× with top-10 benchmarking. **To our best knowledge, d-HNSW is the first vector database designed for RDMA-based disaggregated memory systems.**

2 BACKGROUND

2.1 Graph-based vector search with HNSW.

Vector similarity search is crucial for efficiently retrieving high-dimensional data in modern ML applications such as RAG [9] for LLMs. Traditional methods like KD-trees [23] and LSH [6] struggle with scalability and search accuracy in high-dimensional spaces, leading to the development of graph-based indexing techniques [5, 19]. These methods construct a navigable graph where data points serve as nodes, and edges encode proximity relationships, enabling fast traversal during queries. For example, as shown in Fig. 1, HNSW builds a multi-layered graph [19] where upper layers provide a coarse-grained overview for fast entry into the structure, and lower layers refine the search with more densely connected nodes. During a query, the search starts from an *entry point* and follows a greedy routing strategy, moving to the closest neighbor at each layer. This closest vector will become the entry point to the next layer, performing greedy routing again toward the queried vector while refining the candidate set. The number of vectors in each layer increases *exponentially*. By leveraging small-world properties and efficient greedy search heuristics, HNSW significantly improves both recall and query speed compared to earlier graph-based methods, making it one of the most effective ANN search algorithms in modern vector databases.

2.2 RDMA-based disaggregated memory.

RDMA technologies (e.g. RoCE [20], Infiniband [10]) enable reliable and in-order packet delivery, making them well-suited for indexing structures in disaggregated memory systems. It supports RDMA READ/WRITE for fetching and writing data directly on remote memory without CPU involvement, and atomic operations like Compare-And-Swap (CAS)

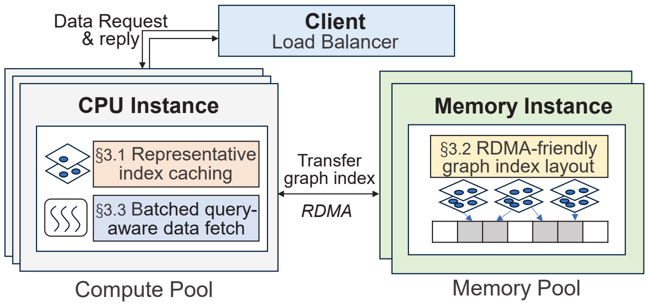


Figure 2: The overview of d-HNSW.

and Fetch-And-Add (FAA), enable efficient and lock-free data access. Designing an efficient indexing data structure tailored for RDMA-based remote memory applications can reduce system computation overheads, minimize network round trips, and realize data access with low latency.

3 d-HNSW DESIGN

We present d-HNSW, an RDMA-based vector similarity search engine on disaggregated memory. d-HNSW exploits the characteristics of RDMA-based memory data accessing and graph-based index HNSW to realize fast and bandwidth-efficient vector query processing. d-HNSW achieves so by representative index caching (§3.1), RDMA-friendly graph index storage in remote memory (§3.2), and query-aware batched data loading (§3.3). Here, we provide a brief overview of d-HNSW as Fig. 2 shows, d-HNSW requires tailored coordination between compute instances and memory instances on vector query serving. We assume the client load balancer distributes the workload across multiple CPU instances. The compute and memory pools are interconnected via RDMA, enabling efficient transfer of vector indices and data. **We target the disaggregated scenario where compute pool contains abundant CPU resources across many instances, each with limited DRAM serving as a cache, while memory instances have extremely weak computational power, handling lightweight memory registration tasks.**

While presented for vector search, d-HNSW's separation of routing metadata (meta-HNSW) from partitioned graph data (sub-HNSWs) provides a template for disaggregating other graph workloads. The architecture naturally extends to other graph computing tasks, including GNN computation and random walk optimizations.

3.1 Representative index caching.

Graph-based vector search schemes [5, 19] rely on greedy routing to iteratively navigate toward the queried vector. However, the search path can span the entire graph, potentially covering distant vectors. For example, HNSW exhibits small-world properties, allowing long-range connections between vectors that are far apart. However, loading the entire graph index from the memory pool to the computer pool for each query is impractical, because the compute pool has limited storage resources in a disaggregated system. This approach would not only consume excessive bandwidth by transferring a significant portion of untraversed vectors but also introduce additional latency, thereby degrading the overall search efficiency.

