FlashANNS: GPU-Driven Asynchronous I/O Pipelining for Eliminating Storage-Compute Bottlenecks in Billion-Scale Similarity Search

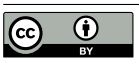
YANG XIAO, Zhejiang University, China
MO SUN, Zhejiang University, China
ZIYU SONG, Zhejiang University, China
BING TIAN, Huazhong University of Science and Technology, China
JIE SUN, Zhejiang University, China
JIE ZHANG, Ningbo Global Innovation Center, Zhejiang University, China and Zhejiang University, China
ZEKE WANG, Zhejiang University, China
ZONGHUI WANG, Zhejiang University, China
WENZHI CHEN, Zhejiang University, China
FEI WU, Zhejiang University, China

Approximate Nearest Neighbor Search (ANNS) enables efficient similarity retrieval in high-dimensional vector spaces, and becomes a fundamental component of upper-layer workloads ranging from recommendation systems to retrieval-augmented generation (RAG). Modern ANNS systems integrate SSDs to support terabyte-scale vector datasets, primarily employing cluster-indexing and graph-indexing. However, cluster-indexing ANNS systems suffer from suboptimal query throughput because of the coarse-grained vector indexing, while graph-indexing systems suffer from suboptimal performance due to two inherent limitations: 1) failing to overlap SSD accesses with distance computation processes and 2) poor I/O performance due to long tail latency.

To address these challenges, we present FlashANNS, a GPU-accelerated out-of-core graph-based ANNS system through I/O-compute overlapping. Our core insight lies in the careful orchestration of I/O and computation through three key innovations: 1) Dependency-relaxed asynchronous pipeline with rigorous theoretical convergence guarantee: FlashANNS decouples I/O-computation dependencies to fully overlap between GPU distance calculations and SSD data transfers; 2) Query-grained concurrent SSD access: FlashANNS implement a lock-free I/O stack with query-grained concurrency control, to avoid I/O performance degradation due to long tail latency; and 3) Computation-I/O balanced graph degree selection, which ensures optimal balance between computational load and storage access latency across different hardware characteristics, different datasets, and different query requirements.

We implement FlashANNS and compare it with three state-of-the-art out-of-core ANNS systems (SPANN, DiskANN, and FusionANNS). Experimental results demonstrate that at the same ≥95% recall@10 accuracy,

Authors' Contact Information: Yang Xiao, Zhejiang University, Hangzhou, China, 12221061@zju.edu.cn; Mo Sun, Zhejiang University, Hangzhou, China, sunmo@zju.edu.cn; Ziyu Song, Zhejiang University, Hangzhou, China, songziyu@zju.edu.cn; Bing Tian, Huazhong University of Science and Technology, Wuhan, China, tbing@hust.edu.cn; Jie Sun, Zhejiang University, Hangzhou, China, jiesun@zju.edu.cn; Jie Zhang, Ningbo Global Innovation Center, Zhejiang University, Ningbo, Zhejiang, China and Zhejiang University, Hangzhou, China, carlzhang4@zju.edu.cn; Zeke Wang, Zhejiang University, Hangzhou, China, wangzeke@zju.edu.cn; Zonghui Wang, Zhejiang University, Hangzhou, China, zhwang@zju.edu.cn; Wenzhi Chen, Zhejiang University, Hangzhou, China, chenwz@zju.edu.cn; Fei Wu, Zhejiang University, Hangzhou, China, wufei@zju.edu.cn.



ACM 2836-6573/2026/2-ART38
https://doi.org/10.1145/3786652

our method achieves 2.7–5.9× higher query throughput compared to existing SOTA methods with a single SSD, and further achieves 3.9–12.2× query throughput improvement in the multi-SSD configurations.

CCS Concepts: • **Information systems → Information retrieval query processing**; *Storage management*.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search (ANNS), Large-scale similarity search, High-dimensional vector retrieval, Parallel query execution

ACM Reference Format:
Yang Xiao, Mo Sun, Ziyu Song, Bing Tian, Jie Sun, Jie Zhang, Zeke Wang, Zonghui Wang, Wenzhi Chen, and Fei Wu. 2026. FlashANNS: GPU-Driven Asynchronous I/O Pipelining for Eliminating Storage-Compute Bottlenecks in Billion-Scale Similarity Search. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 38 (February 2026), 27 pages. https://doi.org/10.1145/3786652

1 Introduction

Approximate Nearest Neighbor Search (ANNS) refers to a set of methods for finding the top-k vectors most similar to a given query vector in a high-dimensional vector dataset. Compared with exact search, ANNS sacrifices a little precision for much less retrieval time. ANNS is widely applied in various domains, including information retrieval [8, 24, 28, 39, 62], recommendation systems [13, 49, 56], and large language models [14, 30, 33, 37, 59]. Especially, the rise of generative AI and large-scale recommendation systems has driven the demand for billion-scale ANNS, with modern datasets (often exceeding 1000M vectors) and their indices (up to 6× dataset size) overwhelming traditional in-memory frameworks due to prohibitive memory and computation requirements.

A straightforward scale-out solution is to employ a distributed in-memory solution [43, 71] that utilizes the host memory of multiple servers for index storage. Vearch [43] partitions a size-N index into n shards of size N/n, each shard is stored in the DRAM of a server. During query execution, all nodes process the same query concurrently on their local shards, and each node computes its local top-K results from its assigned shard. Finally, global results are merged by sorting and truncating the $n \times K$ local results to yield the final K-nearest neighbors. However, bringing $n\times$ computation resource can not increase query throughput by $n\times$, mainly because the query time does not scale linearly with the dataset shard size. As shown in Figure 1, halving the dataset size can only increase query throughput by 10.8%-19.5%.

Considering the suboptimal performance of scale-out solutions, recent ANNS systems [11, 31, 52, 69, 77] are equipped with Solid State Drives (SSDs) to fit such huge datasets/indices (up to tens of Terabytes), to further extend single node capacity. According to the kind of indexing method used, these systems can be categorized into cluster-based and graph-based systems. We identify that both of these systems suffer from low system resource utilization.

Cluster-Based Indexing. Cluster-indexing methods such as SPANN [11] partition the dataset into cluster-based inverted lists, storing original vectors on disk while maintaining centroid indices in memory. However, its coarse-grained clustering leads to a large candidate search space, requiring extensive vector comparisons in high-precision scenarios.

FusionANNS [69] mitigates this issue via GPU-accelerated pre-filtering to reduce I/O overhead by selectively fetching vectors from SSDs. Despite this improvement, FusionANNS still incurs high SSD accesses due to the inherent limitations of coarse-grained clustering. Furthermore, its computational pipeline does not fully utilize the GPU's processing capability due to high overhead from the CPU-managed task synchronization.

Graph-Based Indexing. Fine-grained graph indexing such as DiskANN [31] can achieve the target accuracy with a smaller candidate size, thereby reducing the number of SSD fetches and the overall I/O cost. However, it increases per-fetch computational work (e.g., distance evaluations

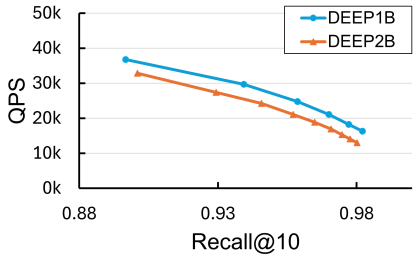


Fig. 1. Comparison of query QPS performance between dataset DEEP1B and DEEP2B

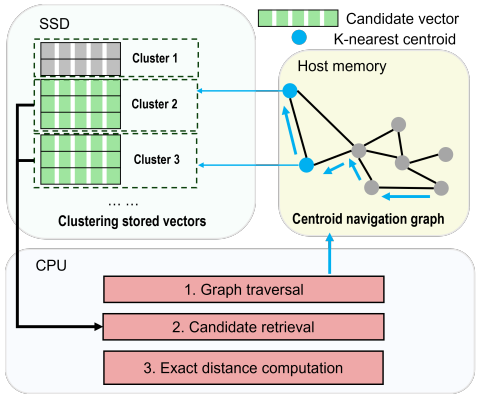
and neighbor management). As a result, a larger share of the end-to-end latency is attributable to computation, indicating the potential for compute-side optimization.

While graph-based indexing reduces I/O, it increases compute intensity, making computation a larger share of latency. This motivates a GPU-offloading direction for compute-side optimization. However, directly integrating GPUs introduces three key challenges: (1) serialized or coarse-grained execution that fails to fully exploit GPU compute capacity; (2) fine-grained synchronization that amplifies I/O latency; and (3) small random SSD accesses that are IOPS-bound, leaving SSD bandwidth underutilized.

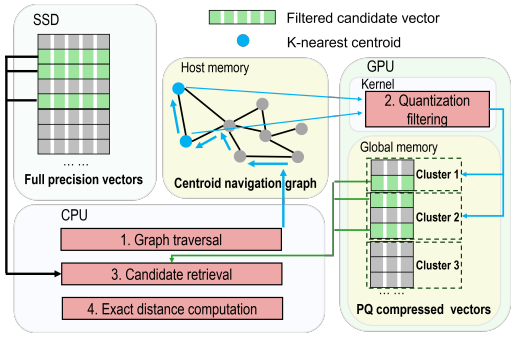
To this end, we present FlashANNS, a GPU-accelerated, SSD-resident graph indexing framework that balances computation and I/O for billion-scale similarity search. Our approach centers on three key designs:

- **Dependency-Relaxed Asynchronous I/O Pipeline.** To address the low GPU/SSD utilization problem caused by the serialized execution of computation and I/O operations, we propose a dependency-relaxed asynchronous I/O pipeline, and provide a rigorous theoretical convergence guarantee to ensure that FlashANNS can achieve the same recall.
- **Query-Grained Concurrent SSD Access.** To address the prolonged I/O latency due to synchronization overhead in the kernel-grained GPU I/O stack, we propose a query-grained concurrent SSD access architecture optimized for ANNS workloads, eliminating the GPU kernel-grained global synchronization inherent to conventional CAM-based I/O stacks. Under this architecture, each query-issued SSD access can complete and resume execution independently within the warp without kernel-wide global stalls.
- **Computation-I/O Balanced Graph Degree Selection.** To address the I/O amplification due to SSD page realignment, we propose a hardware-aware graph degree selection mechanism. This approach quantifies SSD I/O bandwidth and GPU computational capacity through sampling-based profiling, enabling adaptive selection of graph degree parameters in the index structure. This maintains computation-I/O equilibrium throughout pipeline operations.

