Towards High-throughput and Low-latency Billion-scale Vector Search via CPU/GPU Collaborative Filtering and Re-ranking

Bing Tian, Haikun Liu, and Yuhang Tang, *Huazhong University of Science and Technology;* Shihai Xiao, *Huawei Technologies Co., Ltd;* Zhuohui Duan, Xiaofei Liao, and Hai Jin, *Huazhong University of Science and Technology;* Xuecang Zhang and Junhua Zhu, *Huawei Technologies Co., Ltd;* Yu Zhang, *Huazhong University of Science and Technology*

https://www.usenix.org/conference/fast25/presentation/tian-bing

This paper is included in the Proceedings of the 23rd USENIX Conference on File and Storage Technologies.

February 25–27, 2025 • Santa Clara, CA, USA

ISBN 978-1-939133-45-8

Open access to the Proceedings of the 23rd USENIX Conference on File and Storage Technologies is sponsored by

Towards High-throughput and Low-latency Billion-scale Vector Search via CPU/GPU Collaborative Filtering and Re-ranking

Bing Tian[†], Haikun Liu[†,*], Yuhang Tang[†], Shihai Xiao[‡], Zhuohui Duan[†], Xiaofei Liao[†], Hai Jin[†], Xuecang Zhang[‡], Junhua Zhu[‡], Yu Zhang[†]

[†]National Engineering Research Center for Big Data Technology and System,
Service Computing Technology and System Lab/Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, China
[‡]Huawei Technologies Co., Ltd

Abstract

Approximate nearest neighbor search (ANNS) has emerged as a crucial component of database and AI infrastructure. Ever-increasing vector datasets pose significant challenges in terms of performance, cost, and accuracy for ANNS services. None of modern ANNS systems can address these issues simultaneously. In this paper, we present FusionANNS, a high-throughput, low-latency, cost-efficient, and high-accuracy ANNS system for billion-scale datasets using SSDs and only one entry-level GPU. The key idea of FusionANNS lies in CPU/GPU collaborative filtering and re-ranking mechanisms, which significantly reduce I/O operations across CPUs, GPU, and SSDs to break through the I/O performance bottleneck. Specifically, we propose three novel designs: (*1*) *multi-tiered indexing* to avoid data swapping between CPUs and GPU, (*2*) *heuristic re-ranking* to eliminate unnecessary I/Os and computations while guaranteeing high accuracy, and (*3*) *redundant-aware I/O deduplication* to further improve I/O efficiency. We implement FusionANNS and compare it with the state-of-the-art SSD-based ANNS system–SPANN and GPU-accelerated in-memory ANNS system–RUMMY. Experimental results show that FusionANNS achieves 1) 9.4-13.1× higher *query per second* (QPS) and 5.7-8.8× higher cost efficiency compared with SPANN; 2) and 2-4.9× higher QPS and 2.3-6.8× higher cost efficiency compared with RUMMY, while guaranteeing low latency and high accuracy.

1 Introduction

Approximate nearest neighbor search (ANNS) in high-dimensional spaces refers to find top-k vectors most similar to a given query vector. It has a wide range of applications in many fields, including data mining [1], search engines [2], and AI-driven recommendation systems [3, 4]. Specifically, fueled by the recent prosperity of *Large Language Models* (LLMs) [5–8], ANNS systems have become a crucial component of modern AI infrastructure. Figure 1 shows a typical framework of *Retrieval Augmented Generation* (RAG). The domain-specific knowledge is first embedded as high-dimensional vectors and stored in a vector database. When a chatbot receives a query, it uses the ANNS engine to retrieve the most relevant knowledge from the vector database, allowing the LLM to use that knowledge as additional context for more accurate inference.

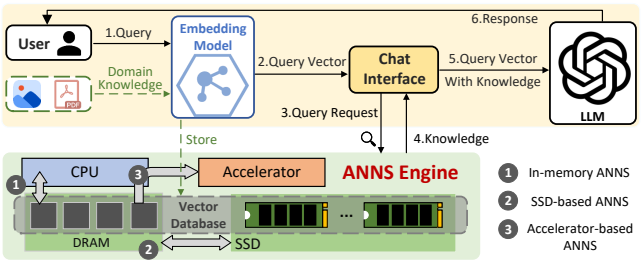


Figure 1: The framework of retrieval augmented generation

ANNS is a typical **memory-hungry** and **compute-intensive** application. Most ANNS systems [9–13] exploit *inverted file* (IVF) [14–16] or graph-based [12, 13, 17] indices to facilitate ANNS. For billion-scale datasets, these indices usually require a large amount of memory resource. For example, state-of-the-art IVF-based RUMMY [9] and graph-based Bang [10] require terabyte-scale memory space to accommodate billion-scale vectors and their indices. The substantial memory demand significantly increases the *total cost of ownership* (TCO), impeding ANNS scaling to extremely large datasets (e.g., hundreds of billions of vectors). Despite the huge memory requirement, ANNS is also computationally intensive because it requires massive distance calculations among vectors, especially for large-scale datasets in high-dimensional spaces. With a rapid growth of the vector database, ANNS has emerged as a new performance bottleneck in RAG scenarios [9], potentially accounting for about 50% of the total latency for an LLM query [18].

To reduce the cost of memory required by ANNS, there are mainly two kinds of approaches, i.e., *Hierarchical Indexing* (HI) [15, 19] and *Product Quantization* (PQ) [20, 21]. **First, the hierarchical indexing approach reduces memory con-**

*Corresponding author: Haikun Liu (hkliu@hust.edu.cn)

sumption by storing indices [15,22] on SSDs. Typically, Microsoft's commercial ANNS system–SPANN [14, 15] stores all IVF-based indices (i.e., posting lists) on SSDs, and maintains the centroids of these posting lists in memory using a navigation graph. Although SPANN achieves low latency, we find that its throughput for concurrent queries is quite limited, peaking at only four CPU threads on a high-end SSD (Section 2.1). The limited scalability hampers its practicality for AI applications requiring high throughput. **Second, PQ is another effective way for memory cost saving**. This vector compression technology can significantly reduce the memory footprint of high-dimensional vectors by up to 95%, and can also accelerate the ANNS speed by several times [20]. However, since PQ is a lossy-compression scheme, a higher compression rate often implies a lower query accuracy. It is usually unacceptable for some scenarios that require high accuracy (e.g. recall$\geq$ 90%) [23].

To address the computing challenge, GPUs have been increasingly leveraged to accelerate extensive distance calculations involved in ANNS. Recent GPU-based ANNS solutions [9, 10, 20, 24–26] have demonstrated high efficiency for handling small datasets that fit within the GPU's *high bandwidth memory* (HBM). However, for billion-scale datasets, the GPU-based approach may suffer from significant performance degradation. Our experiments show that the performance of ANNS even declines by 10% when SPANN directly adopts GPUs for distance calculations (Section 2.3). The root cause is that the limited capacity of HBM causes extensive data movement across GPU's HBM, host memory, and SSDs.

Although the above approaches can address some of the performance/cost/accuracy issues to some extent, none of them can offer high throughput, low latency, cost efficiency, and high accuracy simultaneously for billion-scale ANNS services. Intuitively, one can adopt hierarchical indexing, product quantization, and GPU acceleration techniques together to achieve an optimal ANNS solution. However, we find that the combination of these techniques causes even worse performance than SPANN which exploits hierarchical indexing solely (Section 2.3). Overall, there remain several challenges to collaborate hierarchical indexing with product quantization in a GPU-accelerated ANNS system.

Challenge 1: To improve query accuracy and efficiency, most ANNS systems [14, 15] exploit a replication strategy to build high-quality IVF indices, where boundary vectors are replicated into adjacent posting lists. This can significantly expand the size of indices by 8$\times$ larger than that of raw vectors [14, 15]. Even these indices are compressed with PQ, the GPU's HBM still cannot accommodate all compressed indices, resulting in extensive data swapping between GPU and CPUs. **Challenge 2:** Since PQ incurs non-trivial accuracy loss, it is often associated with a vector re-ranking process to improve the query accuracy. However, since the accuracy loss varies significantly among different compressed vectors, it is challenging to determine the minimum number of vectors that requires re-ranking for each query under a given accuracy constraint. **Challenge 3:** Since a raw vector (128$\sim$384 bytes) is much smaller than the minimum read granularity (4 KB) of modern NVMe SSDs, each request for raw vectors often causes significant read amplification, resulting in low I/O efficiency during re-ranking.

In this paper, we present FusionANNS, a "CPU + GPU" cooperative processing architecture for billion-scale ANNS. FusionANNS achieves high throughput, low latency, cost efficiency and high accuracy simultaneously using only one entry-level GPU. The key idea of FusionANNS is to minimize data movement across GPU, CPUs, and SSDs via CPU/GPU collaborative filtering and re-ranking. Specifically, we propose three novel designs to tackle the above challenges.