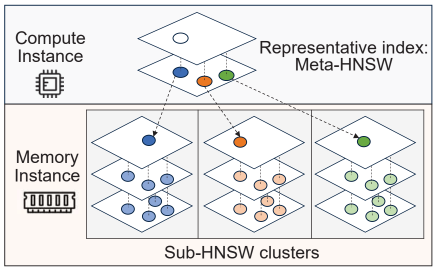


Figure 3: Representative index caching in d-HNSW.

We propose partitioning the vector database into multiple subsets, as shown in Fig. 3. Inspired by Pyramid [3], we construct a three-layer representative HNSW, referred to as *meta-HNSW*, by uniformly selecting 500 vectors. This meta-HNSW serves as a lightweight index and a cluster classifier for the entire dataset, and it only costs 0.373 MB for SIFT1M and 1.960 MB for GIST1M datasets from our experiments. The search process starts from a fixed entry point in the top layer L_2 of meta-HNSW and applies greedy routing at each layer, traversing downward until reaching a vector in its bottom layer L_0. Each vector in L_0 defines a partition and serves as an entry point to a corresponding *sub-HNSW*. All vectors assigned to the same partition will be used to construct their respective sub-HNSW. The overall graph index consists of two components: meta-HNSW, which provides coarse-grained classification, and sub-HNSWs, which enable fine-grained search within partitions. **To improve search efficiency in disaggregation, we cache the lightweight meta-HNSW in the compute pool, allowing it to identify the most relevant sub-HNSW clusters for a given query.** Meanwhile, we put all sub-HNSW clusters in the memory pool. For each vector query, only a small subset of sub-HNSW clusters needs to be loaded from the memory pool via the network, reducing both bandwidth usage and search latency.

3.2 RDMA-friendly graph index storage layout in remote memory.

RDMA enables efficient data access to targeted remote memory addresses. To efficiently read and write sub-HNSW cluster data in remote memory, an intuitive approach is to serialize all sub-HNSW clusters in the registered memory. Given that the top-m closest sub-HNSW clusters for a queried vector q are $\{S_0, .., S_{m-1}\}$, the compute instance can issue

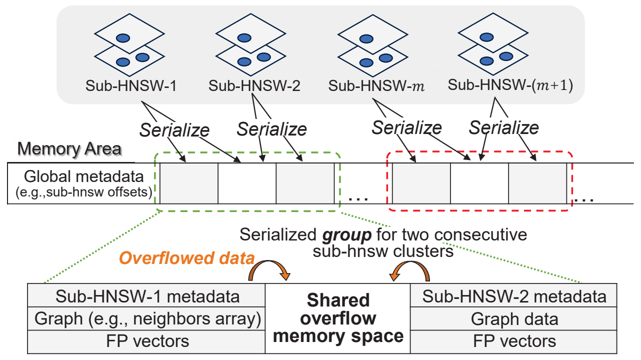


Figure 4: RDMA-friendly sub-HNSW indexing data layout in remote memory.

RDMA_READ commands to access these serialized clusters and then deserialize them. However, two challenges arise: (1) If the queried clusters $\{S_0, .., S_{m-1}\}$ are not stored contiguously in memory, multiple RDMA round trips are required, increasing latency. (2) When new vectors are inserted, the size of each sub-HNSW cluster may exceed the allocated space. Since shifting all stacked sub-HNSW clusters is impractical, newly inserted vectors and their metadata may be placed in non-contiguous memory regions if they are simply appended at the tail of the available area. This fragmentation increases access latency and reduces query throughput due to the higher cost of scattered index access.