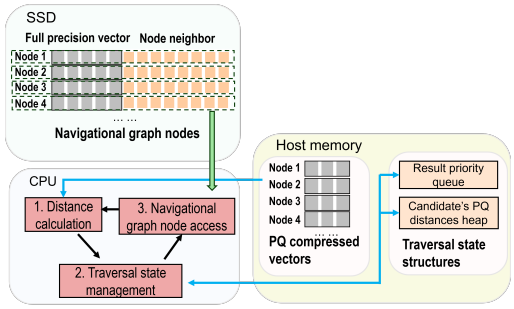
We implement FlashANNS and evaluate it on widely adopted billion-scale vector datasets (SIFT1B, SPACEV1B, DEEP1B), benchmarking against state-of-the-art out-of-core ANNS systems (SPANN, DiskANN) and a GPU-accelerated out-of-core baseline (FusionANNS). Experimental results show that at ≥95% recall@10 accuracy, FlashANNS achieves 2.7–5.9× higher QPS than existing SOTA methods with a single SSD configuration, and scales to up to 12.2× QPS improvements in multi-SSD setups. We will make FlashANNS open-sourced at GitHub to benefit our community once this paper gets accepted.



(a) Cluster-Based indexing



(b) Filtered Cluster-Based indexing



(c) CPU-Based graph indexing

Fig. 2. Overall comparison of indexing architectures

2 Background

The Approximate Nearest Neighbor Search (ANNS) has emerged as a fundamental operation in modern data processing systems, enabling efficient similarity retrieval in high-dimensional vector spaces. As a fundamental component of upper-layer workloads ranging from recommendation systems to retrieval-augmented generation (RAG), ANNS evolves rapidly as the scale of these workloads becomes larger and larger. Vector indexing serves as a critical ANNS component, organizing vector data and determining search methodologies. Substantial index storage overheads consume

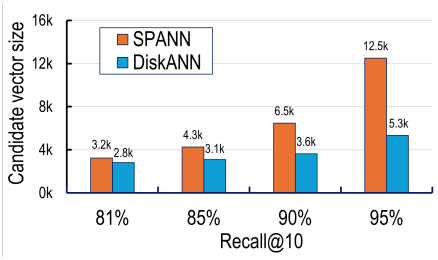


Fig. 3. Candidate vector size during searching progress of cluster-indexing (SPANN) and graph-indexing (DiskANN) under different recall accuracy

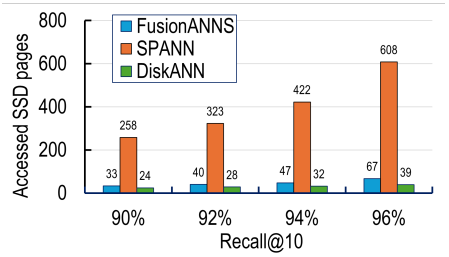


Fig. 4. SSD page read requirements of FussionANNS, SPANN and DiskANN under different recall accuracy

up to 86% of the query execution storage footprint. Modern ANNS systems [11, 31, 69] integrate SSDs to support terabyte-scale vector datasets required by contemporary applications.

2.1 Cluster-Based Vector Indexing

Cluster-based vector indexing is widely used due to its low query latency. These systems mainly maintain two data structures: 1) SSD-resident clustered vectors and 2) an in-memory navigation graph of cluster centroids. During the index construction, the system scans the entire vector dataset and partitions vectors into several clusters. The vectors are stored in the SSD by clusters, while the centroid of each cluster forms a graph and is stored in the CPU memory. Figure 2a demonstrates the three-phase search workflow of cluster-based vector indexing.

- **1. Graph Traversal Phase.** Within the in-memory centroid navigation graph, the search process traverses the graph to identify the K-nearest centroids to the query vector.
- **2. Candidate Retrieval Phase.** The retrieval phase fetches clusters corresponding to the selected K centroids from SSD-resident clustered vectors.
- **3. Distance Calculation Phase.** The search process computes precise distances between all vectors in retrieved clusters and the query vector, then ranks vectors by distance to select the final result.

However, cluster-indexing ANNS systems suffer from suboptimal query throughput, because these systems use coarse-grained vector indexing, and graph traversal phase (❶)'s computation can be very light-weight, while there would be numerous candidates to be accessed in candidate retrieval phase (❷) and corresponding distance calculation in distance calculation phase (❸).

The fundamental limitation stems from the graph traversal phase (❶)'s assumption that centroids within posting lists adequately represent associated cluster vectors. In practice, this representation proves to be inaccurate, manifesting two distinct deficiencies:

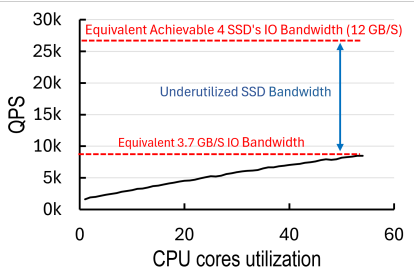


Fig. 5. DiskANN query throughput variation with increasing CPU cores

1. False Inclusion. Centroids proximate to the query may correspond to distant vectors, forcing candidate retrieval phase (❷) and distance calculation phase (❸) to process massive volumes of irrelevant candidates.

2. False Exclusion. During candidate retrieval (❷), the search process may erroneously exclude distant centroids containing neighboring vectors, misclassifying the truly nearest neighbors within them as distant. This initial error leads to their absence in the candidate pool, severely degrading query accuracy.

Figure 3 demonstrates the candidate size disparity between the cluster-indexing implementation (SPANN) and the graph-indexing implementation (DiskANN) on SIFT1B. Under 81%-95% recall@10, to achieve equivalent recall, cluster-indexing implementation requires 1.14-2.34× more candidates than graph-indexing. And this gap is widening with the increasing accuracy. This imposes prohibitive computational and I/O pressure during queries. Consequently, DiskANN achieves 1.23× to 1.88× higher QPS than SPANN.

GPU-Accelerated Cluster-Based Vector Indexing. To mitigate excessive computational and SSD access demands, FusionANNS [69] leverages a GPU and stores quantified vectors in GPU memory. These vectors reduce dimension and are quantified as uint8 values via product quantization (PQ). As Figure 2b demonstrates, compared to the cluster-indexing approach's three-phase search workflow, FusionANNS introduces a GPU-based quantization filtering phase (❷) between graph traversal (❶) and candidate retrieval (❸).

In quantization filtering phase (❷), the GPU calculates quantization distances for vectors within clusters selected during graph traversal (❶) using PQ-coded vectors. It screens vectors with distances below a predetermined threshold, directing subsequent phases to retrieve and process only these filtered candidates from SSD storage for exact distance calculations.

Despite efficiency gains from quantization filtering, FusionANNS can only partially mitigate severe SSD I/O pressures because its filtering stage relies solely on PQ-compressed distances for candidate selection, lacking the fine-grained navigation guidance formed by exact distances. This necessitates additional SSD accesses to obtain exact distances for compensating quantization errors. As shown in Figure 4, FusionANNS still incurs 1.36–1.70× more SSD page accesses than DiskANN when evaluating on the SIFT1B dataset.

2.2 Graph-Based Vector Indexing

To minimize SSD accesses and the corresponding computation, emerging ANNS systems (e.g., DiskANN [31], DiskANN++ [52]) use a fine-grained vector graph instead of a coarse-grained cluster centroid graph. These systems mainly maintain two data structures: 1) in-memory quantified vectors, and 2) SSD-resident adjacency list of the vector graph and full-precision vectors. Figure 2c demonstrates how graph-indexing ANNS systems retrieve results iteratively.

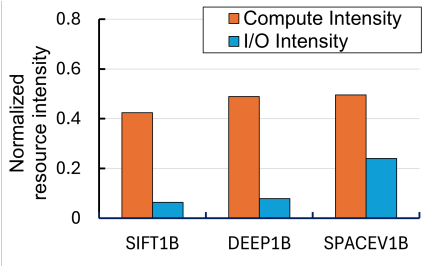


Fig. 6. Normalized I/O and compute demands of graph indexing across datasets (vs. cluster indexing)

- **1. Distance Calculation Phase.** The system computes exact query-to-node distances and calculates PQ distances for neighbors using in-memory quantized vectors.
- **2. Traversal State Management Phase.** The system maintains exact distances in a result heap while managing PQ distances via a min-heap. The next traversal node is dynamically selected from the min-heap by minimal PQ distance.
- **3. Neighbor Access Phase.** The system accesses selected node's full-precision vectors and its neighbor lists from SSDs.

By employing high-precision vector indexing, this approach reduces the candidate size and minimizes distance computations. As shown in Figure 3 and Figure 4, DiskANN achieves significantly smaller candidate size and lower SSD access requirements compared to SPANN and FusionANNS.

3 Motivation

As shown in Figure 6, graph-indexing reduces many more SSD accesses than calculation, which makes the system easily bottlenecked by the computation. Figure 5 shows how query throughput changes with varying CPU core counts. We observe that query throughput scales nearly linearly with core count. In particular, when all 52 cores are used, the consumed SSD throughput is only 3.7 GB/s, which is slightly higher than the I/O bandwidth of a PCIe 4.0x4 SSD (Intel P5510 [63]). When the system scales to multiple SSDs, the SSD I/O throughput is underutilized.

Inspired by FusionANNS, which adopts GPU to address the computation bottleneck of preranking, it is straightforward to consider introducing GPUs to alleviate the computational pressure from distance calculation. The overall design seems straightforward, but faces three severe challenges.