First, we propose a novel *multi-tiered index structure* to enable CPU/GPU collaborative filtering. FusionANNS stores (*i*) raw vectors on SSDs, and (*ii*) compressed vectors using PQ in the GPU's HBM, while maintaining (*iii*) only vector-IDs of each posting list and a navigation graph in host memory. Since the HBM only stores highly-compressed PQ-vectors rather than compressed posting lists, it can accommodate all compressed vectors in billion-scale datasets. Upon a query, the host CPU first traverses the in-memory navigation graph to find the top-*m* nearest posting lists, and then only transmit their vector-IDs (excluding the vectors' content) to GPU for distance calculations. In this way, FusionANNS can significantly reduce data transmission between CPUs and GPU.

Second, we propose *heuristic re-ranking* to improve the query accuracy while avoiding unnecessary I/O operations and distance calculations. We split the re-ranking process into multiple mini-batches and execute them sequentially. Once a mini-batch is finished, we exploit a lightweight feedback control model to check whether subsequent mini-batches are beneficial for improving the query accuracy, and terminate the re-ranking process immediately if successive mini-batches have little contribute to the query accuracy.

Third, we propose *redundancy-aware I/O deduplication* to further improve the I/O efficiency during re-ranking. We store vectors with high similarity compactly to improve the spatial locality on SSDs. This optimized storage layout enables two I/O deduplication mechanisms: 1) merging multiple I/Os mapped to the same page of SSDs within a mini-batch to mitigate read amplification, 2) fully exploiting the DRAM buffer to eliminate redundant I/Os in subsequent mini-batches.

Overall, we make the following contributions:

- We design FusionANNS, the first GPU-accelerated SSD-based ANNS system that achieves high throughput, low latency, cost efficiency and high accuracy simultaneously for billion-scale datasets.
- For **Challenge 1**, we propose a novel *multi-tiered index* that enables GPU/CPU collaborative filtering to significantly reduce data transmission between CPUs and GPU.

- For **Challenge 2**, we propose *heuristic re-ranking* to eliminate unnecessary I/Os and computations during re-ranking.
- For **Challenge 3**, we propose *redundancy-aware I/O deduplication* based on the optimized storage layout to further enhance I/O efficiency.
- We evaluate FusionANNS using a general purpose server equipped with an entry-level GPU. Experimental results show that FusionANNS improves QPS by up to 13.1× and 4.9×, and enhances cost efficiency by up to 8.8× and 6.8×, compared with the state-of-the-art SSD-based system–SPANN and GPU-accelerated in-memory system–RUMMY, respectively, while guaranteeing low latency and high accuracy.

2 Background and Motivation

In this section, we first introduce two kinds of ANNS indexing techniques and product quantization (PQ) for vectors. Then, we present our main idea and analyze its key challenges, which motivate the design of FusionANNS.

2.1 Indexing Techniques for ANNS

Most ANNS algorithms exploit a distance metric such as Euclidean distance to find the top-k nearest neighbors for a given query vector. For high-dimensional large-scale datasets, it is computationally costly due to the curse of dimensionality [27]. To address this issue, most ANNS algorithms [16, 17, 28–32] exploit indexing techniques to prune data regions that are unlikely to contain the nearest neighbors. These indices can significantly improve the query performance by shrinking the search space, but significantly increases memory consumption, especially for large datasets. Among various indexing techniques, IVF [15, 16, 31, 32] and graph-based [17, 33] indices are widely used due to their high efficiency.

The **graph-based index** often organizes vectors in a proximity graph structure, in which vertices and edges represent vectors and distances between two vertices, respectively. Upon a query, the ANNS engine traverses the graph from a given vertex to find the top-k nearest neighbors. DiskANN [22] is a typical graph-based ANNS solution. It uses SSDs to store graph indices of billion-scale datasets while keeping some frequently-accessed vertices in main memory. Although DiskANN is a memory-efficient ANNS solution, it experiences high latency for queries due to rather long iteration paths for large-scale datasets.

The **IVF index** is a popular indexing technique for large-scale datasets stored on SSDs. To create the IVF index, a dataset is often partitioned into many posting lists using a clustering algorithm [34], and each posting list is represented by its centroid. Recent studies [9, 35] have demonstrated that the IVF index [15] is more efficient than the state-of-the-art graph-based index [22] for billion-scale datasets. SPANN [15]

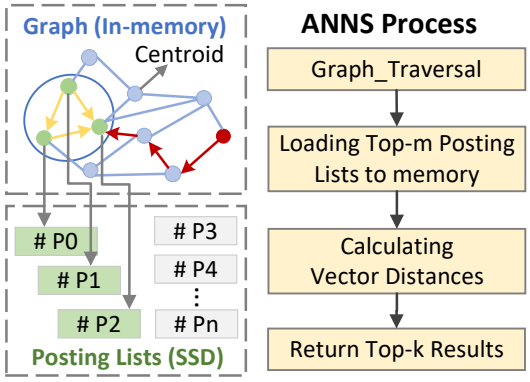
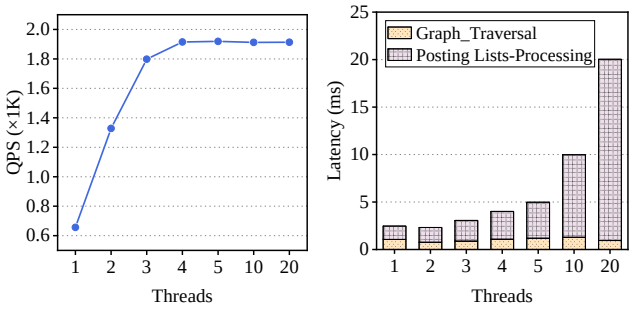


Figure 2: The hierarchical indexing technique in SPANN



(a) Throughput (b) Latency

Figure 3: The throughput and latency of SPANN

is a state-of-the-art billion-scale ANNS system using the IVF index. Unlike conventional IVF indices, SPANN builds an advanced IVF index by replicating boundary vectors of clusters into adjacent posting lists. This replication mechanism significantly expands the size of vector indices by 8×, but improves the query accuracy and efficiency. As shown in Figure 2, SPANN stores all posting lists on SSDs while maintaining the centroids of these posting lists in memory using a graph index. Upon a query, SPANN traverses the in-memory graph to identify top-m nearest posting lists and loads them to host main memory. Then, it finds the top-k ($k < m$) nearest neighbors within these m posting lists via distance calculations.

Although SPANN achieves low latency comparable to in-memory ANNS approaches, we find that its throughput for concurrent queries is quite limited. As shown in Figure 3a, SPANN achieves the peak QPS using only four CPU threads, and its throughput can not scale with more threads. We count the query latency in two stages: (*i*) graph traversal in memory, and (*ii*) processing posting lists from SSD. Figure 3b shows that the query latency increases almost linearly with the number of threads. However, the latency of graph traversal almost remains stable, whereas the latency of processing posting lists increases significantly with the number of threads. The reason is that multiple queries concurrently read many and large-size posting lists from SSDs, resulting in severe I/O contention and high latency.

2.2 Product Quantization

To reduce the size of vector indices and computational costs for large-scale datasets, *product quantization* (PQ) [32] has been explored recently for compressing high-dimensional vectors. Assume a dataset containing N vectors is compressed with PQ, these vectors are first divided evenly into M subspaces, and each contains N sub-vectors. Then, these sub-vectors are clustered to generate a codebook, which contains a set of centroids of all clusters. The codebook allows each sub-vector to be approximated by its nearest centroid. The number of clusters per sub-space is typically set to 256, allowing each cluster ID to be represented by one byte. Once all codebooks are generated, each vector can be compressed into an M-byte PQ code. Upon a query, a distance lookup table is first generated, including all distances between a sub-query-vector and centroids per sub-space. Then, the approximate distance between the query vector q and a compressed vector v can be formulated as:

$$\widehat{dist}(q,v) = \sum_{i=1}^{M} dist(q_i, c_i(v_i)) \quad (1)$$

where M denotes the total number of sub-spaces, q_i denotes the i-th sub-query-vector, and $c_i(v_i)$ denotes the centroid of the i-th sub-space of the compressed vector. Thus, the distance between q_i and $c_i(v_i)$ can be easily retrieved by looking up the distance table using the PQ code as the address. Finally, the distance between q and v can be summed up with all $dist(q_i, c_i(v_i))$.

Essentially, PQ converts a distance calculation between vectors into multiple memory access operations, and thus poses a significant challenge for traditional CPU-based computing architectures due to the relatively low bandwidth of DRAM and limited parallelism. Therefore, the PQ is usually accelerated by GPUs [21, 32] because it can fully utilize their massive GPU kernels and high bandwidth memory to improve the query performance.