As shown in Fig. 4, we allocate and register a continuous memory space in memory instance to store both the serialized HNSW index and floating-point vectors. At the beginning of this memory space, a global metadata block records the offsets of each sub-HNSW cluster, as their sizes vary. The remaining memory space is divided into *groups*, each of which is capable of holding two sub-HNSW clusters. Within each group, the first section stores the first serialized sub-HNSW cluster, which includes its metadata, neighbor array for HNSW, and the associated floating-point vectors. The second sub-HNSW cluster is placed at the end of the group. Between these two clusters, we allocate a 0.75 MB for SIFT1M 3.92 MB for GIST1M shared overflow memory space to accommodate newly inserted vectors for both sub-HNSW clusters. When a vector query requires loading a sub-HNSW cluster, the compute instance issues an RDMA_READ command to retrieve the cluster along with its corresponding shared overflow memory space. This layout ensures that newly inserted vectors are stored continuously with the original sub-HNSW data, enabling them to be read back with a one-time RDMA_READ command. To optimize memory usage, each pair of adjacent sub-HNSW clusters shares a single overflow memory space for accommodating newly inserted vectors rather than allocating a separate one for each cluster.

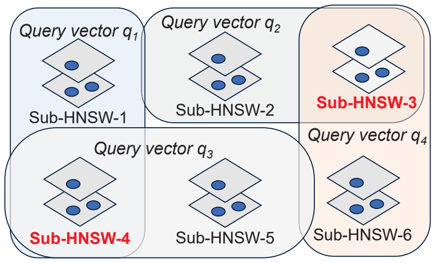


Figure 5: Query-aware sub-HNSW clusters loading.

If multiple sub-HNSW clusters need to be loaded into the compute pool for batched query processing, and they are not stored continuously in memory, we leverage *doorbell batching* to read them in a single network round-trip with RDMA NIC issuing multiple PCIe transactions. However, there is a tradeoff in the number of batched operations within a single RDMA command. If too many operations are included in one round-trip, it can interfere with other RDMA commands and incur long latency due to the scalability of the RDMA NIC. The memory offsets of each sub-HNSW cluster are cached in all compute instances after the sub-HNSW clusters are written to the memory pool, with the latest version stored at the beginning of the memory space in the memory instance.

Our shared overflow mechanism diverges from prior graph systems by employing an append-based, RDMA-aware design. Newly inserted vectors, along with their neighbor arrays and other metadata, are appended sequentially to the overflow memory region. This approach ensures efficient RDMA operations through contiguous memory writes. When any shared overflow region becomes full, the system triggers a background process to rebuild the entire d-HNSW indexing structure. This design avoids memory fragmentation and adapts to dynamic workloads, while maintaining low insertion overhead and preserving the efficiency of RDMA-based access.

3.3 Query-aware batched data loading.

To reduce bandwidth usage for transferring graph index and improve query efficiency, we propose merging sub-HNSW index loading for queried vectors in the same batch.

Given a batch of queried vectors $\{q_1, q_2, ..., q_s\}$ and a total of m sub-HNSW clusters, each queried vector requires searching the top-k closest vectors from the b closest sub-HNSWs. However, the DRAM resources in the compute instance can only accommodate and cache c sub-HNSWs. To optimize loading, we analyze the required $b * s$ sub-HNSWs *online* and ensure that each sub-HNSW is loaded from the memory pool only **once**.

For example, as shown in Fig. 5, queried vector q_1's two closest sub-HNSW clusters are S_1 and S_4, while q_3's two closest sub-HNSWs are S_4 and S_5. Similarly, S_3 is required for both q_2 and q_4. Given a doorbell batch size of 2 for accessing sub-HNSWs, the compute instance can issue an RDMA_-READ command to fetch index S_3 and S_4 in one network round-trip, then compute the top-k closest vectors candidates for all queries $\{q_1,q_2,q_3,q_4\}$ first. The results will be temporarily stored for further computation and comparison because each query vector still requires another sub-HNSW to obtain the final answer. Note that S_3 and S_4 will not be loaded again within the same batch.

Once all required sub-HNSW clusters for the batched queried vectors have been loaded and traversed, the query results will be returned. Additionally, we retain the most recently loaded c sub-HNSWs for the next batch. If the required sub-HNSWs are already in the compute instance, they do not need to be loaded again, further reducing data transfer overhead.

