Efficient Index Layout and Search Strategy for Large-scale High-dimensional Vector Similarity Search

WEIJIAN CHEN, Southern University of Science and Technology, China
HAOTIAN LIU, AlayaDB AI, China
YANGSHEN DENG, University of Edinburgh, United Kingdom
LONG XIANG, AlayaDB AI, China
LIANG HUANG, Southern University of Science and Technology, China
BO TANG*, Southern University of Science and Technology, China

On-disk graph-based approximate nearest neighbor search (ANNS) is essential for large-scale, high-dimensional vector retrieval, yet its performance is widely recognized to be limited by the prohibitive I/O costs. Interestingly, we observed that the performance of on-disk graph-based index systems is compute-bound, not I/O-bound, with the rising of the vector data dimensionality (e.g., hundreds or thousands). This insight uncovers a significant optimization opportunity: existing on-disk graph-based index systems universally target I/O reduction and largely overlook computational overhead, which leaves a substantial performance improvement space.

In this work, we propose Laser, an efficient on-disk graph-based index system for large-scale high-dimensional vector similarity search. In particular, we first conduct performance analysis on existing on-disk graph-based index systems via the adapted roofline model, then we devise a novel on-disk data layout in Laser to effectively alleviate the compute-bound, which is revealed by the above roofline model analysis, by exploiting SIMD instructions on modern CPUs. We next design a suite of optimization techniques (e.g., degree-based node cache, cluster-based entry point selection, and early dispatch strategy) to further improve the performance of Laser. We last conduct extensive experimental studies on a wide range of large-scale high-dimensional vector datasets to verify the superiority of Laser. Specifically, Laser not only surpasses existing on-disk graph-based index systems but also matches or even exceeds the performance of in-memory index systems.

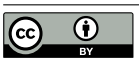
CCS Concepts: • **Information systems** → **Proximity search**; **Record and block layout**; **Search engine indexing**.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, High-dimensional Vector, Disk-based Graph Index

ACM Reference Format:
Weijian Chen, Haotian Liu, Yangshen Deng, Long Xiang, Liang Huang, and Bo Tang. 2026. Efficient Index Layout and Search Strategy for Large-scale High-dimensional Vector Similarity Search. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 168 (June 2026), 27 pages. https://doi.org/10.1145/3802045

*Dr. Bo Tang is the corresponding author.

Authors' Contact Information: Weijian Chen, Southern University of Science and Technology, Shenzhen, China, chenwj2024@mail.sustech.edu.cn; Haotian Liu, AlayaDB AI, Shenzhen, China, haotian.liu@alayadb.ai; Yangshen Deng, University of Edinburgh, Edinburgh, United Kingdom, yangshen.deng@ed.ac.uk; Long Xiang, AlayaDB AI, Shenzhen, China, long.xiang@alayadb.ai; Liang Huang, Southern University of Science and Technology, Shenzhen, China, huangl2025@mail.sustech.edu.cn; Bo Tang, Southern University of Science and Technology, Shenzhen, China, tangb3@sustech.edu.cn.



ACM 2836-6573/2026/6-ART168
https://doi.org/10.1145/3802045

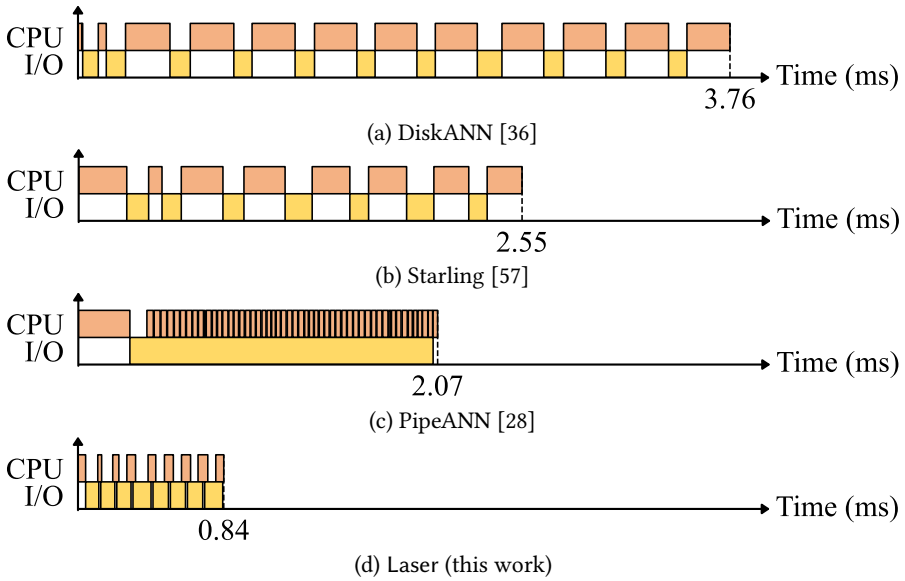


Fig. 1. Query latency profiling of on-disk graph-based index systems, Recall@10=0.90 on GIST1M (960D)