C1: Low GPU Utilization due to Serialized Execution of Computation and I/O. Graph-indexing ANNS algorithms inherently suffer from a fundamental computational dependency: the distance calculation phase requires neighbor vector data stored on storage devices, and the neighbor access phase fetches data depending on preliminary distance results. This cyclic dependency creates unavoidable computation stalls. With GPUs, accelerated distance calculations make severe pipeline imbalance - computation completes quicker, leading to a larger proportion of time waiting for I/O. In our experiments (§ 5.3), under the same recall@10 condition, the QPS of serial execution demonstrates 67.6%–73.1% of our asynchronous execution.

C2: Prolonged I/O Latency due to Synchronization in the Kernel-grained GPU I/O Stack. Kernel-grained GPU I/O stacks optimize I/O throughput via batched SSD accesses, but any single long-tail latency event within a batch propagates delays to all co-batched requests. In the GPU-accelerated ANNS systems, massively increased concurrent SSD access traffic critically intensifies this latency amplification. In our experiments (§ 5.4), under the same recall@10 condition, the QPS of kernel-grained execution demonstrates 55.3%–67.4% of query-grained execution.

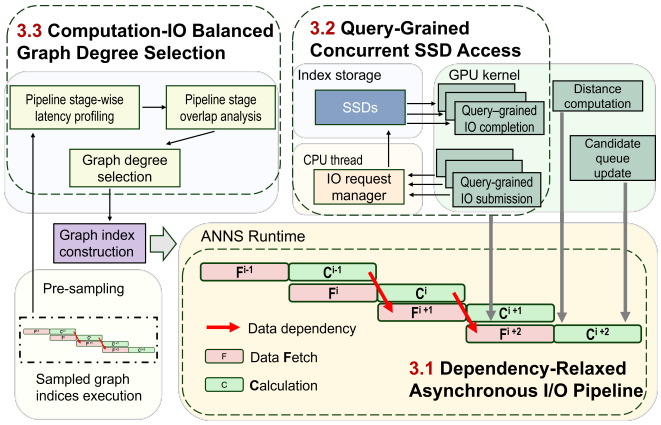


Fig. 7. FlashANNS overview

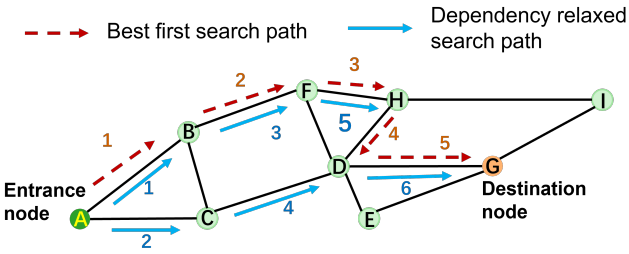


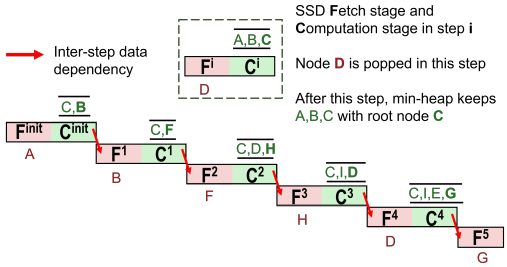
Fig. 8. Exampled search paths of best-first search vs. relaxed dependency search

C3: I/O Amplification due to SSD Page Misalignment. In the graph-indexing ANNS access patterns, SSD reads typically use a 4KB minimum granularity since IOPS remain consistent when the access size is smaller than 4 KB [31]. During graph traversal, each query step accesses one graph node. However, standard graph nodes storing vector data and neighbor indices are often significantly smaller than 4KB, causing severe I/O amplification. For example, a 64-degree graph node occupies merely 384B (9.37% of 4KB), resulting in 90.63% bandwidth waste per access.

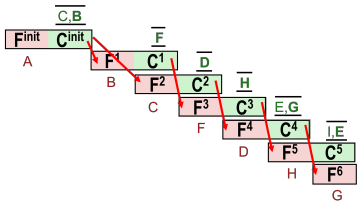
4 Design of FlashANNS

To address these issues, we propose FlashANNS, a GPU-accelerated out-of-core graph-indexing ANNS system.

Figure 7 shows the overall architecture of FlashANNS. FlashANNS consists of three novel designs. 1) Pipelined I/O-compute processing. It parallelizes GPU computation and I/O accesses by loosening data dependency. 2) query-grained GPU-SSD direct I/O stack. It employs query-grained synchronization to reduce tail latency interference during SSD data retrieval. 3) Sampling-based graph degree selector. Using sample indices to characterize pipeline performance across varying graph degrees under current hardware profiles, thereby selecting optimal degrees that maximize pipeline overlap for distinct SSD quantities.



(a) Best-first search pipeline



(b) Relaxed dependency search pipeline

Fig. 9. Execution pipeline of best-first search and relaxed dependency search

4.1 Dependency-Relaxed Asynchronous I/O Pipeline

To address C1, we propose a dependency-relaxed I/O pipeline. Instead of enforcing strict step-by-step dependencies, our design permits SSD accesses to proceed before the preceding compute stage has finished. In this section, we first identify where strict sequential dependencies arise and then present a staleness-aware pipeline that relaxes them. Finally, we establish convergence guarantees for the dependency-relaxed search and provide an explanation supporting the choice of staleness step.

4.1.1 Data-Dependency in Traditional Best-First Search. As the search path shown in Figure 8, traditional best-first search in ANNS graph indices begins from an entrance node and iteratively visits neighboring nodes. In this process, we define one *search step* as a pop–expand iteration: at step i, the algorithm pops the current closest node v_i from the candidate min-heap, issues an SSD read to fetch all neighbors of v_i, computes their distances to the query, and inserts them into the candidate min-heap.

Figure 9a shows that the algorithm computes neighboring nodes' distances to the query and maintains a candidate min-heap that is ranked by nodes' distances to the query vector. Each "min-heap" in figure 9 is a snapshot of the candidate min-heap at the *end* of a search step, and the node at the head of the queue is the heap root that will be popped and expanded. Once a node has been popped and its neighbors have been processed, the node is removed from the heap and therefore does not appear in the later 'min-heap'. Each step has two time-consuming stages: 1) an SSD access stage (reading neighbors of the popped node), and 2) a distance computation stage (calculating distances of these neighbor nodes to the query vector and updating the candidate min-heap).

These two stages exhibit two types of dependencies: a) **intra-step dependency** that requires distance computation and uses the results of SSD access from the same step, and b) **inter-step dependency** that requires SSD access in the current step, uses the candidate min-heap updated by

the computation results of the previous step. The dependencies prevent the overlapping of the two stages.

4.1.2 Dependency Relaxed Strategy. To mitigate low computational resource utilization, we propose relaxing inter-step dependencies to overlap the computation and I/O stages. This approach is grounded in a key insight: the graph search process exhibits a natural tolerance to staleness (using candidate sets from up to k steps prior). This insight is supported by two observations. First, our analysis on the SIFT1B dataset reveals that only 24.3% of search steps directly depend on the immediately up-to-date candidate min-heap. Second, even with the stale candidate min-heap, the search direction often remains valid. Consequently, the total number of additional steps incurred by staleness is limited and slight. As shown in Figure 10, the step count rises by just 2.4% to 9.8% per additional staleness step.

Upon the inherent tolerance of graph search to staleness, we implement a dependency-relaxed pipeline to maximize resource utilization. As Figure 9b shows, we break the inter-step dependency by allowing the SSD access stage of the current step **i** to start without waiting for the completion of the distance calculation stage in the previous step **i-1**. The node selected for access in the step **i** is chosen from the candidate min-heap updated based on the distance calculation results from the step **i-2**. As illustrated in Figure 8, in this case, the step **i** proceeds without incorporating the distance calculation results from the step **i-1**, which may overlook a closer node that should be chosen in the step **i** in the traditional best-first search. As such, the staleness mechanism may require slightly more steps to search the vector.

However, as shown in the comparison of time consumption between Figure 9a and Figure 9b, the staleness algorithm allows the overlapping of the SSD access stage and distance calculation stage, and the overall execution time can be greatly reduced even with slightly more steps.

4.1.3 Convergence Analysis. We provide the convergence guarantee of the search algorithm under the staleness mechanism with bounded search steps.

Define Δ_t as the maximal positional deviation between paths $\mathcal{P}_{\text{relax}}$ and $\mathcal{P}_{\text{strict}}$ at the step t, constrained by at most k-step directional variances. The deviation satisfies:

$$\Delta_t \leq \Delta_{t-1} + k \quad \text{for} \quad t \geq 1 \tag{1}$$

with initial condition at $t = 0$:

$$\Delta_0 = 0 \tag{2}$$

since no staleness exists at initialization.

By mathematical induction over steps:

- **Base Case ($t = 0$).** Trivially $\Delta_0 = 0 \leq k \cdot 0$
- **Inductive Step.** Assume $\Delta_{t-1} \leq k(t-1)$. Then

$$\Delta_t \leq \Delta_{t-1} + k \leq k(t-1) + k = kt \tag{3}$$

After T steps, the maximal cumulative detour depth d satisfies:

$$d \leq \Delta_T \leq kT \tag{4}$$

Consequently, the total traversal steps of the relaxed path are bounded by:

$$|\mathcal{P}_{\text{relax}}| \leq \underbrace{T}_{\text{strict path steps}} + \underbrace{k \times T}_{\text{max detour steps}} = (k+1) \times T \tag{5}$$

This guarantees that the relaxed search converges within a bounded number of additional steps determined by k.