4 EVALUATION

We develope and evaluate the prototype of d-HNSW on CloudLab [25] using real-world hardware. Our testbed consists of four Dell PowerEdge R650 servers, each equipped with two 36-core Intel Xeon Platinum CPUs, 256GB RAM, a 1.6TB NVMe SSD, and a Mellanox ConnectX-6 100Gb NIC. Three servers act as a compute pool, while one serves as the memory instance. We compare d-HNSW against the following baselines: (1) Naive-HNSW: when a vector query arrives at a compute node, Naive-HNSW issues an RDMA read command to fetch corresponding sub-HNSW clusters, bypassing the memory node's CPU. (2) d-HNSW (w./o. doorbell): with meta-HNSW caching and query-aware data loading, the compute node reads sub-HNSW clusters in multiple round-trips, while our d-HNSW reads discontinuous sub-HNSW clusters in a single doorbell batch.

Each server has 144 hyperthreads, which are divided into 8 compute instances. Each instance runs a vector query worker that sends RDMA commands for top-k vector retrieval. Each instance uses 18 threads for OpenMP parallel HNSW search. The cache in each compute instance is configured to store only 10% of the total sub-HNSW clusters in the memory pool. At runtime, the batch size for vector queries is set to 2000.

Latency-recall curve evaluation. We evaluate d-HNSW and the baselines using the SIFT1M and GIST1M datasets, setting the top-k parameter as top-1 and top-10, respectively. All compute instances across three servers issue vector queries to the memory instance together.

Fig. 6 presents latency-recall curves for all three schemes with the varied $efSearch$ from 1 to 48. The $efSearch$ parameter determines the number of dynamic candidates maintained during the sub-HNSW search process. In the SIFT1M dataset with top-10 vector query shown in Fig. 6(a), d-HNSW reduces latency by up to 117× and 1.12× compared to naive

Scheme	Network	Sub-HNSW	Meta-HNSW
Naive d-HNSW	90271.2μs	6564.5μs	13.52μs
d-HNSW (w./o. doorbell)	607.5μs	287.0μs	9.97μs
d-HNSW	**527.6μs**	269.2μs	9.75μs

Table 1: Latency breakdown for SIFT1M@1 with ef-Search as 48.

Scheme	Network	Sub-HNSW	Meta-HNSW
Naive d-HNSW	422.9ms	35.3ms	61.1μs
d-HNSW (w./o. doorbell)	2.9ms	1.27ms	52.6μs
d-HNSW	**1.3ms**	1.48ms	46.9μs

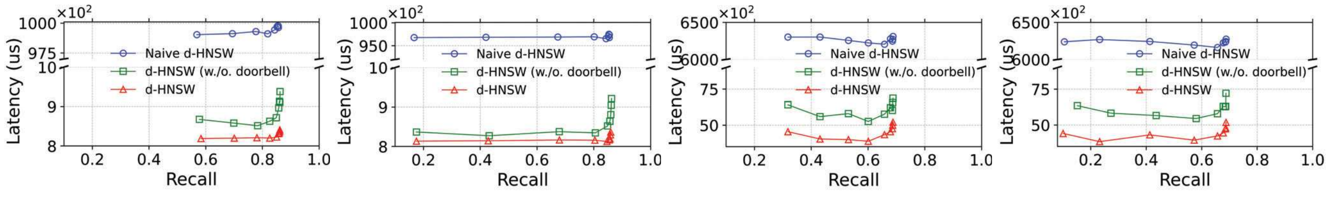
Table 2: Latency breakdown for GIST1M@1 with $efSearch$ as 48.

d-HNSW and d-HNSW without doorbell batching, respectively, while achieving a recall of approximately 0.86 when $efSearch$ reaches 48. The reason is naive d-HNSW issues an RDMA read round-trip to access each involved sub-HNSW cluster. Also, the doorbell batching further optimizes performance by batching memory accesses across multiple fragmented addresses within a single round-trip. For top-1 query in SIFT1M shown in Fig. 6(b), the upper recall reaches 0.85 when $efSearch$ is set to 48. The latency is further reduced across all schemes since only the closest vector needs to be selected.