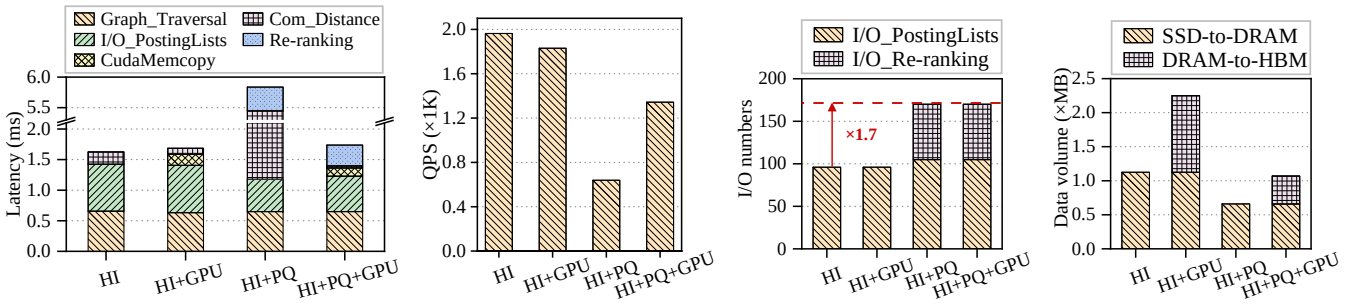
2.3 Main Idea and Challenges

To circumvent the challenges of substantial computing and memory resource requirements posed by billion-scale datasets, our goal is to design a high-throughput, low-latency, cost-efficient, and high-accuracy ANNS system using SSDs and an entry-level GPU. However, a significant challenge for designing a GPU-accelerated ANNS system is that the limited capacity of GPU's HBM causes extensive data swapping between GPU and CPUs, significantly degrading the ANNS performance for large-scale datasets.

A Straightforward Solution using PQ and HI. Fortunately, the PQ technique can significantly reduce the memory footprint of vectors, thereby alleviating the performance bottleneck associated with data transmission between CPUs and GPUs. As a result, PQ has the potential to fully harness the capabilities of GPUs to accelerate distance calculations involved in ANNS [20]. Here, we first discuss a straightforward GPU-accelerated ANNS solution using PQ and *hierarchical indexing* (HI) techniques. Except that all vectors are compressed using PQ, this straightforward solution uses the same hierarchical indices as SPANN. Upon a query, the ANNS engine first traverses the navigation graph to identify top-m nearest posting lists, and loads these compressed posting lists to the GPU's HBM for distance calculations. Then, GPU finds the top-n candidate vectors by calculating the distance between the query vector and each compressed vector in the top-m nearest posting lists. Since PQ has a negative impact on the query accuracy, these intermediate results obtained by the GPU should be re-ranked to improve the query accuracy. During re-ranking, the raw data of the top-n candidate vectors should be compared with the query vector to find the final top-k nearest neighbours.

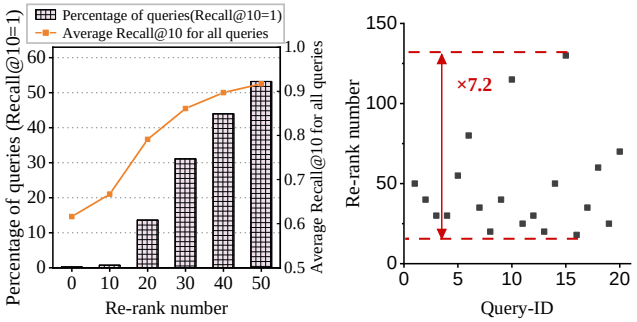
Observations. Disappointingly, we find that the above solution does not achieve expected high performance. To better understand the root causes, we conduct four different experiments to evaluate three combinations of HI, PQ, and GPU acceleration techniques. In all experiments, different ANNS systems have to meet the same level of query accuracy. As shown in Figure 4a, neither the PQ nor the GPU acceleration can reduce the end-to-end query latency compared with the HI proposed by SPANN. Although "HI+GPU" can significantly reduce the latency of distance calculations, the overhead of transferring posting lists between CPUs and the GPU (i.e., CudaMemcpy) offsets the benefits of the GPU acceleration. For "HI+PQ", it still uses CPUs to process PQ-based posting lists. Since vectors are compressed using PQ, the I/O latency due to loading PQ-based posting lists from SSDs to main memory is reduced. However, the CPU faces a new challenge in calculating distances between the query vector and compressed vectors due to intensive memory accesses, resulting in a significant increase of the end-to-end query latency. For "HI+PQ+GPU", the latency of distance calculations is reduced to an extremely low level. However, the CudaMemcpy and the additional re-ranking process incur substantial overheads, offsetting the benefits of using GPU. Moreover, none of these combinations achieve higher throughput than the original SPANN using HI solely, as shown in Figure 4b. Particularly, a direct adoption of PQ to SPANN even significantly reduces its QPS by 65%.

Root Causes. To reveal the root cause of such performance degradation, we measure the I/O numbers and the data volume transferred across SSDs, main memory, and GPU's HBM required by each ANNS query on average. As shown in Figure 4c, although the PQ technique significantly reduces the I/O size of posting lists from 12∼48 KB to a page granularity (4 KB), it increases the number of I/Os by 70% due to the re-ranking process. As a result, the I/O performance bottleneck shifts from the SSD's bandwidth to its *input/output operations per second* (IOPS). Moreover, a large volume of posting lists are transferred between CPUs and GPU, thereby offsetting the benefit of GPU acceleration, as shown in Figure 4d.



(a) Latency per query (b) Throughput (c) I/O numbers per query (d) Data transmission per query

Figure 4: Three combinations of *hierarchical indexing* (HI), *product quantization* (PQ), and GPU acceleration



(a) Accuracy vs. re-ranking numbers (b) Re-ranking number per query

Figure 5: Differential characterization between queries

Challenge 1: Overall, the combinations of HI, PQ, and GPU acceleration techniques cause even higher latency and lower throughput than SPANN that adopts HI solely. The root cause is that the GPU's HBM still cannot accommodate all posting lists compressed by PQ, allowing extensive data transmission between GPU and CPUs to become a new performance bottleneck. **Without a sophisticated design of the data layout across different devices and a careful collaboration among these three techniques, it is impossible to fully realize the GPU's potential for ANNS acceleration**.

Challenge 2: To achieve the same level of query accuracy, a re-ranking process is usually required to refine intermediate results generated by the GPU. The number of the top-n vectors that should be re-ranked (i.e., the re-ranking number) is usually several times larger than the final top-k nearest neighbors. To evaluate the impact of the re-ranking number on the query accuracy, we execute 10,000 queries by linearly increasing the re-ranking number. As shown in Figure 5a, when the re-ranking number is set to 40, about 42% of queries get the accurate top-10 nearest neighbors under $Recall@10 = 1.0$, while all queries achieve an accuracy level[1] of $Recall@10 = 0.9$ on average. In this case, it is only beneficial to increase the re-ranking number for straggler queries. Moreover, we find that the minimum re-ranking numbers are very distinct for different ANNS queries, as shown in Figure 5b. This significant variance usually causes unnecessary I/O operations and distance calculations if the number of re-ranked vectors is fixed for all queries. However, **it is challenging to determine the minimum re-ranking number for each query under a given accuracy constraint**.

Challenge 3: The re-ranking process introduces a number of I/O requests to raw vectors on SSDs. The size of a raw vector generally ranges from 128 to 384 bytes, while the smallest operating unit of modern NVMe SSDs is typically a page (4 KB). **This mismatch in granularity causes significant read amplification, resulting in extremely low I/O efficiency during re-ranking.** Fortunately, we find these vectors requiring re-ranking usually are highly similar to each other. This similarity offers an opportunity to mitigate the read amplification by reorganizing the data layout on SSDs.

3 FusionANNS Design

In this section, we present key designs of FusionANNS, i.e., multi-tiered indexing, CPU/GPU collaborative filtering, heuristic re-ranking, and redundant-aware I/O deduplication.

3.1 Multi-tiered Indexing

Figure 6 illustrates the structure of multi-tiered indices residing in host main memory, GPU's HBM, and SSDs. We construct multi-tiered indices in an offline manner. At first, we adopt the hierarchical balanced clustering algorithm [34] to iteratively partition the dataset into several posting lists. Each posting list contains multiple vector-IDs and the corresponding vector content. We follow SPANN to configure the number of posting lists (about 10% of the total number of vectors in a datasets) and use the same replication mechanism to address the boundary concern [15]. Specifically, when a vector lies on the boundary of multiple clusters, we assign this boundary vector to a cluster according to Equation 2.

$$v \in C_i \Leftrightarrow \text{Dist}(v, C_i) \leq (1+\varepsilon) \times \text{Dist}(v, C_1) \qquad (2)$$

where v represents the vector to be assigned, C_i denotes the i-th clusters. Particularly, C_1 represents the cluster that is clos-

[1] $Recall@k$ refers to the proportion of relevant vectors that are retrieved within the ground-truth top k nearest neighbors.