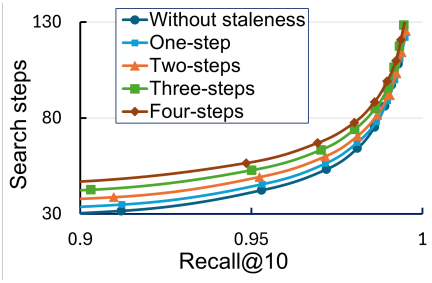


Fig. 10. Search step count under different staleness steps

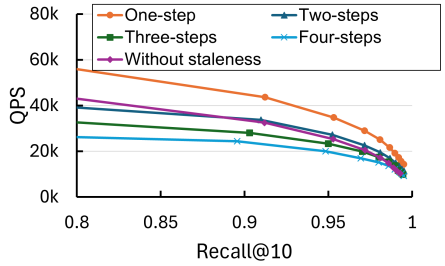


Fig. 11. End-to-end QPS performance under different staleness steps

4.1.4 Staleness Step Selection. In FlashANNS, we fix the staleness step at k = 1 as the default optimal configuration. This decision is grounded in a co-design principle under our graph degree selector (Section 4.3). The degree selector tunes the graph degree before index construction to explicitly balance the per-step GPU computation time T_c and SSD I/O time T_f, making them approximately equal. Under this balanced condition, a staleness of $k = 1$ is sufficient to achieve full overlap between the computation and I/O stages, thereby maximizing pipeline efficiency without introducing unnecessary latency, as illustrated in Figure 9b.

Increasing the staleness to k = 2 or higher is not beneficial. For example, increasing the staleness to k=2 does not reduce the intrinsic durations of T_f and T_c. While higher staleness can initiate data fetches earlier, the computation unit cannot finish its work ahead of time. As a result, as shown in Figure 12, the pipeline cycle time remains unchanged compared to k=1. Furthermore, as shown in Figure 10, a larger k generally increases the total number of search steps due to the use of a more stale candidate, ultimately resulting in higher overall query latency. The same rationale applies to even larger values of k. As shown in Figure 11, k=1 consistently achieves the highest QPS in FlashANNS's end-to-end evaluation.

4.2 Resource-Efficient Query-Grained Concurrent I/O Stack

To address C2, we introduce a resource-efficient, query-grained concurrent I/O stack tailored to ANNS workloads. It overcomes the limitations of existing I/O stacks when processing the iterative, batched, and fine-grained SSD requests characteristic of ANNS workloads. Our design facilitates direct GPU-SSD data transfer with minimal GPU overhead, entirely bypassing the host OS file system.

Design and Execution of the Query-Grained I/O Stack. As illustrated in Figure 13, FlashANNS implements the query-grained I/O stack with three components: (1) A CPU-hosted I/O request array,

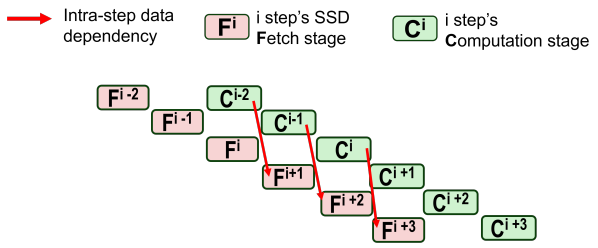


Fig. 12. Two-steps staleness pipeline

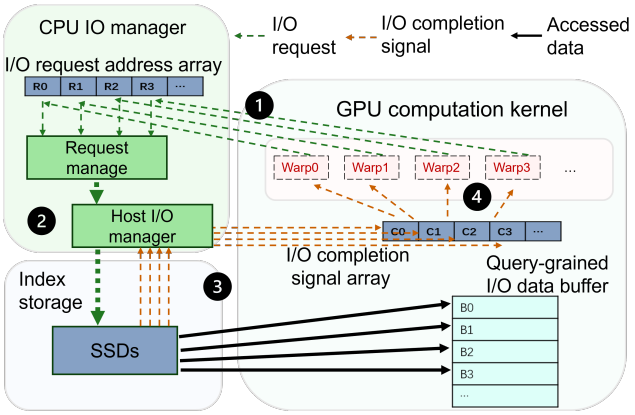
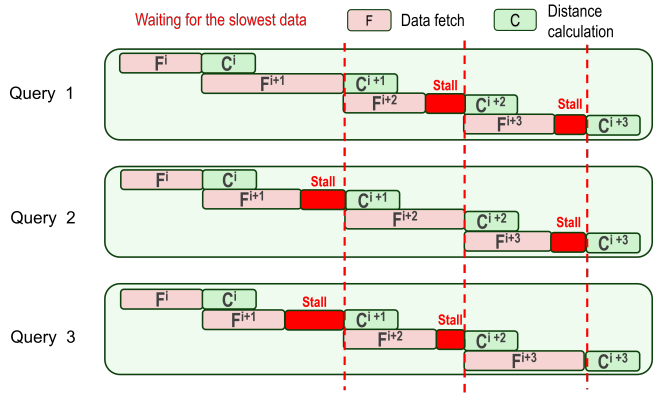


Fig. 13. I/O process of FlashANNS

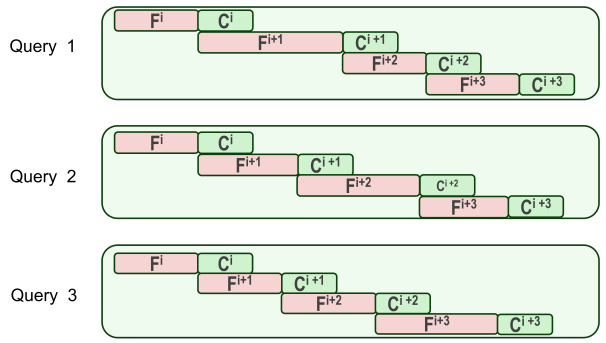
(2) A GPU-resident completion signal array (3) A GPU data buffer. Each array element corresponds to a single query, and each query is executed by a warp (a group of 32 threads in GPU computing).

When issuing an SSD read, the pipeline proceeds in four stages that enable computation and I/O overlap: (❶) Each warp writes its target SSD block address into a request buffer that is allocated in pinned host memory and mapped into the GPU address space using CUDA's mapped memory. After submitting the address, the warp is free to resume its computational work, without blocking for the I/O request to complete. (❷) A CPU agent polls this array, batches the addresses into read requests, and submits them to the SSD. (❸) On completion, the SSD DMA transfers the payload directly into the query-grained data buffer in GPU global memory, and the CPU I/O threads post a completion signal to the GPU-side array. (❹) When a warp finishes its current computational tasks and reaches a synchronization point, it checks for the completion signal. If the data is ready, the warp retrieves the data.

ANNS workload's unique character and challenge ANNS workloads are characterized by a large batch of queries, where the processing of each query involves an iterative execution of the SSD access stage and GPU computation stage. This introduces three challenges: First, ANNS workloads contain a massive number of small, random I/Os. Second, our proposed SSD access/GPU computation overlapping requires handling SSD accesses asynchronously. Third, synchronizing SSD accesses across different queries causes the long-tail latency (stragglers) of individual I/Os to delay the entire batch. In an iterative search process, these delays accumulate significantly, severely degrading overall throughput. To the best of our knowledge, none of the existing I/O stacks address these challenges simultaneously.



(a) Kernel-Grained SSD access



(b) Query-Grained SSD access

Fig. 14. Pipeline performance comparison: kernel-grained vs. query-grained SSD access

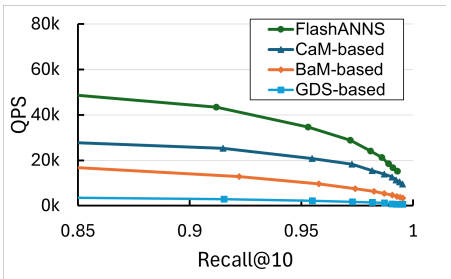


Fig. 15. End-to-End QPS performance with various I/O stack-based FlashANNS

Limitations of existing GPU–SSD IO stacks. Existing I/O stacks introduce critical bottlenecks that undermine performance. **GDS (GPU Direct Storage)** [55] bypasses the CPU for data transfer but relies on the host OS filesystem for control operations. This reliance necessitates system calls and frequent kernel-user mode transitions, which incur substantial overhead when managing the massive number of small, random I/Os typical in ANNS. **BaM (GPU-Initiated On-Demand Storage)** [58] employs a GPU-centric I/O control path. BaM can't resolve the second challenge due

to its synchronous interface. BaM's threads are forced to wait for their I/O requests to complete instead of performing computations. Thereby, it prevents the overlapping of computation and I/O stages in ANNS workloads. **CAM (Asynchronous GPU-Initiated, CPU-Managed SSD Management)** [64] attempts a balanced approach with asynchronous, CPU-managed access but enforces a kernel-grained global synchronization model. This means the GPU must wait for all I/O requests in a batch to complete before proceeding. When facing the third challenge of ANNS offload, as shown in Figure 14a, CAM's synchronization mechanism waits for the completion of all pending SSD accesses across different queries and thus becomes a source of significant latency inflation.

How does the query-grained I/O stack overcome the limitations of existing designs? To overcome GDS's overhead, FlashANNS leverages SPDK to entirely bypass the kernel stack, thereby eliminating filesystem overhead. To improve BaM's synchronous design, FlashANNS offloads I/O management to the CPU, and implements an asynchronous data-passing mechanism to make computation and IO process parallelize. To mitigate CAM's long waiting time for batch SSD access, as shown in Figure 14b, FlashANNS enables each query to independently issue I/O requests and receive completion signals, thereby preventing straggler requests from delaying the entire batch.

We evaluate FlashANNS with three alternative I/O stacks integrated into its architecture: GDS, BaM, and CAM. As shown in Figure 15, under the 4-SSD SIFT1B setup, our custom I/O stack demonstrates superior performance, achieving 14.5×, 3.9×, and 1.5× higher QPS than these respective alternatives. These results conclusively demonstrate that FlashANNS' I/O stack delivers superior performance and is better optimized for the access patterns of ANNS workloads compared to existing solutions.

4.3 Sampling-Based Graph Degree Selector

To address C3, we propose a sampling-based graph degree selector. This is a pre–index-construction procedure. It analyzes pipeline behavior on sample indices with different degrees to estimate the relative latencies of I/O and computation, and then selects the degree configuration that maximizes pipeline overlap by making use of the wasted bandwidth caused by I/O amplification.