Similarly, in the GIST1M dataset shown in Figs. 6(c)(d), d-HNSW achieves up to 121× and 1.30× lower latency compared to naive d-HNSW and d-HNSW without doorbell, respectively. Due to the higher dimensionality of GIST1M vectors, query latency is generally higher than in SIFT1M.

Latency breakdown of vector query. We break down the latency of each scheme to analyze the source of d-HNSW's performance advantage. The total latency of a vector query consists of three components: data transfer over the network, meta-HNSW (cache) computation, and sub-HNSW computation on loaded data. Table 1 presents the latency breakdown for the SIFT1M dataset with top-1 queries. d-HNSW benefits from significantly reduced network latency, measured at 527μs, which is 0.005× and 0.84× lower than that of naive d-HNSW and d-HNSW without doorbell, respectively. The number of round-trips per vector query are 3.547 for naive d-HNSW and 0.896 for d-HNSW w./o. doorbell, 4.75×10^{-3} for d-HNSW for SIFT1M. Similarly, as shown in Table. 2, d-HNSW also achieves the lowest network latency in the dataset GIST1M.

Latency with different # of sub-HNSWs. Figure 7 presents the latency breakdown across varying cluster configurations from 70 to 100. The results show that total system latency remains at stable levels (742-771us) regardless of the number of sub-HNSW clusters. While network latency increases with



(a) SIFT1M@10. (b) SIFT1M@1. (c) GIST1M@10. (d) GIST1M@1.

Figure 6: Latency-recall evaluation of d-HNSW and baselines.

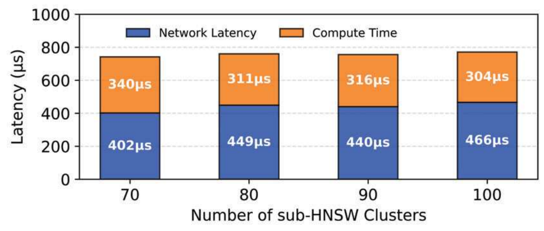


Figure 7: Latency breakdown with different # sub-HNSWs in SIFT1M@1.

more clusters from 402 to 466us, because we need to load more sub-HNSWs that are not cache hits, as each sub-HNSW contains fewer vectors.

5 RELATED WORK

Disaggregated memory system. Disaggregated memory systems have recently received essential attention due to enabling flexible resource allocation and improving hardware utilization in data centers. Existing work studies various solutions to managing and developing memory disaggregation from various systematic views, including architectural support [28, 31, 35, 37], operating systems [26, 32], KVCache management for LLMs [7, 22], disaggregated KV stores [12, 15–18, 27, 36, 38, 48], transactional systems [43, 44], in-network computation systems [1, 39, 40, 47]. d-HNSW is orthogonal to these works.

Approximate similarity search system. Approximate similarity search has become a fundamental technique for efficiently retrieving high-dimensional data vectors. Various algorithms have been developed to balance search efficiency and accuracy, like KD-trees [23], graph-based search structures [19], and quantization techniques [13]. Furthermore, advancements in hardware acceleration, such as GPU-based indexing [46] and CXL-based indexing [8], have further improved search performance. As data volumes continue to grow, optimizing vector search for both accuracy and resource efficiency remains an active research area. This includes adaptive search strategies [11, 42] and storage tier-aware optimizations [24] to meet different service level objectives (SLOs) [30, 45].

6 DISCUSSION

To improve the performance of RDMA-based disaggregated vector database further, we still face the following challenges on top of d-HNSW, and we leave them as our future work:

Unified memory address for storing index and vectors. d-HNSW adopts a shared-nothing architecture, partitioning vectors into different shards based on similarity at the load balancer layer, followed by the compute instance. This design eliminates consistency issues related to data insertion or updates across memory instances. To realize a fully disaggregated memory pool in the datacenter and store a huge graph in multiple memory instances, it is valuable to explore how to unify the memory address space across all memory instances.