1 Introduction

Large-scale, high-dimensional vector similarity search is a core subroutine in many applications, including recommendation systems [17, 32, 44, 60], computer vision [8, 25, 49], Retrieval-Augmented Generation (RAG) [19, 39, 40, 52, 66], and inference engines [15, 18, 43] in Large Language Models. Due to the curse of dimensionality, exact search becomes prohibitively expensive in high-dimensional spaces. Consequently, approximate nearest neighbor search (ANNS) has emerged as the standard solution [9, 23, 36, 41, 45, 54, 56, 58, 62]. The performance of ANNS solutions is typically assessed along two complementary dimensions: (i) search efficiency, measured by latency and throughput (queries per second, QPS), and (ii) search accuracy, commonly quantified by recall. A wide range of techniques has been proposed to optimize this trade-off, including hashing-based, cluster-based, and graph-based indexing methods. Among these, in-memory graph-based index methods have emerged as the state-of-the-art approach, consistently achieving superior recall while maintaining low latency and high throughput [1, 5, 6, 20–22, 26, 29, 46, 47]. However, the rapid growth of vector datasets has exposed a critical bottleneck: the memory demands of in-memory graph-based index far exceed the capacity of a single machine. For example, SymphonyQG [26], despite its high search performance, encounters a hard memory constraint: it is unable to build its graph index for a 10.1 million 764-dimensional vector dataset (28.96 GB) within a single machine with 128 GB memory, as we will verify in Section 5.

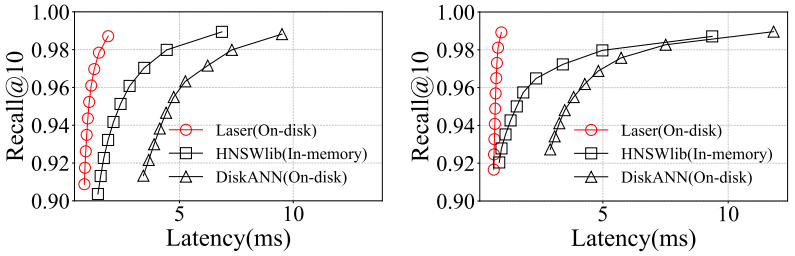
Intuitively, on-disk graph-based index systems [28, 36, 57] have been proposed to overcome the limitation of the prohibitive memory demands of in-memory graph-based indexes for efficient ANNS on large-scale high-dimensional vector dataset. The general idea of existing on-disk graph-based index systems is that they store the underlying graph-based index structure to cost-effective Solid-State Drives (SSDs), which provide high I/O throughput and low random read latency [30, 31]. In particular, DiskANN [36] is the first on-disk graph-based index system. It introduces a new on-disk graph-based ANN index Vamana, which enables DiskANN to minimize the number of disk I/O reads. Moreover, Vamana can be combined with vector compression schemes (e.g., product quantization), where the compressed vectors are cached in memory for fast and approximate

distance computation. It is now well-established that I/O latency constitutes the fundamental performance bottleneck in DiskANN as graph traversal invokes expensive disk I/O accesses [28, 57]. We profile the query latency of DiskANN on 960-dimensional GIST1M by setting Recall@10=0.90. Figure 1(a) shows the corresponding CPU computation costs and I/O costs from one of the profiled queries. To reduce the I/O costs, Starling [57] modifies the data layout of DiskANN via "block-shuffling" to improve on-disk data locality, and employs an in-memory navigation graph to shorten search paths. Comparing the profiling result of DiskANN in Figure 1(a), Starling successfully reduced the I/O costs to process the same query, see I/O cost in Figure 1(b). PipeANN [28], in contrast, redesigns the search strategy in DiskANN by replacing the synchronous beam search with an asynchronous pipeline, which overlaps CPU computation cost and I/O cost, then hides I/O latency, see Figure 1(c).