4.3.1 Impact of Graph Degree on Computation / IO Bottleneck. The I/O characteristics of SSDs dictate that when the access granularity is no more than 4KB, IOPS instead of bandwidth becomes the primary performance bottleneck. Graph-index's node sizes are mostly smaller than 4KB. For instance, in DiskANN's [31] default configuration(degree 64) for SIFT1B, a node is 384 bytes. In this case, SSD access is usually performed in 4KB blocks (as noted in DiskANN). This mismatch between the SSD access block and the graph-index's size node results in significant I/O amplification, as each read retrieves an entire 4KB block but utilizes only a small portion.

The node is composed of an original full-precision vector and indices of its neighbors. While increasing the graph degree to inflate the node size to 4KB seems like a direct solution to eliminate I/O amplification, it introduces a different issue. A higher degree linearly increases the per-step GPU computation time, as more neighbors necessitate more distance calculations. Meanwhile, adding more neighbors yields diminishing marginal returns. Therefore, selecting the optimal graph degree requires a careful balance between I/O and computational load, and must be adapted to the available SSD bandwidth.

As mentioned in Section 4.1.4, to enable the one-step staleness pipeline, the degree selector chooses to balance the per-step GPU computation time T_c and SSD I/O time T_f, making them approximately equal.

4.3.2 Workflow of Graph Degree Selector. Prior to the full index construction, FlashANNS uses the lightweight graph degree selector to determine the optimal graph degree. This process operates on

a compact data sample (e.g., 100k nodes, index size<200MB) that matches the target dataset's data type and dimensionality. It constructs temporary graph indices for a set of candidate degrees (e.g., 64, 150, 250), where edges are formed using random links rather than true neighbor relationships. It is sufficient to accurately probe the memory and I/O patterns for each degree. Using the same runtime pipeline and a short warm-up of synthetic queries, the selector measures for each candidate d the data fetch latency $T_f(d)$ and per-step computation latency $T_c(d)$. The objective is to **make I/O and computation take the same time per step to maximize pipeline overlap**, so d is calculated as

$$d \;=\; \arg\min_{d} \,|\, T_{\mathrm{c}}(d) - T_{\mathrm{f}}(d) \,|\,. \tag{6}$$

4.3.3 Overhead of Graph Degree Select. By operating on a small sample (e.g., 100k nodes, 0.01% of a billion-scale dataset) with random links, this profiling avoids the cost of building true neighborhoods, making both graph construction and performance measurement extremely low-cost. The entire process completes in minutes, incurring less than 1% overhead compared to the multi-hour runtime of a full index construction.

4.3.4 Hardware Adaptation via the Degree Selector. The degree selector equips FlashANNS with the capability to effectively utilize diverse hardware settings. The degree selector guides the user to leverage hardware improvements by re-balancing the computational load T_c and the I/O latency T_f, ensuring the pipeline remains efficient.

In case of using SSDs with higher IOPS, the degree selector guides the user to decrease the graph degree. This reduces T_c to realign with the shorter T_f, thereby shortening the pipeline cycle and thus accelerating queries. In case of using a faster GPU, the degree selector guides the user to increase the graph degree. This leverages the additional computational capacity to examine more neighbors per step while maintaining the same pipeline cycle time, and ultimately reduces the total number of search steps and thus speeds up overall query execution.

5 Evaluation

Our evaluations aim to answer the following questions:

- How does the performance of FlashANNS compare to other SSD-based frameworks on a single SSD and multiple SSDs (§5.2)?
- How effective is the dependency-relaxed asynchronous I/O pipeline (§5.3)?
- How effective is the query-grained concurrent SSD access (§5.4)?
- How does throughput scale when increasing the returned top-k (§5.5)?
- How does the computation-I/O-balanced graph degree selector guide degree selection (§5.6)?
- How does FlashANNS perform on larger-than-memory indices (§5.7)?

5.1 Experimental Setting

Experimental Platform. The experiments are conducted using a single server equipped with dual Intel Xeon Gold 5320 processors operating at 2.20 GHz (52 threads), 768 GB of DDR4 system memory, and an NVIDIA 80 GB-PCIe-A100 GPU. The storage subsystem comprises eight 3.84TB Intel P5510 NVMe SSDs configured in a PCIe 4.0 x16 topology. The platform runs Ubuntu 22.04 LTS.

Datasets. Our experimental configuration incorporates three canonical billion-scale datasets extensively adopted in high-dimensional similarity search benchmarks:

- **SIFT-1B** comprising 1 billion 128-dimensional vectors with unsigned 8-bit integer (uint8) precision, evaluated using 10,000 query instances.

- **DEEP-1B** featuring 96-dimensional floating-point vectors (float32) across 1 billion entries, benchmarked with 10,000 queries.
- **SPACEV-1B** containing 100-dimensional signed 8-bit integer (int8) vectors at billion-scale, tested with 29,300 queries.

Baselines. FlashANNS has three baselines to compare.

- **SPANN [11]**: A clustering-based SSD-resident framework that stores cluster lists on SSDs while maintaining cluster centroids in CPU memory. Its search mechanism achieves low latency at the cost of computationally intensive per-query operations.
- **DiskANN [31]**: A graph-indexed SSD-resident framework that stores both index graphs and raw vectors on SSDs, complemented by product quantization compressed vectors in CPU memory. It achieves optimized performance through parallel searching.
- **FusionANNS [69]**: A GPU-accelerated clustering-based SSD-resident framework leveraging cooperative CPU-GPU processing to optimize ANNS.

Evaluation Metrics. We quantify query throughput in queries per second (QPS) and accuracy via recall@10. Recall@10 measures the proportion of true top-10 neighbors retrieved from ground truth among ANNS-returned candidates. Cluster-indexing systems tune recall by adjusting the count of retrieved posting lists during graph traversal, whereas graph-indexing systems control recall by configuring the candidate min-heap size. On QPS-recall@10 tradeoff curves, superior implementations occupy top-right positions, achieving higher QPS with greater accuracy under identical configurations.

5.2 End-to-End QPS-Recall Tradeoff

We evaluate FlashANNS against all SSD-based baselines [11, 31, 69] by maximizing CPU threads to 52. Figure 8 reports query throughput versus recall@10 across three datasets, scaling SSD count from 1 to 8 for index storage.

Comparing under Single SSD Configuration. We first examine QPS-recall@10 performance under single-SSD configurations. The top row of Figure 16 shows FlashANNS achieving 2.7-5.9× query throughput gains over CPU-based baselines (SPANN/DiskANN) at 96% recall@10. Against GPU-enhanced FusionANNS, FlashANNS delivers comparable query throughput (1.03-1.4×) on SIFT1B and SPACEV1B at 98% recall@10, where constrained I/O bandwidth reduces computational demands and avoids CPU bottlenecks. However, on DEEP1B at 98% recall@10, FlashANNS achieves 14.3× higher query throughput than FusionANNS. This performance discrepancy originates from DEEP1B's single-precision floating-point (FP32) format, which imposes substantial CPU pressure on FusionANNS during precision distance calculations, revealing CPU-bound limitations. Overall, FlashANNS consistently occupies the top-right region of QPS-recall@10 curves, demonstrating superior query throughput through parallel pipeline optimization and I/O stack efficiency under limited I/O bandwidth.

Comparison under an Increasing Number of SSDs. As available I/O bandwidth increases with additional SSDs, FlashANNS demonstrates disproportionately greater performance improvements than alternatives by navigation graph degree optimizing and I/O-computation latency balancing. Using the SIFT1B exemplar data in Figure 16 (first column), FlashANNS delivers progressively higher query throughput at 98% recall@10—achieving 1.4×–7.0×, 2.0×–5.4×, 2.7×–5.9×, and 3.9×–5.7× QPS gains over baselines when scaling from 1 to 8 SSDs.

Comparison to the In-Memory Baseline. Furthermore, we observe that FusionANNS underperformed relative to expectations in high I/O bandwidth scenarios. Our analysis suggests this stems from implicit inefficiencies in FusionANNS' I/O submission queue scheduling mechanisms.

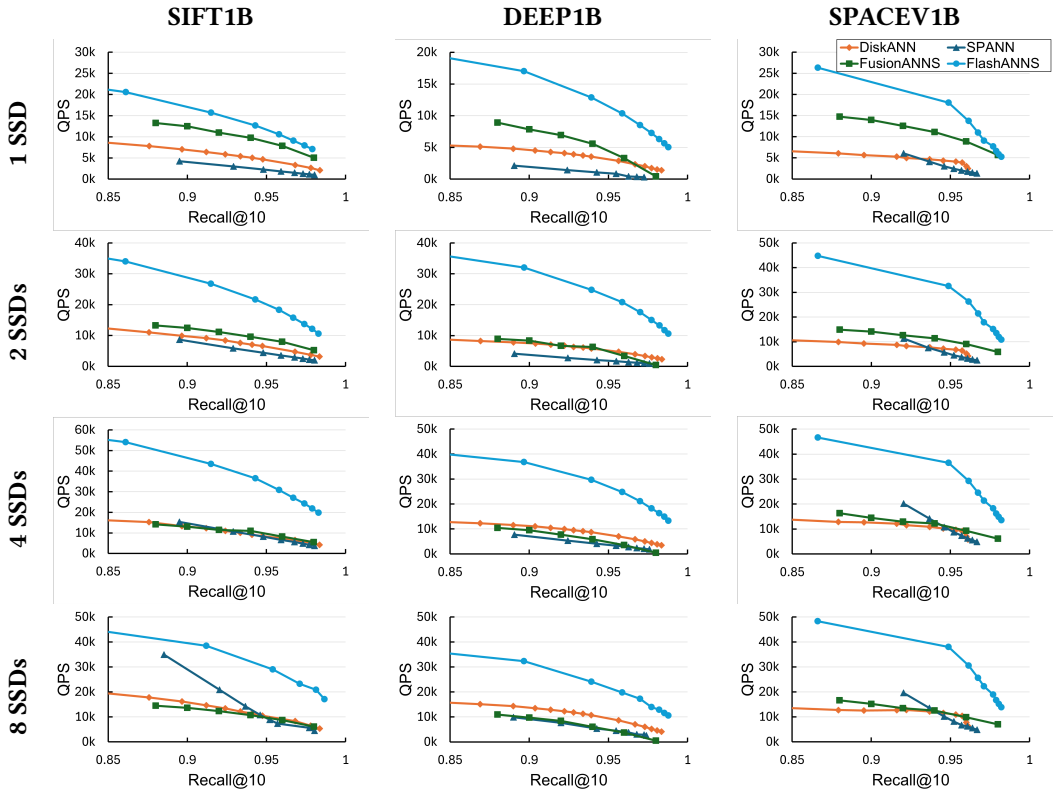


Fig. 16. The query throughput of different ANNS systems varying with the recall@10 on different SSD counts

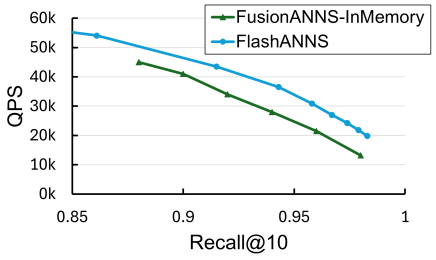


Fig. 17. FlashANNS's QPS performance compared with in-memory FusionANNS.