Orchestrating vector search and network data transfer pipeline. To optimize end-to-end performance, we can orchestrate the vector search pipeline with the network data transfer. Specifically, we overlap the data transfer of the sub-HNSW index with the computation of both meta-HNSW and sub-HNSW search. By carefully managing the pipeline, the latency of transferring sub-HNSW data can be effectively hidden within the vector search time of the last batch, thus minimizing overall query latency.

7 CONCLUSION

We present d-HNSW, the first RDMA-based vector similarity search engine designed for disaggregated memory. d-HNSW enhances vector request throughput and minimizes data transfer overhead by implementing an RDMA-friendly data layout for memory nodes. Additionally, d-HNSW optimizes batched vector queries by eliminating redundant vector transfers for batched vector queries. We evaluate d-HNSW on various datasets and show its latency outperforms other RDMA-based baselines by up to 117× with top-10 benchmarking.

ACKNOWLEDGMENTS

We would like to thank our shepherd Youngjae Kim, and anonymous reviewers for their valuable feedback. The authors were partially supported by NSF Grants 2322919, 2420632, 2426031, and 2426940.

REFERENCES

[1] Xinyi Chen, Liangcheng Yu, Vincent Liu, and Qizhen Zhang. 2023. Cowbird: Freeing CPUs to Compute by Offloading the Disaggregation of Memory. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 1060–1073.

[2] DeepSeek. [n.d.]. https://www.deepseek.com/.

[3] Shiyuan Deng, Xiao Yan, KW Ng Kelvin, 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 (Big Data)*. IEEE, 1066–1071.

[4] Fire flyer file system. [n.d.]. https://github.com/deepseek-ai/3FS/.

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

[6] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[7] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. 2024. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565* (2024).

[8] 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 *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 585–600.

[9] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457* (2024).

[10] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. Design guidelines for high performance RDMA systems. In *2016 USENIX annual technical conference (USENIX ATC 16)*. 437–450.

[11] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.

[12] Pengfei Li, Yu Hua, Pengfei Zuo, Zhangyu Chen, and Jiajie Sheng. 2023. ROLEX: A Scalable RDMA-oriented Learned Key-Value Store for Disaggregated Memory Systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 99–114.

[13] A library for efficient similarity search and clustering of dense vectors. [n.d.]. https://github.com/facebookresearch/faiss.

[14] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. 2024. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516* (2024).

[15] Yi Liu, Minghao Xie, Shouqian Shi, Yuanchao Xu, Heiner Litz, and Chen Qian. 2024. Outback: Fast and Communication-efcient Index for Key-Value Store on Disaggregated Memory. *PVLDB* 18, 2 (2024), 335 – 348. https://doi.org/10.14778/3705829.3705849

[16] Baotong Lu, Kaisong Huang, Chieh-Jan Mike Liang, Tianzheng Wang, and Eric Lo. 2024. DEX: Scalable Range Indexing on Disaggregated Memory. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2603–2616.

[17] Xuchuan Luo, Jiacheng Shen, Pengfei Zuo, Xin Wang, Michael R Lyu, and Yangfan Zhou. 2024. CHIME: A Cache-Efficient and High-Performance Hybrid Index on Disaggregated Memory. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 110–126.

[18] Xuchuan Luo, Pengfei Zuo, Jiacheng Shen, Jiazhen Gu, Xin Wang, Michael R Lyu, and Yangfan Zhou. 2023. SMART: A High-Performance Adaptive Radix Tree for Disaggregated Memory. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 553–571.

[19] 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.

[20] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting network support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 313–326.

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

[22] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079* (2024).

[23] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 1378–1388.

[24] 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*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 10672–10684. https://proceedings.neurips.cc/paper_files/paper/2020/file/788d986905533aba051261497ecffcbb-Paper.pdf

[25] CloudLab: Flexible scientific infrastructure for research on the future of cloud computing. [n.d.]. https://www.cloudlab.us.

[26] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 69–87.

[27] Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, Yuxin Su, Jiazhen Gu, Hao Feng, Yangfan Zhou, and Michael R Lyu. 2023. Ditto: An elastic and adaptive memory-disaggregated caching system. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 675–691.