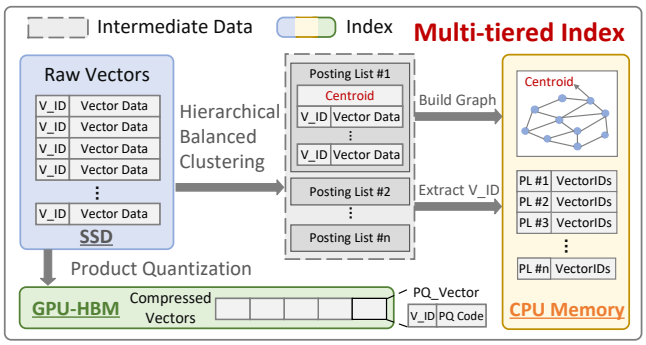


Figure 6: Mutil-tiered indices in FusionANNS

est to the vector v. The parameter ε determines the maximum distance in which a vector should be assigned simultaneously to multiple clusters. To balance the query accuracy and efficiency, each vector is assigned to eight clusters at most [15].

In-memory Indices. After the dataset is clustered, we build a graph index based on SPTAG [36] using the centroids of all posting lists and store it in main memory. With this navigation graph, FusionANNS can efficiently identify the top-m nearest posting lists for a query vector. This graph is constructed by continuously adding new vectors to an empty graph. When a vector is added as a new vertex, new edges are created to connect this newly-added vertex with its top-k (typically 64) nearest neighbors. Then, its neighboring vertices should update their nearest neighbors to limit the maximum number of edges. Unlike SPANN that stores all posting lists on SSDs, we extract only vector-IDs of each posting list as metadata (excluding vector content) and store it in memory, as shown in Figure 6. When the graph index and metadata are generated, the intermediate posting lists can be discarded. Since the memory footprint of the graph and metadata is relatively small, FusionANNS can support billion-scale ANNS in a memory cost-efficient way using general-purpose servers.

PQ-based Vectors in GPU's HBM. Since PQ can significantly reduce the memory footprint of high-dimensional vectors via lossy-compression, even an entry-level GPU such as NVIDIA V100 with 32 GB HBM can accommodate all compressed vectors in its HBM for billion-scale datasets. In FusionANNS, we pin all compressed vectors in the HBM, avoiding extensive data swapping between GPUs and CPUs that is commonly experienced in previous GPU-accelerated ANNS systems [9, 20, 24]. Since in-memory indices still remain the benefit of the replication mechanism for boundary vectors, FusionANNS can efficiently obtain all IDs of candidate vectors, and then sends these vector-IDs (excluding vectors' content) to the GPU for distance calculations. In this way, FusionANNS also eliminates the performance bottleneck caused by the limited PCIe bandwidth between CPUs and GPUs.

Raw Vectors on SSDs. Unlike IVF-based SPANN that stores all posting lists on SSDs, FusionANNS only needs to store raw vectors on SSDs for re-ranking. Since the volume of raw vectors is almost 8 times smaller than that of posting lists, FusionANNS can significantly reduce the storage consumption. For each query, since only the re-ranking process lead to a few I/O requests, FusionANNS can also alleviate the I/O bottleneck of SSDs for concurrent queries.

Remarks. Previous SSD-based ANNS systems such as SPANN [15] and DiskANN [22] cannot utilize GPU to accelerate ANNS since GPU cannot reduce I/O requests for each ANNS query but introduces significant performance overhead of data movement. In contrast, our multi-tiered indexing approach can significantly reduce the storage footprints on HBM, main memory, and SSDs. It also significantly reduces the amount of data transferred across SSDs, CPUs, and GPUs.

3.2 CPU/GPU Collaborative Filtering

The multi-tiered indices enable CPU/GPU collaborative filtering for ANNS queries. Figure 7 illustrates the system architecture of FusionANNS and the workflow of an ANNS query. Upon a vector query, FusionANNS first utilizes the GPU to generate the query vector's distance table for subsequent PQ distance calculations (❶). Meanwhile, the CPU traverses the in-memory navigation graph to identify the top-m posting lists nearest to the query vector (❷). Then, the CPU consults the metadata to collect vector-IDs within these candidate posting lists (❸). After that, the CPU transfers these vector-IDs to the GPU and invokes GPU kernels (❹) for further processing.

When the GPU receives vector-IDs, it first deduplicates them using a parallel hash module (❺). For each vector-ID, the GPU reads the corresponding PQ vector from HBM and computes the PQ distance between it and the query vector. In this step, the GPU allocates a thread for each dimension of the PQ vector to access the corresponding precomputed value in the distance table. Then, a coordinator thread accumulates these values to count the PQ distance (❻) for each candidate vector. Subsequently, the GPU sorts all distances in ascending order and returns the top-n vectors' ID to the CPU (❼). Since the intermediate results obtained using PQ vectors are not precise, the CPU reads the raw vectors according to these vector-IDs from SSDs for further re-ranking (❽). Finally, the CPU returns the final top-k nearest neighbors.

Remarks. This CPU/GPU collaborative filtering mechanism can filter out most irrelevant vectors throughput two rounds of retrieving, and thus can significantly reduce the number of I/O requests to SSDs during the re-ranking stage.

3.3 Heuristic Re-ranking

Since PQ causes an accuracy loss during distance calculations, a re-ranking process is usually required to refine the intermediate results reported by the GPU. As mentioned in Section 2.3, to achieve the same level of query accuracy, the minimum re-ranking numbers for different ANNS queries usually vary significantly. Thus, a static configuration of the

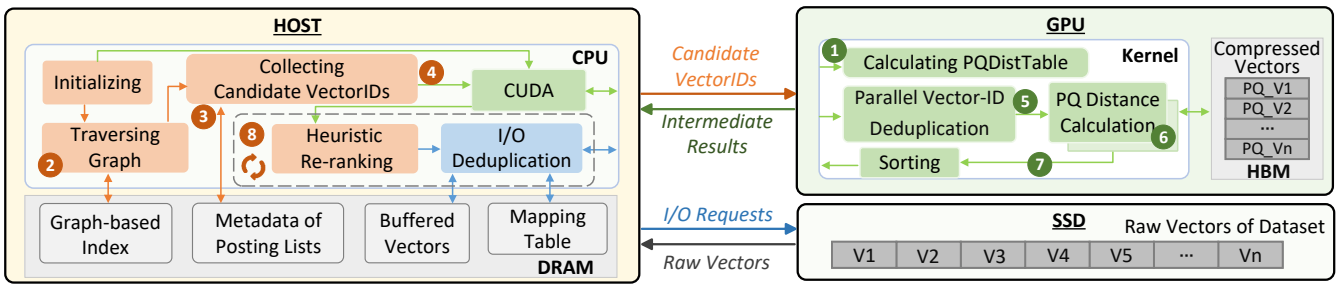


Figure 7: The FusionANNS architecture and the workflow of an ANNS query

Algorithm 1: Heuristic Re-ranking

Input: $Tasks, BatchSize, k, \varepsilon, \beta$
Output: Q

1 *Initialize* $Q \leftarrow NULL$
2 *Initialize* $StabilityCounter \leftarrow 0$
3 **for** $i = 0; i < Tasks.size(); i += BatchSize$ **do**
4 $S_{n-1} \leftarrow Q.GetVectorIDs()$
5 **for** $j = i; j < i + BatchSize; j++$ **do**
/* Tasks[j] involves I/Os and computations */
6 $Candidate\_Vector \leftarrow GetDistance(Tasks[j])$
7 $Q.insert(Candidate\_Vector)$
8 **end**
9 $S_n \leftarrow Q.GetVectorIDs()$
10 $\Delta \leftarrow \frac{|S_n - S_n \cap S_{n-1}|}{k}$ /* Calculating change rate Δ */
11 **if** $\Delta \leq \varepsilon$ **then**
12 $StabilityCounter \leftarrow StabilityCounter + 1$
13 **if** $StabilityCounter \geq \beta$ **then**
14 **return** Q /* Terminate re-ranking */
15 **end**
16 **else**
17 $StabilityCounter \leftarrow 0$
18 **end**
19 **end**
20 **return** Q

re-ranking number may cause unnecessary I/O operations and distance calculations, or results in an accuracy loss. Previous in-memory ANNS systems such as LanceDB [37] re-rank the same number of candidate vectors for simplicity because the fast memory can tolerate some unnecessary data accesses. However, this simple approach is inefficient for SSD-based ANNS systems because massive-yet-small I/O requests initiated by concurrent ANNS queries can cause significant performance degradation due to the limited IOPS of SSDs.