Interestingly, as shown in Figures 1(a), (b) and (c), we observed that the CPU costs contributed to a large proportion of the query latency in all existing on-disk graph-based index systems (i.e., DiskANN, Starling, and PipeANN). We further systematically validate the above observation with an adapted roofline model. Contrary to current wisdom, i.e., the performance of on-disk graph-based index systems is limited by I/O costs, our analyzed results from the roofline model reveal the performance of these systems changes from I/O-bound to compute-bound with the rising of the vector dimensionality. For example, the performance of DiskANN, Starling and PipeANN is actually I/O-bound on 96-dimensional DEEP10M, but it turned to compute-bound on 960-dimensional GIST1M, we will elaborate the details shortly in Section 2.2.

However, it is not trivial to overcome the compute-bound of existing on-disk graph-based index systems. The technical challenges are from two-fold: (i) sequential computation pattern, and (ii) precision-storage dilemma. In particular, the fast and approximate distance computation of DiskANN [36] relies on a sequential, step-by-step lookup table (LUT) accesses. Both Starling and PipeANN utilize the same mechanism of DiskANN. However, this computation procedure cannot be effectively parallelized via SIMD instructions in modern CPUs. Inspired by existing in-memory SIMD-optimized graph-based index systems (e.g., SymphonyQG [26]), a straight-forward way to exploit SIMD instruction in modern CPUs is appending the quantization compressed vectors to the end of the data layout in DiskANN. Unfortunately, it brings precision-storage dilemma, i.e., there is a tradeoff between the precision of compressed vector and the storage cost. Specifically, low precision quantization compressed vectors consume less space but lose search accuracy, and high precision ones maintain good search accuracy but consume more space, see the detailed analysis in Section 3.1.

In this work, we propose Laser, which consists of a novel index Layout and an optimized Search strategy, for efficient ANNS. In particular, a novel SIMD-friendly layout is devised in Laser to address the above two technical challenges. Each data vector will divide into two components: principal component and residual component via Principal Component Analysis (PCA). Only the quantization compressed vectors (we also refer it as quantized codewords) of principal component will be appended into the data layout, which addresses the precision-storage dilemma. Furthermore, the quantized codewords of all neighbors for each data vector are arranged in an interleaved fashion, facilitating efficient utilization of SIMD registers. Interestingly, the SIMD-friendly layout in Laser not only alleviates the compute-bound of on-disk graph-based index systems, but also unlocks a novel and unique opportunity for us to improve the search performance as the quantized codewords of data vectors do not need to store in memory cache any more and the available memory budget can be further utilized. Hence, we propose in-degree based node cache scheme to fully utilize the memory budget in Laser, which effectively reduces the number of I/O accesses and overlaps CPU computation costs and I/O costs, simultaneously. As shown in Figure 1(d), the CPU costs of Laser are significantly reduced due to the effectiveness of the new SIMD-friendly layout.



(a) GIST1M (960D) (b) Cohere (768D)

Fig. 2. Overall performance evaluation of Laser

We further accelerate the performance of Laser by optimizing its search strategy. First, we design cluster-based entry point selection method to shorten search paths. Second, an adaptive beam expansion strategy is employed to avoid wasteful node exploration during graph traversal. Third, we identify the root cause of the long-tail I/O latency of modern SSDs, which is overlooked in existing work, and devise an asynchronous early dispatch mechanism to handle slow I/O requests. The efficiency of the optimized search strategy in Laser is confirmed by the profiling result in Figure 1(d). As depicted in Figures 2(a) and (b), the Recall@10-latency curve of our Laser not only outperforms the representative on-disk index system DiskANN [36], but also surpasses the widely-used in-memory index HNSWlib [46] on both GIST1M and Cohere, respectively.

The major contributions of this work are summarized as follows:

- We reveal a fundamental paradigm shift in the performance characteristics of on-disk graph-based index systems. Contrary to the prevailing I/O-centric view, our adapted roofline model verifies that CPU computation, not disk I/O latency, has become the dominant bottleneck for large-scale, high-dimensional ANNS, a finding previously unrecognized in the literature.
- We address the emergent CPU bottleneck through a novel SIMD-friendly index layout in Laser. It not only resolves the technical challenges in on-disk graph-based index systems, but also enables an efficient in-degree node cache scheme.
- We design a suite of optimization techniques (e.g., cluster-based entry point selection, adaptive beam expansion, and early dispatch mechanism) on top of SIMD-friendly layout in Laser to further improve the search performance.
- We conduct comprehensive experiments to evaluate the superiority of Laser, which is substantially outperforms state-of-the-art on-disk graph-based index systems. Counterintuitively, the performance of Laser is comparable to or even slightly better than the leading in-memory ANNS systems (e.g., HNSWlib).