To rigorously isolate algorithmic and architectural advantages from implementation-specific biases, we conduct a controlled comparison: we evaluate FlashANNS (using 4 SSDs) against an in-memory variant of FusionANNS (all indices loaded into DRAM), thereby removing any confounding effects of storage-layer optimizations. As shown in Figure 17, FlashANNS achieves higher performance than the in-memory FusionANNS baseline. This result demonstrates that FlashANNS' SSD-based implementation maintains a performance advantage over even the in-memory indexed FusionANNS.

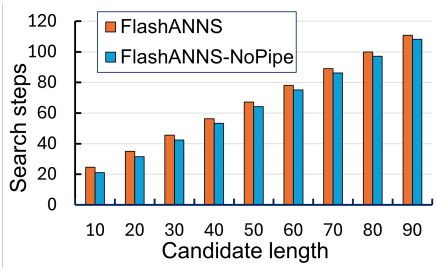


Fig. 18. Query throughput of FlashANNS vs. FlashANNS-Nopipe under different candidate min-heap lengths

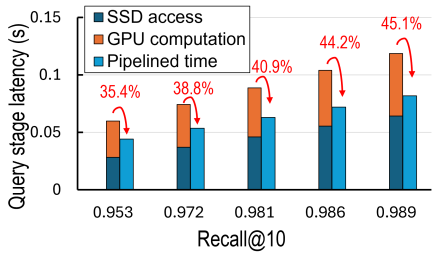


Fig. 19. FlashANNS's query latency breakdown of SSD access, GPU computation and total pipelined time under different recall rates

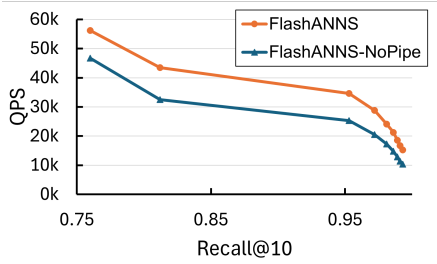


Fig. 20. End-to-end QPS performance of FlashANNS and the FlashANNS-Nopipe

5.3 Impact of Dependency-Relaxed Asynchronous I/O Pipeline

In this subsection, we quantitatively evaluate how dependency-relaxed pipeline parallelism affects query throughput during execution. We separately evaluate: 1) the effect of dependency relaxation on step count in graph queries, 2) the latency reduction achieved through execution overlapping, and 3) the improvement in overall query throughput.

Impact on Searching Steps. To elucidate the impact of dependency relaxation on search procedures, we present two implementations of FlashANNS: the pipeline version and the no-pipe variant. The pipeline implementation introduces dependency relaxation to enable concurrent execution between SSD read operations and data computation during graph traversal, whereas the no-pipe variant strictly adheres to the strong dependency constraints inherent in best-first search algorithms, enforcing serialized execution between SSD access and computational processing.

Figure 18 illustrates the impact of dependency relaxation on search path lengths, comparing the pipeline and best-first search implementations on the sift1B dataset under a 250-degree graph

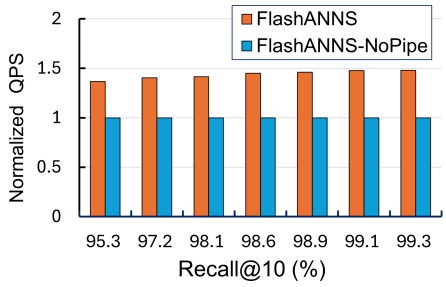


Fig. 21. Normalized end-to-end QPS performance comparison of FlashANNS and the no pipeline version

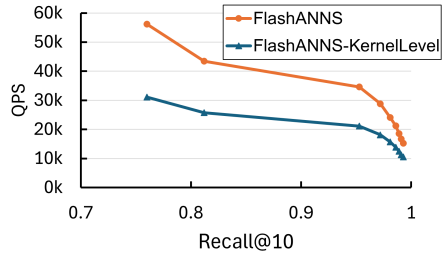


Fig. 22. End-to-end QPS performance of FlashANNS and kernel-grained access version

configuration. We observe that the relaxed-dependency scheme incurs only 2.8-3.2 additional steps, representing a mere 2.5%-7% overhead relative to the total query steps in the baseline search strategy. We conclude that dependency relaxation induces only minor variations in search step counts.

Impact on Query Latency. This section quantifies pipeline overlap characteristics in terms of latency. Evaluated on the SIFT-1B dataset with a 250-degree graph and 4-SSD parallelism, Figure 19 compares the average query latency of pipelined FlashANNS against the summed latency of its SSD access phase and GPU computation phase. The experimental results show that the overlapped execution time accounts for 35.4% to 45.1% of the total pipelined latency across different recall rates. This demonstrates that FlashANNS effectively overlaps the SSD accesses and GPU computation stages, thereby reducing search latency and improving QPS.

Overall Pipeline Performance. Collectively, the dependency-relaxed asynchronous I/O pipeline trades only 2.5%-7% additional computational steps for 36%-47% pipeline overlap ratio, thereby improving the overall query throughput. As Figures 20 and 21 demonstrate, this approach delivers 33.6%–46.6% higher QPS compared to the no-pipeline variant under the same recall.

5.4 Impact of Query-grained Concurrent SSD Access

To validate the effectiveness of query-grained concurrent SSD access, we employ an unoptimized kernel-grained I/O stack (CAM [64]) for comparison. Unlike query-grained SSD access that treats individual warp operations as autonomous units, the kernel-grained approach aggregates all SSD requests issued during a kernel function's execution phase into a batch process. This requires completing full data retrieval for an entire request group before initiating subsequent data processing and next-stage access operations. In contrast, query-grained SSD access enables a finer-grained

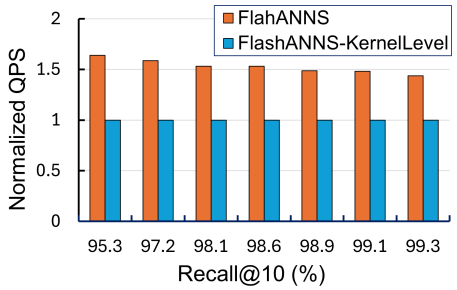


Fig. 23. Normalized end-to-end QPS performance of FlashANNS and kernel-grained access version

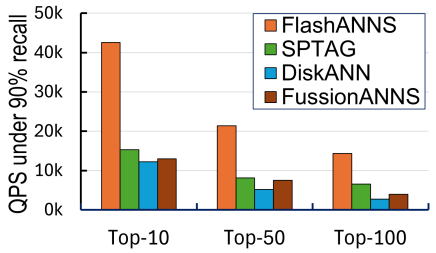


Fig. 24. End-to-end QPS performance under different top-K sizes

synchronization mechanism for immediate processing of subsequent read requests upon completion of queries handled by individual warps.

We evaluate our experiment on the sift1B dataset with a 4-SSD configuration under 250-degree graph parameters. As Figure 22 and Figure 23 show, the query-grained SSD access implementation achieves 43%-68% query throughput improvement compared to its kernel-grained counterpart CAM. These performance gains are attributed primarily to the finer synchronization granularity of query-grained operations, which effectively amortize sporadic long-tail SSD read latencies from ANNS systems.

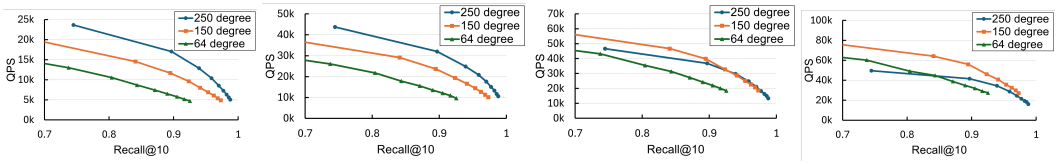
5.5 Scalability to Larger Return Sets

To validate FlashANNS' performance with a larger returned set, we compare FlashANNS to three state-of-the-art baselines on SIFT-1B (4-SSD) under more top-K sizes. For each method and $K \in \{10, 50, 100\}$, we tune the parameters to measure the achievable QPS for different top-K sizes when the recall rate is higher than 90%. As shown in Figure 24, FlashANNS delivers **2.2–5.2×** higher QPS than the other three baselines, demonstrating that it can maintain high query throughput under different K sizes.

5.6 Process of Computation I/O-Balanced Graph Degree Selection

We conduct index construction experiments across three graph degree configurations (64, 150, and 250 degrees) under varying SSD parallelism conditions (1, 2, 4, and 8 SSDs). Our evaluation systematically measures query throughput characteristics across different I/O bandwidth settings. Results demonstrate the effectiveness of selecting graph degrees via sample-indexed analysis.