To circumvent this problem, we propose a *heuristic re-ranking* mechanism to minimize I/O operations and distance calculations. The key idea is to set a relative large re-ranking number conservatively for high accuracy, and to terminate the re-ranking process immediately once the subsequent search is no longer beneficial for improving the query accuracy. To achieve this goal, we divide the re-ranking process into multiple mini-batches and execute them sequentially. Each mini-batch contains the same number of candidate vectors. Since all candidate vectors are sorted with their distances in ascending order, the mini-batch executed earlier usually has a higher possibility to identify more vectors that belong to the final top-k nearest neighbours. Once a mini-batch is finished, we exploit a lightweight feedback control model to check whether subsequent mini-batches are beneficial for improving the query accuracy.

To simplify the problem, we use a priority queue Q (i.e., a max-heap) to maintain the current top-k nearest neighbours. Initially, the max-heap is empty. For each mini-batch, we calculate the distances between the query vector and vectors within this mini-batch, and insert the vector whose distance is less than the current maximum distance in the max-heap. When a mini-batch is finished, we calculate the change rate of the max-heap according to Equation 3:

$$\Delta = \frac{|S_n - S_n \cap S_{n-1}|}{k} \tag{3}$$

where S_n and S_{n-1} represent the sets of vectors' IDs in the max-heap when the mini-batch n and the mini-batch n-1 are just completed, respectively. k represents the number of vectors maintained in the max-heap, i.e., the number of the final nearest neighbors. We terminate the re-ranking process if the change rate of the max-heap for successive mini-batches is smaller than a given threshold ε for β times continuously.

Algorithm 1 presents the pseudo-code for the heuristic re-ranking. We first initialize the max-heap Q as NULL, and use a *StabilityCounter* to record the times that the change rate remains lower than ε continuously. The size of *Tasks* denotes the total number of vectors should be re-ranked in-batch. The parameter *BatchSize* denotes the number of candidate vectors in a mini-batch. For each mini-batch, we retrieve the top-k vectors' IDs from Q before processing tasks in this mini-batch (line 4). Then, we sequentially perform each task including reading the raw vector from SSDs, calculating its distance to the query vector, and inserting this vector into Q if its distance is less than the maximum value in the max-heap. When a mini-batch is finished, we collect the IDs of updated top-k vectors from Q, and calculate the change rate Δ of Q between

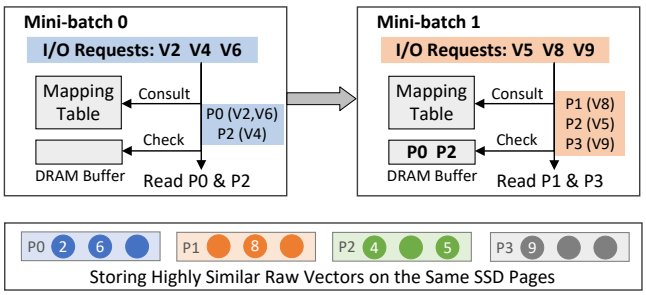


Figure 8: Optimized data layout and I/O deduplication

these two successive mini-batches (line 9-10). If the change rate is lower than the given threshold ε, we increase the *StabilityCounter* by one. Once the *StabilityCounter* becomes larger than the given threshold β, the re-ranking process is terminated. In contrast, if the change rate exceeds the threshold ε, we reset the *StabilityCounter* and continue the following mini-batches. At last, we return the final top-k vectors in Q. To achieve the optimal performance, we set ε, β, and *BatchSize* as 0.1, 1, and k, respectively according to our experimental studies.

Remarks. Our heuristic re-ranking algorithm can minimize I/O requests to SSDs and CPU resource consumption for distance calculation while guaranteeing high accuracy, and eventually reduces the latency of re-ranking.

3.4 Redundant-aware I/O Deduplication

FusionANNS uses raw vectors on SSDs to re-rank the intermediate results returned by the GPU. A straightforward approach is to store all raw vectors sequentially on SSD pages. However, since a raw vector (128∼384 bytes) is quite smaller than the page granularity (4KB), individual requests to these raw vectors often result in significant read amplification. Moreover, since the re-ranking process introduces a lot of random and small I/O operations, the I/O latency has a crucial impact on the end-to-end query latency. Like most SSD-based ANNS systems [15], we adopt Direct I/O [38, 39] to fully exploit the low-latency property of modern NVMe SSDs. To further improve I/O efficiency, we first optimize the data layout to improve the spatial locality on SSDs. Then, we exploit *redundancy-aware I/O deduplication* to mitigate the effect of read amplification.

Optimized Storage Layout. Although the vectors requiring re-ranking are obtained by PQ distances, they are highly similar to the query vector, allowing them usually spatially close to each other. This similarity offers an opportunity to mitigate the read amplification by carefully organizing the data layout on SSDs. Specifically, when the in-memory indices are created offline, for each centroid in the navigation graph, we use a bucket to store a number of raw vectors that are closest to the centroid. We note that there are not duplicate vectors among buckets. For each bucket, if it does not align with SSD pages, we combine buckets based on the size of unaligned portions using a max-min algorithm [40] to minimize the free space on a SSD page. Finally, we group all buckets as a single file and store it on SSDs, and use a table in memory to maintain the mappings between vectors and SSD pages.

Intra- and Inter- Mini-batch I/O Deduplication. Empowered by the optimized data layout, we design two I/O deduplication mechanisms, including merging I/Os mapped to the same SSD page within a mini-batch, and exploiting the DRAM buffer to eliminate redundant I/Os in subsequent mini-batches. Here, we use a simple example to describe these mechanisms, as shown in Figure 8. Assume that there are two mini-batches in the re-ranking process, where the tasks of *mini-batch 0* is to re-rank vectors: *V2*, *V4*, and *V6*. The tasks of *mini-batch 1* is to re-rank vectors:*V5*, *V8*, and *V9*. When *mini-batch 0* is executed, it first consults the mapping table to obtain the SSD page-IDs corresponding to the requested vectors. Since both *V2* and *V6* are stored in the same SSD page *P0*, we can merge these two I/O requests and only read one SSD page to get *V2* and *V6*. Since *P0* and *P2* do not exist in the DRAM buffer, we directly read them to the DRAM buffer via two I/O requests. In *mini-batch 1*, although *V5*, *V8*, and *V9* are stored in different SSD pages, the DRAM buffer already contains *P2* which includes *V5*. Therefore, *mini-batch 1* only needs to read *P1* and *P3* via two I/O requests.

Remarks. We optimize the data layout on SSDs to enable intra- and inter-mini-batch I/O deduplication mechanisms, which eventually mitigate the effect of read amplification and improve I/O efficiency.

4 Implementation

We implement the system prototype of FusionANNS using 22K lines of codes in C++ and CUDA. FusionANNS can be widely deployed in general-purpose servers equipped with an entry-level GPU. Like most ANNS systems [14, 15, 22, 41], we use each CPU thread to handle an individual query.

Contention-free GPU Memory Management. FusionANNS implements a GPU memory manager specifically for concurrent ANNS queries. During system initialization, we first load compressed vectors into GPU's HBM, and use the remaining space as a memory pool. Then, we divide the memory pool into several independent blocks, each of which is assigned to a single query as working memory. Once a query is finished, its block can be assigned to other pending queries. This approach can avoid frequent memory allocations and lock contention between queries, improving the system performance.

Efficient GPU Kernels. For each vector, we allocate multiple GPU threads according to their dimensions to calculate distances in parallel. Moreover, we design a kernel to support parallel deduplication of vector-IDs using a hash algorithm. For a list of candidate Vector-IDs, we allocate a GPU thread for each vector-ID and use a spinlock to ensure that only one thread can access and update a hash table entry at a time.

This approach can fully exploit GPU's high parallelism to accelerate deduplication.

5 Evaluation

Our experiments are conducted on a server equipped with two Intel Xeon CPUs with 2.2 GHz 64 cores, 1 TB main memory, an entry-level NVIDIA V100 GPU with 32 GB HBM, and a Samsung 990Pro SSD with 2 TB storage capacity. For FusionANNS and other SSD-based ANNS solutions, we only use 64 GB main memory to accommodate the in-memory index and intermediate results.

Benchmarks. In our experiments, we use three standard billion-scale datasets [42] that are widely used by previous studies [14, 15, 22], as illustrated in Table 1. Each benchmark simulates workloads using a set of query vectors.

Compared Solutions. We compare FusionANNS with three representative ANNS solutions designed for billion-scale datasets, including two SSD-based solutions and a GPU-accelerated in-memory solution. We do not evaluate accelerator-based solutions using IVFPQ [32] because its low accuracy can not meet the requirement of real-world applications, as reported in many previous studies [15, 22, 41, 43].

- **SPANN** [15] is the state-of-the-art `SSD-based ANNS solution using the IVF index`. It is designed particularly for low latency.
- **DiskANN** [22] is a `SSD-based ANNS solution using the graph index`. It achieves high throughput, but suffers from extremely high latency.
- **RUMMY** [9] is a state-of-the-art `GPU-accelerated in-memory ANNS solution using the IVF index`. It stores all vectors and their indices entirely in host memory. We extend RUMMY to support high-accuracy queries by adopting an advanced IVF index [15], without causing any performance degradation.