[28] Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, Tianyi Yang, Yuxin Su, Yangfan Zhou, and Michael R Lyu. 2023. FUSEE: A fully Memory-DisaggregatedKey-Value store. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 81–98.

[29] Wentao Shi, Xiangnan He, Yang Zhang, Chongming Gao, Xinyue Li, Jizhi Zhang, Qifan Wang, and Fuli Feng. 2024. Large language models are learnable planners for long-term recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1893–1903.

[30] Yongye Su, Yinqi Sun, Minjia Zhang, and Jianguo Wang. 2024. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[31] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. 2020. Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated Key-Value stores. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 33–48.

[32] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2020. Semeru: A Memory-Disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 261–280.

[33] Jianguo Wang, Eric Hanson, Guoliang Li, Yannis Papakonstantinou, Harsha Simhadri, and Charles Xie. 2024. Vector Databases: What's Really New and What's Next?(VLDB 2024 Panel). *Proceedings of the*

VLDB Endowment 17, 12 (2024), 4505–4506.

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

[35] Jianguo Wang and Qizhen Zhang. 2023. Disaggregated database systems. In *Companion of the 2023 International Conference on Management of Data*. 37–44.

[36] Qing Wang, Youyou Lu, and Jiwu Shu. 2022. Sherman: A write-optimized distributed b+ tree index on disaggregated memory. In *Proceedings of the 2022 international conference on management of data*. 1033–1048.

[37] Ruihong Wang, Jianguo Wang, Stratos Idreos, M Tamer Özsu, and Walid G Aref. 2022. The case for distributed shared-memory databases with rdma-enabled memory disaggregation. *arXiv preprint arXiv:2207.03027* (2022).

[38] Ruihong Wang, Jianguo Wang, Prishita Kadam, M Tamer Özsu, and Walid G Aref. 2023. dlsm: An lsm-based index for memory disaggregation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2835–2849.

[39] Zhonghua Wang, Yixing Guo, Kai Lu, Jiguang Wan, Daohui Wang, Ting Yao, and Huatao Wu. 2024. Rcmp: Reconstructing RDMA-Based Memory Disaggregation via CXL. *ACM Transactions on Architecture and Code Optimization* 21, 1 (2024), 1–26.

[40] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. 2023. Characterizing Off-path SmartNIC for Accelerating Distributed Systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 987–1004.

[41] Yipeng Xing, Yongkun Li, Zhiqiang Wang, Yinlong Xu, and John CS Lui. 2023. LightTraffic: On Optimizing CPU-GPU Data Traffic for Efficient Large-scale Random Walks. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 882–895.

[42] Minjia Zhang and Yuxiong He. 2019. Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1673–1682.

[43] Ming Zhang, Yu Hua, and Zhijun Yang. 2024. Motor: Enabling Multi-Versioning for Distributed Transactions on Disaggregated Memory. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 801–819.

[44] Ming Zhang, Yu Hua, Pengfei Zuo, and Lurong Liu. 2022. FORD: Fast one-sided RDMA-based distributed transactions for disaggregated persistent memory. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. 51–68.

[45] 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 23)*. 995–1011.

[46] Zili Zhang, Fangyue Liu, Gang Huang, Xuanzhe Liu, and Xin Jin. 2024. Fast Vector Query Processing for Large Datasets Beyond GPU Memory with Reordered Pipelining. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 23–40.

[47] Wenbin Zhu, Zhaoyan Shen, Qian Wei, Renhai Chen, Xin Yao, Dongxiao Yu, and Zili Shao. 2025. HiDPU: A DPU-Oriented Hybrid Indexing Scheme for Disaggregated Storage Systems. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. 271–285.

[48] Pengfei Zuo, Qihui Zhou, Jiazhao Sun, Liu Yang, Shuangwu Zhang, Yu Hua, James Cheng, Rongfeng He, and Huabing Yan. 2022. RACE: one-sided RDMA-conscious extendible hashing. *ACM Transactions on Storage (TOS)* 18, 2 (2022), 1–29.