The remainder of this paper is organized as follows. Section 2 introduces the preliminaries and analyzes the performance bottleneck via the adapted roofline model. Section 3 presents the novel SIMD-friendly index layout of Laser and in-degree based node cache scheme. Section 4 elaborates on the optimizations of the search strategy in Laser. Section 5 conducts extensive experimental evaluations to verify the effectiveness of Laser. Sections 6 summarize the related work, and Section 7 makes a conclusion and highlights the promising future work directions.

2 Preliminaries and Analysis

Let $\mathcal{D}$ be a dataset of d-dimensional vectors. For a query vector q, vector similarity search identifies k vectors in $\mathcal{D}$ with the smallest Euclidean distances to q. In this work, we design an on-disk, graph-based index system that enables efficient ANNS on large-scale high-dimensional vector

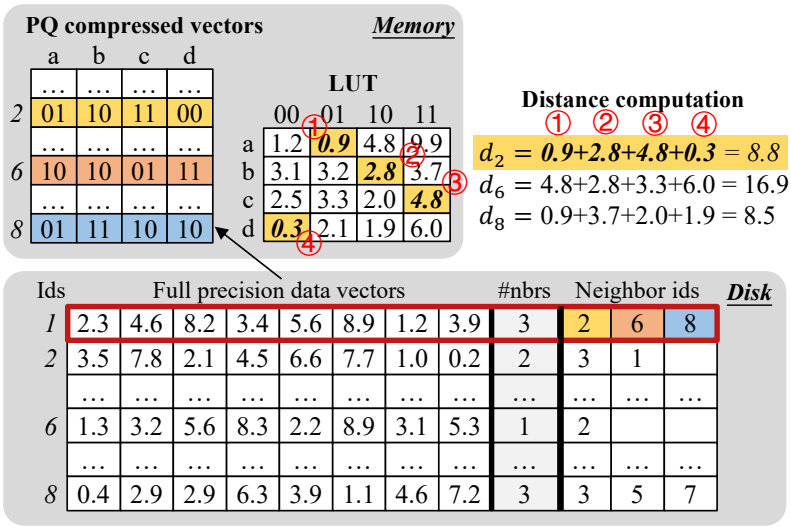


Fig. 3. Data layout and distance computation on DiskANN

dataset $\mathcal{D}$ with a limited memory budget B. Specifically, the size of vector dataset $\mathcal{D}$ vastly exceeds the available memory budget B. The objective of this work is to maximize search performance (i.e., high throughput and low latency), while maintaining good result accuracy (i.e., high recall) for high-dimensional vectors (e.g., those with hundreds or thousands of dimensions). In this section, we first present the general idea of existing on-disk graph-based index systems for high-dimensional vector similarity search in Section 2.1, then identify the performance bottleneck of these systems on two vector datasets with different dimensionalities via adapted roofline model in Section 2.2.

2.1 Existing On-disk Graph-based Indexes

DiskANN [36] is the first, also the *de facto* standard, on-disk graph-based index system that enables efficient ANNS over large-scale high-dimensional vector datasets. DiskANN consists of (i) the new graph-based ANNS index structure Vamana, and (ii) the off-the-shelf vector compression schemes (e.g., product quantization).

Data layout of DiskANN. DiskANN first constructs the new on-disk graph-based index Vamana, which initializes the index graph by randomly adding R out-neighbors for each data vector, then prunes and reconnects the edges among these vectors to ensure the quality of the graph-based index. The constructed full graph-based index along with the full-precision raw vectors are stored on SSD, as shown in Figure 3. DiskANN then uses product quantization (PQ) to compress raw vector and caches compressed vectors (a.k.a. quantized codewords) in memory. Taking the vectors in Figure 3 as an example, the 8-dimensional data vector space is divided into four 2-dimensional sub-vector spaces, we denote them as a, b, c, and d. For each sub-vector space, PQ clusters the corresponding 2-dimensional data vectors into 4 clusters by running 4-Means algorithm. Thus, each full precision data vector can be compressed into 8 bits quantized codewords via the PQ compression scheme. For example, the compressed vector 2 is an 8-bits string "01 10 11 00", where the first 2-dimensional of the data vector 2 is represented by "01" in the sub-vector space a.