Performance Metrics. We use *query per second* (QPS) and average latency to evaluate the performance of various ANNS systems. Like previous studies [14, 19, 44], unless specified otherwise, the query accuracy is evaluated by Recall@10. To achieve a given accuracy level such as Recall@10=0.9, we can adjust two parameters in FusionANNS for different datasets, i.e., the number of top-m nearest posting lists retrieved from the graph index, and the top-n candidate vectors requiring a re-ranking process.

5.1 Performance

Besides the native SPANN and DiskANN, we also compare FusionANNS with GPU-accelerated SPANN and DiskANN (i.e, SPANN-GPU and DiskANN-GPU) which exploit GPU to accelerate distance calculations. We measure the QPS and latency of different ANNS systems under the same constraint of query accuracy. For all experiments, we gradually increase the number of threads for concurrent queries till these systems achieve the peak QPS while still guaranteeing low latency (The latency of multi-threads increases less than 50% relative to the latency of a single thread).

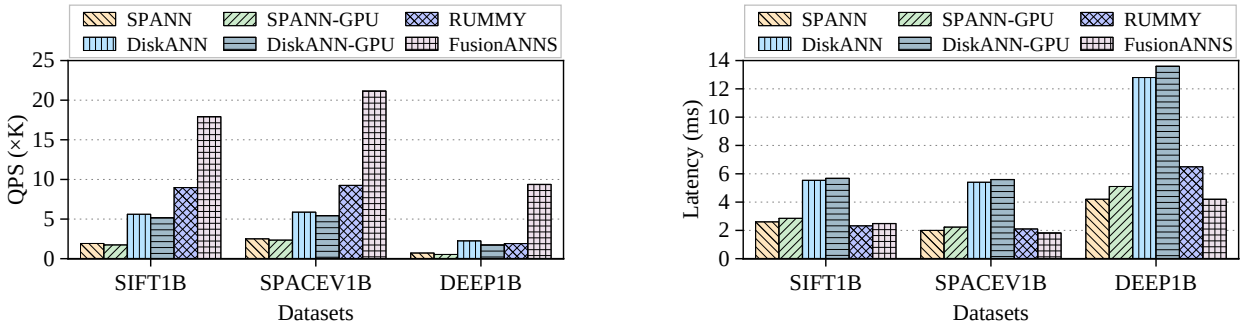
Table 1: Datasets (one billion items)

Dataset	Dimension	Raw Data Size	Data Type	Domain
SIFT1B	128	119 GB	uint8	Image
SPACEV1B	100	93 GB	int8	Web Search
DEEP1B	96	358 GB	float32	Image

Performance under Different Datasets. We compare FusionANNS with other ANNS systems using three datasets. Figure 9a and Figure 9b show QPS and latency, respectively, under an accuracy level of Recall@10=90%. Compared with SSD-based SPANN and DiskANN, FusionANNS can significantly improve QPS by 9.4-13.1× and 3.2-4.3×, respectively. SPANN-GPU and DiskANN-GPU show even lower performance than the native SPANN and DiskANN since the cost of extensive data movement offsets the benefit of GPU acceleration. Although SPANN shows very low latency, its throughput is rather low compared with other ANNS systems. In contrast, FusionANNS achieves low latency comparable to SPANN, but significantly improves the throughput. These results demonstrate that FusionANNS achieves both high throughput and low latency for these billion-scale datasets. Such improvement mainly stems from the multi-tiered index enabled CPU/GPU collaborative filtering and re-ranking techniques. Because FusionANNS can avoid extensive data swapping between CPUs and the GPU, and can also mitigate unnecessary I/O operations on SSDs, it eliminates the I/O performance bottleneck due to limited PCIe bandwidth.

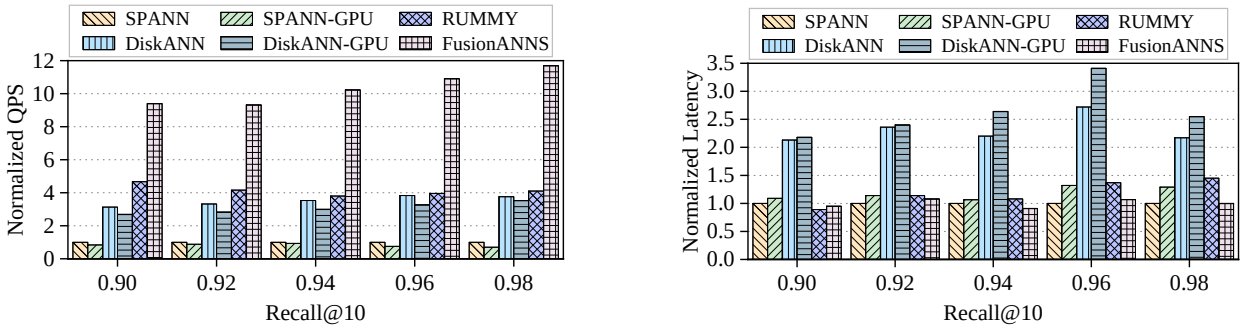
Compared with the GPU-accelerated in-memory solution-RUMMY, FusionANNS improves the QPS by 2-4.9× while remaining low latency for different datasets. Notably, RUMMY exhibits much lower performance for the DEEP1B dataset compared with other datasets. The reason is that the data transfer for a larger dataset from main memory to GPU's HBM consumes more PCIe bandwidth, making the bandwidth bottleneck between CPUs and GPU more pronounced. In contrast, FusionANNS only needs to transfer lightweight vector-IDs rather than the vectors' content between CPUs and the GPU. Thus, FusionANNS achieves significant performance improvement relative to RUMMY.

Performance under Different Accuracy Levels. To evaluate the performance of FusionANNS under different accuracy levels, we change the Recall@10 from 90% to 98%. Figure 10a and Figure 10b show QPS and latency under different levels of accuracy using the SIFT1B dataset, respectively. All results are normalized to SPANN. FusionANNS achieves about 9.4-11.7 × and 3.2 × QPS improvement compared with



(a) Throughput (b) Latency

Figure 9: Throughput and latency of various ANNS systems using different datasets, under Recall@10=0.9



(a) Recall vs. QPS (b) Recall vs. latency

Figure 10: Normalized throughput and latency of different ANNS systems using SIFT1B, under different accuracy levels

SPANN and DiskANN, respectively. FusionANNS achieves more QPS improvement with the increase of query accuracy compared with SPANN, and even offers much lower latency than the in-memory RUMMY under the constraint of higher accuracy. The root cause is that the CPU/GPU corroborative filtering mechanism can effectively eliminate data swapping between main memory and GPU's HBM, an thus can extend the search space to meet a higher accuracy level while still remaining high performance. In contrast, other ANNS solutions cause more I/O operations and distance calculations when the search space becomes larger.

5.2 Scalability

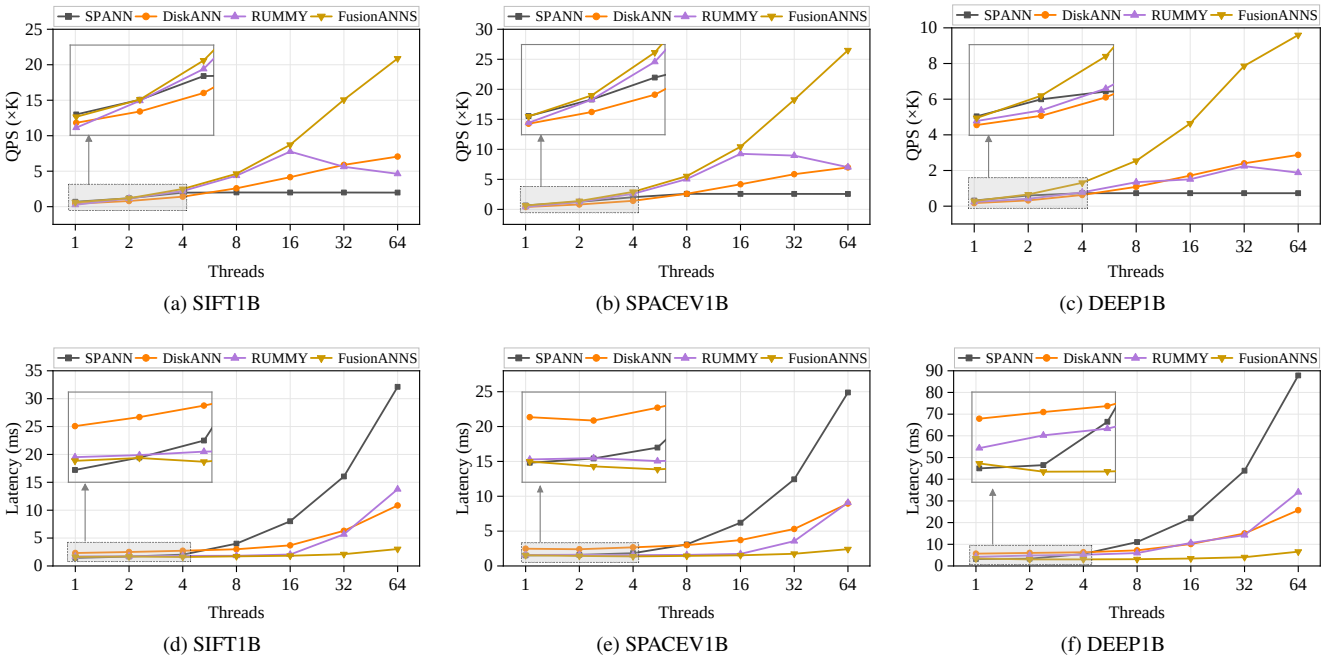
The throughput of ANNS systems is highly correlated to the number of CPU threads. To evaluate the performance scalability of different ANNS systems, we increase the number of threads exponentially by a factor of 2.