We evaluate the QPS-recall tradeoffs of varying graph degrees on the DEEP1B dataset under different I/O bandwidth configurations. As demonstrated in Figure 25, under low I/O bandwidth



(a) 1 SSD (b) 2 SSDs (c) 4 SSDs (d) 8 SSDs

Fig. 25. FlashANNS QPS-recall performance on the dataset DEEP1B dataset under different SSD configurations

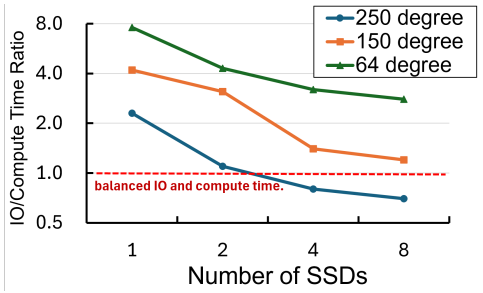


Fig. 26. I/O-computation time ratio of sample indices in different SSD counts

constraints (with indices stored on 1-2 SSDs), higher-degree graphs consistently achieve superior query throughput while satisfying equivalent recall targets compared to lower-degree graphs. This advantage stems from their enhanced ability to leverage GPU parallel computation units to mask I/O latency when SSD bandwidth is scarce.

However, as I/O bandwidth increases with higher SSD parallelism (up to 8 SSDs), the query throughput gap between high-degree and low-degree graphs narrows significantly. Notably, in the 8-SSD configuration, the 150-degree graph outperforms its 250-degree counterpart by 13% in query throughput under 96% recall@10. Beyond critical bandwidth thresholds, the higher computational demands inherent to high-degree graph processing (e.g., requiring access to more neighbors at each traversal step) may conversely create computational bottlenecks.

Before determining the optimal graph degree for index construction, we create sample indices (as described in Section 4.3) — which share the same data type as the vector index but exclude actual neighbor relationships — to pre-estimate pipeline stage latency. Experimental measurement shows the I/O-computation ratio characteristics for different graph degree sample indices in Figure 26. These sample indices reflect the actual performance characteristics of full-scale index graphs at corresponding degree configurations.

With 1 SSD, the I/O latency of the 150-degree graph is 4.2× its computational latency, while the 250-degree graph exhibits I/O latency at 2.3× computational latency, indicating that both configurations remain I/O-bound, thus higher-degree graphs retain their advantage.

With 2 SSDs, the 150-degree graph's I/O latency becomes 3.1× computational latency (still I/O-bound), whereas the 250-degree graph achieves near-balance at 1.1× I/O-to-compute latency ratio, enabling full pipeline utilization.

With 4 SSDs, the 150-degree graph's I/O latency reduces to 1.4× computational latency (marginally I/O-bound), while the 250-degree graph becomes compute-bound (0.7× I/O-to-compute ratio). This implies the optimal degree lies between 150 and 250 to avoid asymmetrical bottlenecks.

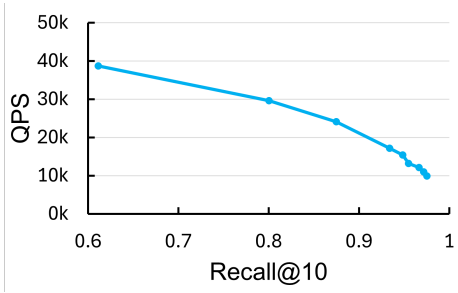


Fig. 27. FlashANNS' QPS performance under TB-scale dataset

The performance trends predicted by our degree sampling via lightweight sample indices align closely with actual test results, empirically validating the effectiveness of graph degree pre-selection for workload-aware index optimization.

5.7 Out-of-Core Efficiency on 30B-Vector Dataset

To demonstrate FlashANNS' processing capability to out-of-core indices, we augment the DEEP1B dataset to a 30 billion-vector dataset (1,073 GB) by duplicating each vector twice with minor perturbations, thereby expanding its scale while preserving proximity relationships. Figure 27 shows that FlashANNS achieves good QPS-recall trade-offs on this exabyte-scale dataset, validating FlashANNS's capability to efficiently perform ANNS on huge datasets far exceeding DRAM capacity for RAG applications.

6 Related Works

To our knowledge, this work presents the first GPU-accelerated, SSD-based graph ANNS framework. While Section 5 provides comprehensive comparisons with state-of-the-art SSD-based ANNS systems (DiskANN, SPANN, FusionANNS), we review related work in two key areas: in-memory ANNS frameworks and SSD I/O optimization implementations.

In-Memory ANNS Frameworks. In-memory ANNS systems are widely utilized, with their index structures primarily categorized into four types: 1. Tree-based indices [7, 9, 15, 21, 50, 51, 62, 78]: The core premise centers on constructing a tree-like data structure that hierarchically organizes vectors via partitioning criteria based on distance or density metrics. 2. Hash-based [2–4, 16, 23, 67]: Locality-Sensitive Hashing (LSH) maps high-dimensional vectors into hash buckets while preserving similarity relationships, enabling efficient approximate nearest neighbor search. 3. Quantization-based [5, 6, 22, 32, 36, 53, 76]: This method divides high-dimensional vectors into subvectors, independently quantizes each subvector into compact codes. This method reduces the storage footprint and accelerates similarity computation through lookup tables. 4. Graph-based [18, 20, 26, 34, 38, 46, 47]: Graph-based indices demonstrate superior search performance in Euclidean spaces due to their explicit modeling of local neighbor proximity. In particular, the edges in the graph structure directly encode vector adjacency relationships, enabling greedy traversal toward nearest neighbors. In contrast, FlashANNS focuses on out-of-order ANNS systems while achieving slightly higher performance than the in-memory counterparts.

SSD I/O Optimizations. Recently, the SSD has been deployed in many applications for its massive storage volume while achieving high performance compared to traditional hard disk drives [1, 25, 35, 48]. There are many works proposed to exploit the potentialities of SSD [1, 10, 12, 17, 19, 27, 29, 40–42, 44, 45, 48, 57, 60, 65, 66, 68, 70, 72–75, 80]. Existing works [79] achieve direct GPU-SSD data

transfer using the GPUDirect [54] technology. However, it relies on the CPU to initiate or trigger SSD access and fails to eliminate the OS kernel overhead entirely. Systems [61] do not support GPUDirect [54]. These methods involve OS kernel overheads, especially making it hard to saturate SSD throughput for batching access patterns of ANNS workloads.

BaM [58] proposes GPU-initiated on-demand direct SSD access without CPU involvement. However, BaM's design introduces new GPU core contention issues, which are intended to be used in computation tasks, while CAM [64] can achieve high throughput without GPU SM occupancy. However, its implementation enforces kernel-grained global synchronization: all warps within a kernel must wait for the slowest SSD I/O request to complete before proceeding. In contrast, FlashANNS enables query-grained concurrent SSD access to accelerate GPU-based ANNS.

7 Conclusion

In this work, we present FlashANNS, a GPU-accelerated, out-of-core graph-based ANNS system. Our approach achieves full I/O-compute overlap through three coordinated mechanisms: First, a dependency relaxation I/O pipeline to overlap SSD node retrieval with GPU computation while providing a rigorous theoretical convergence guarantee. Second, a query-grained concurrent SSD access that eliminates the GPU kernel-grained global synchronization inherent. Third, a hardware-aware graph degree selection mechanism maximizing pipeline stage overlap efficiency. Experimental results show that under the same recall accuracy, FlashANNS achieves 2.7–5.9× higher query throughput than existing SOTA methods with a single SSD configuration, and scales to 3.9–12.2× query throughput improvements in multi-SSD setups.

Acknowledgments

The work is supported by the following grants: National Science and Technology Major Project (2022ZD0117000), the Major Project of the Zhejiang Provincial Natural Science Foundation under Grant No. LD26F020002, the National Natural Science Foundation of China under the grant numbers (62472384, 62441236, U24A20326), Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study (SN-ZJU-SIAS-0010). Jie Zhang, Zeke Wang, and Fei Wu are the corresponding authors.

References

[1] Gustavo Alonso, Natassa Ailamaki, Sailesh Krishnamurthy, Sam Madden, Swami Sivasubramanian, and Raghu Ramakrishnan. 2023. Future of Database System Architectures. In *Companion of the 2023 International Conference on Management of Data*. 261–262.

[2] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.

[3] Alexandr Andoni, Piotr Indyk, Huy L Nguyễn, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1018–1028.

[4] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal Data-Dependent Hashing for Approximate Near Neighbors. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing* (Portland, Oregon, USA) *(STOC '15)*. Association for Computing Machinery, New York, NY, USA, 793–801. doi:10.1145/2746539.2746553

[5] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[6] Artem Babenko and Victor Lempitsky. 2015. Tree quantization for large-scale similarity search and classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4240–4248.

[7] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. 97–104.

[8] Yuan Cao, Heng Qi, Wenrui Zhou, Jien Kato, Keqiu Li, Xiulong Liu, and Jie Gui. 2018. Binary Hashing for Approximate Nearest Neighbor Search on Big Data: A Survey. *IEEE Access* 6 (2018), 2039–2054. doi:10.1109/ACCESS.2017.2781360

[9] Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning*. 112–119.

[10] Yunpeng Chai, Yanfeng Chai, Xin Wang, Haocheng Wei, Ning Bao, and Yushi Liang. 2019. LDC: a lower-level driven compaction method to optimize SSD-oriented key-value stores. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 722–733.

[11] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[12] Jiajia Chu, Yunshan Tu, Yao Zhang, and Chuliang Weng. 2020. LATTE: A native table engine on NVMe storage. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1225–1236.

[13] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. doi:10.1145/2959100.2959190

[14] Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. 2023. ChatLaw: Open-Source Legal Large Language Model with Integrated External Knowledge Bases. (2023). https://openreview.net/forum?id=Cjas49BCAf

[15] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 537–546.

[16] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[17] Jaeyoung Do, Ivan Luiz Picoli, David Lomet, and Philippe Bonnet. 2021. Better database cost/performance via batched I/O on programmable SSD. *The VLDB Journal* 30, 3 (2021), 403–424.