Distance computation. For a given query q, DiskANN generates a lookup table (LUT) in memory to store the distances of query sub-vector and the cluster centroids of every data sub-vector space, see the in-memory LUT in Figure 3. The LUT can be used to obtain the approximate distance between query vector and candidate data vectors and guide the best vectors (and neighbors)

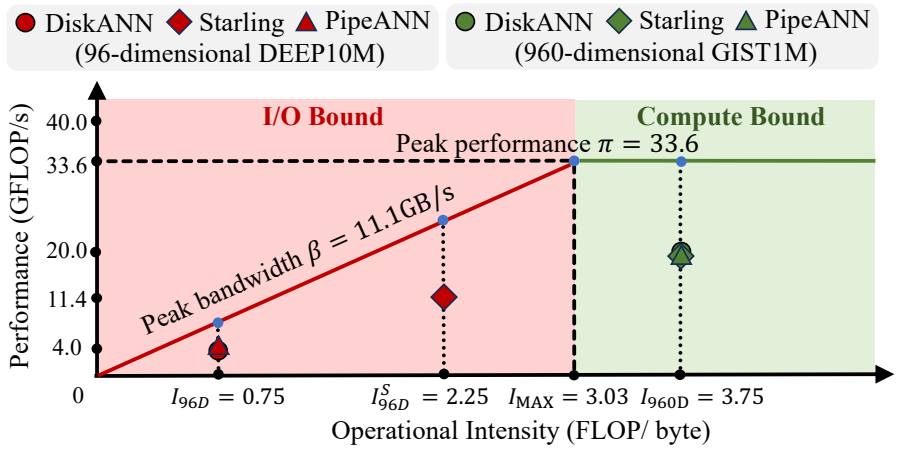


Fig. 4. Roofline model analysis

to read from disk. For example, the approximate distance between query q and data vector 2 can be computed by summing the values which are obtained by accessing ① to ④ of LUT. The approximate distance can be refined by computing the exact distance between the query vector and full precision data vectors, and the final results can be returned by re-ranking these candidates with exact distances.

Beam search strategy. To reduce the number of I/Os, DiskANN devises beam search strategy, which fetches the neighborhoods of a small number of the closest points (a.k.a., beam width BW) in one shot as reading a few random sectors from an SSD takes almost the same amount of time as reading one sector.

Recently, several work [28, 57] are built upon DiskANN to improve the search performance. In particular, Starling [57] introduced a block-shuffling technique to improve on-disk data locality. Moreover, it enhances the search strategy of DiskANN by utilizing an in-memory navigation graph to shorten search paths. The core idea of PipeANN [28] is replacing the synchronous beam search strategy by an asynchronous pipeline, which overlaps the computation cost and I/O cost, then hides the I/O latency.

2.2 Roofline Model Analysis

The roofline model [59] typically used to identify performance bottleneck of a system on a given hardware platform, e.g., whether it is compute-bound (limited by processor speed) or memory-bound (limited by memory bandwidth). Since we focus on on-disk graph-based index systems for ANNS, we adapt the classic roofline model to analyze whether the system is compute-bound or I/O-bound. In particular, in the adapted roofline model, the x-axis is the ratio of floating-point operations performed per byte read from the SSD (in FLOP/byte). The y-axis is the same as the classic roofline model, which shows the performance of the processor in GFLOP/s. Our experimental studies are conducted on a machine with a 24-core/48-thread CPU (Intel Xeon Gold 5318Y @ 2.10GHz) and NVMe SSDs (SAMUSUNG PM9A3). We evaluate the performance of three on-disk graph-based index systems for ANNS (i.e., DiskANN, Starling, and PipeANN) on two widely-used vector datasets: the 96-dimensional DEEP10M [64] and the 960-dimensional GIST1M [34]. Without loss of generality, all I/O reads in these systems are bypassing the effect of operating system caching.

Roofline model formulation. In the roofline model, there are two parameters: (i) the peak performance π and (ii) the peak bandwidth β of the given NVMe SSD, and one variable: the