Figure 11 shows the QPS and latency of different ANNS systems using different numbers of threads. FusionANNS show a significant growth in QPS when the number of threads increases from 1 to 64. For all ANNS systems, the QPS is almost the same when only one thread is used. However, when the number of threads increases to 64, FusionANNS significantly improves QPS by up to 13.2 ×, 3.8 ×, and 5.1 × while remaining low latency, compared with SPANN, DiskANN, and RUMMY, respectively. SPANN achieves its peak QPS using only 4 threads, and its latency increases significantly with more threads. Notably, for SIFT1B and SPACEV1B datasets, the QPS of RUMMY peaks at 16 threads, and then decreases distinctly. Meanwhile, the latency of RUMMY also significantly increases when the number of threads becomes larger than 16. This reason of such limited scalability is that more concurrent queries lead to a large amount of data transmission between CPUs and GPU, which lead to significant PCIe bandwidth contention. Due to the bigger vector size of the DEEP1B dataset, RUMMY suffers from more severe bandwidth contention, and thus its QPS is even lower than DiskANN, as shown in Figure 11f. FusionANNS shows much better scalability than others even using limited memory resource because it eliminates the data swapping between CPUs and the GPU, and also significantly improves the I/O efficiency on SSDs.

5.3 Effectiveness of Individual Techniques

In this subsection, we evaluate the effectiveness of individual techniques in FusionANNS using SIFT1B. We first use our multi-tiered indexing technique to conduct a CPU-based variant (i.e., MI(CPU)) in which we replace the GPU with host CPUs to process compressed vectors. Then, we incrementally add other techniques of FusionANNS to evaluate their impacts on the performance and the amount of I/Os.

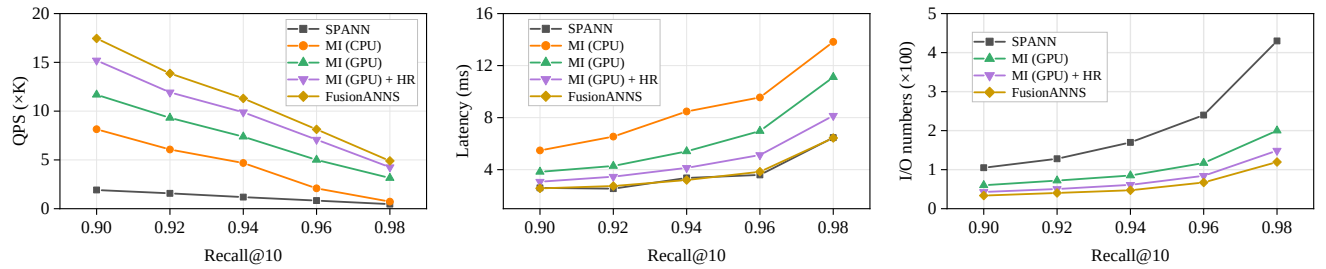
As shown in Figure 12a and 12b, the multi-tiered indexing with CPUs, i.e., MI(CPU), achieves 1.5-4.2 × higher QPS



(a) SIFT1B (b) SPACEV1B (c) DEEP1B

(d) SIFT1B (e) SPACEV1B (f) DEEP1B

Figure 11: The throughput and latency of different ANNS systems vary with the number of threads.



(a) Throughput (b) Latency (c) I/O Reduction

Figure 12: Performance improvement and I/O reduction introduced by different technologies. MI (CPU) and MI (GPU) denote our multi-tiered indexing approach using CPUs and GPU acceleration, respectively. HR denotes heuristic re-ranking. FusionANNS exploits all three techniques, including multi-tiered indexing, heuristic re-ranking, and redundant-aware I/O deduplication.

compared with SPANN, but suffers from high latency. However, the multi-tiered indexing with GPU, i.e., MI(GPU), can significantly reduce latency compared with MI(CPU). It also improves the QPS by 5.9-6.8 × compared with SPANN. This is because CPUs' DRAM bandwidth is very limited, while the high-bandwidth and high-parallelism capabilities of GPU's HBM allow MI(GPU) to process PQ distance calculations more efficiently. Moreover, the multi-tiered indexing enables CPUs only transfer vector-IDs to the GPU, alleviating the performance bottleneck of PCIe bandwidth. Based on the multi-tiered indexing, both *heuristic re-ranking* (HR) and redundancy-aware I/O deduplication can further reduce latency and improve QPS by up to 39% and 17%, respectively.

Figure 12c shows the average I/O numbers aroused by each query. The number of I/O requests in SPANN increases significantly when a higher accuracy level should be guaranteed. The multi-tiered indexing technique can reduce I/O numbers by 3.2-3.8 × compared with SPANN. The heuristic re-ranking and the redundancy-aware I/O deduplication can further reduce I/O numbers by up to 30% and 23%, respectively. In addition, each I/O operation launched by SPANN usually involves multiple SSD pages, while other ANNS systems only involve one page. FusionANNS not only significantly reduces the number of I/Os per query, but also reduces the I/O size, thus achieving substantial performance improvement.

5.4 Cost and Memory Efficiency

We compare FusionANNS with other ANNS systems in terms of cost and memory efficiency. We use the QPS/$ and QPS/GB to evaluate the cost and memory efficiency, respectively. The system cost includes the server cost (around $5000, including CPUs and server chassis), the memory cost (around $10/GB), the storage cost ($400 for a 2TB Samsung SSD),

Table 2: Cost Efficiency (QPS/$)

Datasets	SPANN	DiskANN	RUMMY	**FusionANNS**
SIFT1B	0.32	1.01	0.88	**1.98**
SPACEV1B	0.41	1.12	1.40	**2.35**
DEEP1B	0.12	0.41	0.15	**1.01**

Table 3: Memory Efficiency (QPS/GB)

Datasets	SPANN	DiskANN	RUMMY	**FusionANNS**
SIFT1B	29.98	94.81	47.75	**280.23**
SPACEV1B	39.12	96.4	88.4	**330.79**
DEEP1B	11.17	38.1	4.51	**146.16**

and the GPU cost (around $3000 for Nvidia V100). These prices are referenced from Amazon. Although it may be not fair to compare DiskANN with other ANNS systems because DiskANN improves QPS at the expense of high latency, we still report the results of DiskANN for reference.

As shown in Table 2, FusionANNS achieves 5.67-8.78×, 2-2.5×, and 2.25-6.82× improvement in QPS/$ compared to SPANN, DiskANN, and RUMMY, respectively. This is mainly due to the significant performance of FusionANNS. Also, FusionANNS achieves higher memory efficiency, as shown in Table 3. Specifically, for the large-volume dataset DEEP1B, FusionANNS improves memory efficiency by 13.1 ×, 3.8 ×, and 32.4 × compared with SPANN, DiskANN, and RUMMY, respectively. FusionANNS dramatically improves the cost and memory efficiency because our multi-level indexing technology can significantly improve performance and reduce memory footprint.

6 Related Work

In-memory ANNS Solutions. ANNS has been extensively studied for decades, mainly focusing on in-memory indexing techniques [16, 17, 31, 33, 36]. *Hierarchical Navigable Small World* (HNSW) [17] maintains a multi-layered navigable graph in memory to achieve fast ANNS. *Space Partition Tree and Graph* (SPTAG) [36] exploits a relative neighborhood graph and space partition trees to find several seeds for graph traversal acceleration. However, existing in-memory ANNS algorithms require a large amount of memory resource to maintain raw vectors and their indices. The huge memory requirement significantly increases the total cost of ownership, impeding the ANNS scaling to large-scale datasets. FusionANNS leverages SSDs to support large-scale datasets, and achieves both high performance and cost efficiency through multi-tiered indexing empowered system optimizations.