[18] Wei Dong. 2011. *High-dimensional similarity search for large datasets*. Princeton University.

[19] Carl Duffy, Jaehoon Shim, Sang-Hoon Kim, and Jin-Soo Kim. 2023. Dotori: A key-value ssd based kv store. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1560–1572.

[20] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).

[21] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers* 100, 7 (1975), 750–753.

[22] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

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

[24] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time Personalization using Embeddings for Search Ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 311–320. doi:10.1145/3219819.3219885

[25] Gabriel Haas and Viktor Leis. 2023. What modern nvme storage can do, and how to exploit it: High-performance i/o for high-performance storage engines. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2090–2102.

[26] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22. 1312.

[27] Michael Haubenschild, Caetano Sauer, Thomas Neumann, and Viktor Leis. 2020. Rethinking logging, checkpoints, and recovery for high-performance storage engines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 877–892.

[28] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2553–2561. doi:10.1145/3394486.3403305

[29] Yuchen Huang, Xiaopeng Fan, Song Yan, and Chuliang Weng. 2024. Neos: A nvme-gpus direct vector service buffer in user space. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3767–3781.

[30] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot Learning with Retrieval Augmented Language Models. 24, 251 (2023), 1–43. http://jmlr.org/papers/v24/23-0037.html

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

[32] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[33] 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* abs/2404.12457 (2024). https://api.semanticscholar.

org/CorpusID:269283058
[34] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.
[35] Yuhun Jun, Shinhyun Park, Jeong-Uk Kang, Sang-Hoon Kim, and Euiseong Seo. 2024. We ain't afraid of no file fragmentation: causes and prevention of its performance impact on modern flash SSDs. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies* (Santa Clara, CA, USA) *(FAST '24)*. USENIX Association, USA, Article 12, 16 pages.
[36] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2321–2328.
[37] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. *ArXiv* abs/2004.04906 (2020). https://api.semanticscholar.org/CorpusID:215737187
[38] Jon M Kleinberg. 2000. Navigation in a small world. *Nature* 406, 6798 (2000), 845–845.
[39] Atsutake Kosuge and Takashi Oshima. 2019. An Object-Pose Estimation Acceleration Technique for Picking Robot Applications by Using Graph-Reusing k-NN Search. In *2019 First International Conference on Graph Computing (GC)*. 68–74. doi:10.1109/GC46384.2019.00018
[40] Artem Kroviakov, Petr Kurapov, Christoph Anneser, and Jana Giceva. 2024. Heterogeneous Intra-Pipeline Device-Parallel Aggregations. In *Proceedings of the 20th International Workshop on Data Management on New Hardware*. 1–10.
[41] Bohyun Lee, Mijin An, and Sang-Won Lee. 2023. LRU-C: Parallelizing Database I/Os for Flash SSDs. In *VLDB*.
[42] Alberto Lerner and Gustavo Alonso. 2024. Data flow architectures for data processing on modern hardware. In *2024 IEEE 40th International Conference on Data Engineering*. IEEE, 5511–5522.
[43] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The design and implementation of a real time visual search system on JD E-commerce platform. In *Proceedings of the 19th International Middleware Conference Industry*. 9–16.
[44] Jing Li, Hung-Wei Tseng, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. 2016. Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics. *Proceedings of the VLDB Endowment* 9, 14 (2016), 1647–1658.
[45] Changyue Liao, Mo Sun, Zihan Yang, Jun Xie, Kaiqi Chen, Binhang Yuan, Fei Wu, and Zeke Wang. 2025. Ratel: Optimizing Holistic Data Movement to Fine-Tune 100B Model on a Consumer GPU. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 292–306.
[46] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68. doi:10.1016/j.is.2013.10.006
[47] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473
[48] Fabio Maschi and Gustavo Alonso. 2023. The difficult balance between modern hardware and conventional CPUs. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 53–62.
[49] Yitong Meng, Xinyan Dai, Xiao Yan, James Cheng, Weiwen Liu, Jun Guo, Benben Liao, and Guangyong Chen. 2020. PMD: An Optimal Transportation-Based User Distance for Recommender Systems. In *Advances in Information Retrieval*, Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva, and Flávio Martins (Eds.). Springer International Publishing, Cham, 272–280.
[50] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 11 (2014), 2227–2240. doi:10.1109/TPAMI.2014.2321376
[51] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (2002), 28–46.
[52] Jiongkang Ni, Xiaoliang Xu, Yuxiang Wang, Can Li, Jiajie Yao, Shihai Xiao, and Xuecang Zhang. 2023. Diskann++: Efficient page-based search over isomorphic mapped graph index using query-sensitivity entry vertex. *arXiv preprint arXiv:2310.00402* (2023).
[53] Mohammad Norouzi and David J Fleet. 2013. Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. 3017–3024.
[54] NVIDIA. 2011. NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect.
[55] NVIDIA Corporation. 2023. GPUDirect Storage. https://docs.nvidia.com/gpudirect-storage/index.html.
[56] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based News Recommendation for Millions of Users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1933–1942. doi:10.1145/3097983.3098108
[57] Tarikul Islam Papon. 2024. Enhancing data systems performance by exploiting SSD concurrency & asymmetry. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 5644–5648.

[58] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, CJ Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-mei Hwu. 2023. GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture. In *ASPLOS*.

[59] Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen tau Yih, Joëlle Pineau, and Luke Zettlemoyer. 2022. Improving Passage Retrieval with Zero-Shot Question Generation. In *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:248218489

[60] Zuocheng Shi, Jie Sun, Ziyu Song, Mo Sun, Yang Xiao, Fei Wu, and Zeke Wang. 2025. Moment: Co-optimizing Physical Communication Topology and Data Placement for Multi-GPU Out-of-core GNN Training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*. Association for Computing Machinery, New York, NY, USA, 250–264. doi:10.1145/3712285.3759788

[61] Mark Silberstein, Bryan Ford, Idit Keidar, and Emmett Witchel. 2013. GPUfs: Integrating a file system with GPUs. In *ASPLOS*.

[62] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8. doi:10.1109/CVPR.2008.4587638

[63] Solidigm. 2023. *D7-P5510 Series Product Page*. https://www.solidigm.com/products/data-center/d7/p5510.html

[64] Ziyu Song, Jie Zhang, Jie Sun, Mo Sun, Zihan Yang, Zheng Zhang, Xuzheng Chen, Fei Wu, Huajin Tang, and Zeke Wang. 2025. CAM: Asynchronous GPU-Initiated, CPU-Managed SSD Management for Batching Storage Access . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2309–2322. doi:10.1109/ICDE65448.2025.00175

[65] Jie Sun, Zuocheng Shi, Li Su, Wenting Shen, Zeke Wang, Yong Li, Wenyuan Yu, Wei Lin, Fei Wu, Bingsheng He, and Jingren Zhou. 2025. Helios: Efficient Distributed Dynamic Graph Sampling for Online GNN Inference. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Las Vegas, NV, USA) *(PPoPP '25)*. Association for Computing Machinery, New York, NY, USA, 2–15. doi:10.1145/3710848.3710854

[66] Jie Sun, Mo Sun, Zheng Zhang, Zuocheng Shi, Jun Xie, Zihan Yang, Jie Zhang, Zeke Wang, and Fei Wu. 2025. Hyperion: Co-Optimizing SSD Access and GPU Computation for Cost-Efficient GNN Training. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 321–335.

[67] Kengo Terasawa and Yuzuru Tanaka. 2007. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *Algorithms and Data Structures: 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007. Proceedings 10*. Springer, 27–38.

[68] Risi Thonangi and Jun Yang. 2017. On log-structured merge for solid-state drives. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 683–694.

[69] Bing Tian, Haikun Liu, Yuhang Tang, Shihai Xiao, Zhuohui Duan, Xiaofei Liao, Hai Jin, Xuecang Zhang, Junhua Zhu, and Yu Zhang. 2025. Towards High-throughput and Low-latency Billion-scale Vector Search via {CPU/GPU} Collaborative Filtering and Re-ranking. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. 171–185.

[70] Leonard Von Merzljak, Philipp Fent, Thomas Neumann, and Jana Giceva. 2022. What Are You Waiting For? Use Coroutines for Asynchronous I/O to Hide I/O Latencies and Maximize the Read Bandwidth!. In *ADMS@ VLDB*. 36–46.

[71] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[72] Yi Wang, Jiajian He, Kaoyi Sun, Yunhao Dong, Jiaxian Chen, Chenlin Ma, Amelie Chi Zhou, and Rui Mao. 2024. Boosting write performance of KV stores: An NVM-enabled storage collaboration approach. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2082–2095.

[73] Yi Wang, Jianan Yuan, Shangyu Wu, Huan Liu, Jiaxian Chen, Chenlin Ma, and Jianbin Qin. 2024. Leaderkv: Improving read performance of kv stores via learned index and decoupled kv table. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 29–41.

[74] Zhenghao Wang, Lidan Shou, Ke Chen, and Xuan Zhou. 2024. Bushstore: Efficient b+ tree group indexing for lsm-tree in non-volatile memory. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4127–4139.

[75] Jia Wei and Xingjun Zhang. 2022. How much storage do we need for high performance server. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 3221–3225.

[76] Yan Xia, Kaiming He, Fang Wen, and Jian Sun. 2013. Joint inverted indexing. In *Proceedings of the IEEE International Conference on Computer Vision*. 3416–3423.

[77] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, et al. 2023. Spfresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 545–561.

[78] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, Vol. 93. 311–21.
[79] Jie Zhang, David Donofrio, John Shalf, Mahmut T Kandemir, and Myoungsoo Jung. 2015. Nvmmu: A non-volatile memory management unit for heterogeneous gpu-ssd architectures. In *PACT*.
[80] Tobias Ziegler, Carsten Binnig, and Viktor Leis. 2022. ScaleStore: A fast and cost-efficient storage engine using DRAM, NVMe, and RDMA. In *Proceedings of the 2022 International Conference on Management of Data*. 685–699.

Received July 2025; revised October 2025; accepted November 2025