SSD-based ANNS Solutions. A number of recent studies have proposed hierarchical indices to reduce memory footprint for billion-scale datasets, such as DiskANN [22], SPANN [15], Starling [41], BBANN [45], and GRIP [46]. These proposals exploit the characteristics of modern SSDs to optimize vector indexing. SmartANNS [19] explores hierarchical indexing technique for SmartSSD-based ANNS. It leverages the internal PCIe bandwidth of multiple SmartSSDs to alleviate I/O bottlenecks, achieving near-linear scalability for billion-scale ANNS. However, the throughput of SSD-based ANNS system is quite limited due to severe I/O contention among concurrent queries, making them hard to meet the high-throughput requirement. FusionANNS significantly reduces I/Os per query through a novel multi-tiered index structure and CPU/GPU collaborative searching, and thus achieves both high throughput and low latency.

Accelerator-based ANNS Solutions. A few recent studies [24–26, 47] exploit GPUs to accelerate the graph traversal involved in graph-based ANNS. Moreover, a number of works have exploited PQ techniques [32, 48], GPUs [20, 21], and FPGAs [49–51] to accelerate IVF-based ANNS. However, most these approaches can only support small-scale datasets due to the limited memory capacity in GPUs and FPGAs [24–26, 47, 49], or can not guarantee high accuracy [49–51]. Although state-of-the-art RUMMY [9] expands GPU memory with host memory and proposes a reordered pipelining technique to support billion-scale datasets, its performance is still limited due to extensive data transmission between CPUs and GPU. Through a careful collaboration of hierarchical indexing, PQ, and GPU acceleration techniques, FusionANNS can eliminate data swapping between GPUs and CPUs, and thus achieves high throughput, low latency, and low cost while still guaranteeing high accuracy.

7 Conclusion

In this paper, we present FusionANNS, a "CPU + GPU" collaborative processing architecture for billion-scale ANNS. FusionANNS exploits GPU/CPU collaborative filtering and reranking mechanisms to significantly improve the performance and cost efficiency of ANNS while still guaranteeing high accuracy. Compared with the state-of-the-art SSD-based ANNS solution–SPANN, FusionANNS significantly improves the throughput of concurrent queries while still remaining low latency. Moreover, FusionANNS also achieves higher throughput and cost efficiency than the GPU-accelerated in-memory ANNS solution–RUMMY.

Acknowledgments

We sincerely thank our shepherd Aishwarya Ganesan and anonymous reviewers for helping us improve our paper significantly. This work is supported jointly by National Key Research and Development Program of China under grant No. 2022YFB4500303, National Natural Science Foundation of China (NSFC) grants No. 62332011, 62302178, Natural Science Foundation of Hubei Province grant No.2021CFA037, and Huawei grant No.YBN2021035018A7.

References

[1] Yukihiro Tagami. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-Label Classification. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, page 455–464, 2017.

[2] 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. Uni-retriever: Towards learning the unified embedding based retriever in Bing sponsored search. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 4493–4501, 2022.

[3] Yanhao Zhang, Pan Pan, Yun Zheng, Kang Zhao, Yingya Zhang, Xiaofeng Ren, and Rong Jin. Visual search at Alibaba. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 993–1001, 2018.

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

[5] Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. Chatlaw: Open-source legal large language model with integrated external knowledge bases. arxiv preprint arxiv:2306.16092, 2023.

[6] Cheonsu Jeong. A study on the implementation of generative AI services using an enterprise data-based LLM application architecture. arxiv preprint arxiv:2309.01105, 2023.

[7] Cheng Wen, Xianghui Sun, Shuaijiang Zhao, Xiaoquan Fang, Liangyu Chen, and Wei Zou. Chathome: Development and evaluation of a domain-specific language model for home renovation. arxiv preprint arxiv:2307.15290, 2023.

[8] Xuanyu Zhang, Qing Yang, and Dongliang Xu. Xuanyuan 2.0: A large chinese financial chat model with hundreds of billions parameters. arxiv preprint arxiv:2305.12002, 2023.

[9] Zili Zhang, Fangyue Liu, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast vector query processing for large datasets beyond GPU memory with reordered pipelining. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 23–40, 2024.

[10] Karthik V, Saim Khan, Somesh Singh, Harsha Vardhan Simhadri, and Jyothi Vedurada. Bang: Billion-scale approximate nearest neighbor search using a single GPU. arxiv preprint arxiv: 2401.11324, 2024.

[11] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, and Yu Wang. 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)*, page 283–296, 2023.

[12] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. Parlayann: Scalable and deterministic parallel graph-based approximate nearest neighbor search algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, page 270–285, 2024.

[13] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. iQAN: Fast and accurate vector search with efficient intra-query parallelism on multi-core architectures. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, page 313–328, 2023.

[14] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, and Mao Yang. SPFresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, page 545–561, 2023.

[15] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: highly-efficient billion-scale approximate nearest neighborhood search. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 5199–5212, 2021.

[16] Artem Babenko and Victor Lempitsky. The inverted multi-index. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3069–3076, 2012.

[17] Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[18] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation. arxiv preprint arxiv:2404.12457, 2024.

[19] Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. Scalable billion-point approximate nearest neighbor search using SmartSSDs. In *Proceedings of the 2024 USENIX Annual Technical Conference (ATC)*, pages 1135–1150, 2024.

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

[21] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. JUNO: Optimizing high-dimensional approximate nearest neighbour search with sparsity-aware algorithm and ray-tracing core mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, page 549–565, 2024.

[22] Suhas Jayaram Subramanya, Fnu Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. arxiv preprint arxiv:2404.12457, 2024.

[24] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P. A. Lensch. GGNN: graph-based GPU nearest neighbor search. *IEEE Trans. Big Data*, 9(1):267–279, 2023.

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

[26] Weijie Zhao, Shulong Tan, and Ping Li. SONG: approximate nearest neighbor search on GPU. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*, pages 1033–1044, 2020.

[27] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, page 604–613, 1998.

[28] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. iDEC: Indexable distance estimating codes for approximate nearest neighbor search. *Proc. VLDB Endow.*, 13(9):1483–1497, 2020.

[29] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.*, 9(1):1–12, 2015.

[30] Chanop Silpa-Anan and Richard I. Hartley. Optimised kd-trees for fast image descriptor matching. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[31] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the 15th European Conference Computer Vision (ECCV)*, page 209–224, 2018.

[32] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[33] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.

[34] John Hartigan and Ma-Li Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C*, 28(1):100–108, 1979.

[35] Rongxin Cheng, Yifan Peng, Xingda Wei, Hongrui Xie, Rong Chen, Sijie Shen, and Haibo Chen. Characterizing the dilemma of performance and index size in billion-scale vector search and breaking it with second-tier memory. arxiv preprint arxiv:2404.12457, 2024.

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

[37] LanceDB. https://lancedb.github.io/lancedb/.

[38] Jörg Thalheim, Harshavardhan Unnibhavi, Christian Priebe, Pramod Bhatotia, and Peter Pietzuch. rkt-io: a direct I/O stack for shielded execution. In *Proceedings of the 16th European Conference on Computer Systems (EuroSys)*, page 490–506, 2021.

[39] Yingjin Qian, Marc-André Vef, Patrick Farrell, Andreas Dilger, Xi Li, Shuichi Ihara, Yinjin Fu, Wei Xue, and Andre Brinkmann. Combining buffered I/O and direct I/O in distributed file systems. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FAST)*, pages 17–33, 2024.

[40] Giuseppe Ottaviano and Rossano Venturini. Partitioned Elias-Fano indexes. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR)*, page 273–282, 2014.

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

[42] BIG ANN-Benchmarks. https://big-ann-benchmarks.com/neurips21.html.

[43] Jie Ren, Minjia Zhang, and Dong Li. HM-ANN: efficient billion-point nearest neighbor search on heterogeneous memory. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 6–12, 2020.

[44] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 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 (ATC)*, pages 585–600, 2023.

[45] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189, 2022.

[46] Minjia Zhang and Yuxiong He. GRIP: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, page 1673–1682, 2019.

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

[48] Zhibin Pan, Liangzhuang Wang, Yang Wang, and Yuchen Liu. Product quantization with dual codebooks for approximate nearest neighbor search. *Neurocomputing*, 401:59–68, 2020.

[49] Wenqi Jiang, Shigang Li, Yu Zhu, Johannes De Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli, Shuai Zhang, Theodoros Rekatsinas, Torsten Hoefler, and Gustavo Alonso. Co-design hardware and algorithm for vector search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–16, 2023.

[50] Yejin Lee, Hyunji Choi, Sunhong Min, Hyunseung Lee, Sangwon Beak, Dawoon Jeong, Jae W. Lee, and Tae Jun Ham. ANNA: specialized architecture for approximate nearest neighbor search. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 169–183, 2022.

[51] Jialiang Zhang, Soroosh Khoram, and Jing Li. Efficient large-scale approximate nearest neighbor search on opencl FPGA. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4924–4932, 2018.