operational intensity $\mathcal{I}$. We first analyze the peak performance of these three evaluated on-disk graph-based index systems on the tested machine. In particular, we derive it by investigating the operations of approximate distance computation in memory. As shown in Figure 3, the approximate distance between the given query q and each data vector i is computed by three steps: (i) reading PQ compressed bits in each sub-vector, (ii) looking up the corresponding distance from LUT, and (iii) accumulating these distance to the estimated approximate distance d_i. Suppose both the compressed vectors and LUT are in L1 cache, the total latency is 3 clock cycles (i.e., 2 L1 loads and 1 FP add). Thus, the peak throughput per thread is 2.1 GHz / 3 cycles = 0.7 GFLOP/s as the CPU frequency is 2.1 GHz. Consequently, the peak performance for the 24-core/48-thread CPU is π = 0.7 GFLOP/s × 48 = 33.6 GFLOP/s, as the compute roof shown in Figure 4. We next derive the *maximum operational intensity* $\mathcal{I}_{max}$ in the roofline model. The maximum I/O bandwidth of the used NVMe SSDs is the peak bandwidth β = 11.1 GB/s. The ridge point (where the diagonal and horizontal roof meet) defines the *maximum operational intensity* $\mathcal{I}_{max} = \pi/\beta$ = 33.6 GFLOP/s / 11.1 GB/s ≈ 3.03 FLOP/byte, see $\mathcal{I}_{max}$ in the x-axis of Figure 4. Thus, if the operational intensity of the evaluated system $\mathcal{I} < \mathcal{I}_{max}$, it is I/O-bound, see the left part in Figure 4. Conversely, if $\mathcal{I} > \mathcal{I}_{max}$, it is compute-bound, as the right part shown in Figure 4.

Operational intensity of on-disk graph-based indexes. We then analyze the operational intensity of existing on-disk graph-based index systems (i.e., DiskANN, Starling, PipeANN) on different datasets with the given hardware configuration on the used machine. In fact, the operational intensity $\mathcal{I}$ of an on-disk graph-based index system is determined by: $\mathcal{I} = \frac{\text{\# of FLOPs per disk page}}{\text{disk page size}}$.

DEEP10M: For 96-dimensional DEEP10M, each data vector is divided into 48 sub-vectors and each has at most 64 neighbors as out-neighbor R is 64. Processing a 4KB disk page in DiskANN and PipeANN for a node involves 64 × 48 = 3,072 FLOPs. Thus, the arithmetic intensity of DiskANN and PipeANN on 96-dimensional DEEP10M is $\mathcal{I}_{96D}$ = 3,072 FLOPs / 4096 bytes = 0.75 FLOP/byte, see $\mathcal{I}_{96D}$ at x-axis of Figure 4. The block-shuffling technique in Starling [57] enables it processes 3 nodes per disk page on DEEP10M. Thus, the arithmetic intensity of Starling on DEEP10M is $\mathcal{I}^S_{96D}$ = 3,072 x 3 FLOPs / 4096 bytes = 2.25 FLOP/byte, as shown in Figure 4. All three on-disk graph-based index systems are I/O-bound at 96-dimensional DEEP10M as both $\mathcal{I}_{96D} < \mathcal{I}_{max}$ and $\mathcal{I}^S_{96D} < \mathcal{I}_{max}$.

GIST1M: For 960-dimensional GIST1M, each data vector is divided into 480 sub-vectors and each has at most 64 neighbors. Processing an 8KB disk page for a node and its neighbors requires 64 × 480 = 30,720 FLOPs. Thus, the arithmetic intensity of all three solutions (i.e., DiskANN, Starling, and PipeANN) on 960-dimensional GIST1M is $\mathcal{I}_{960D}$ = 30,720 FLOPs / 8192 bytes ≈ 3.75 FLOP/Byte. The reason why the arithmetic intensity of Starling is the same as its of DiskANN and PipeANN on GIST1M as each 8KB disk page only can store 1 data node with its neighbors in all these three on-disk graph-based index systems. All these solutions are compute-bound at 960-dimensional GIST1M as $\mathcal{I}_{960D} > \mathcal{I}_{max}$.

Achieved performance evaluation. We measure the achieved performance of these three on-disk graph-based index systems on two vector datasets with the used machine. We use all 48 available threads to maximize the system utilization. As depicted in Figure 4, the achieved performance of DiskANN, Starling and PipeANN on 96-dimensional DEEP10M are 3.96 GFLOP/s, 11.37 GFLOP/s and 5.09 GFLOP/s, respectively. On DEEP10M, the performance of DiskANN and Starling is limited by the actual SSD bandwidth as it only achieved 6.5 GB/s, which is far below its peak bandwidth β = 11.1 GB/s. The reason is that the beam search width BW is configured as 8, which limits the depth of I/O queue in them [31]. PipeANN achieves a higher performance than DiskANN by employing a deeper I/O queue (i.e., 32) in its pipeline, which utilizes the available SSD bandwidth better. On 960-dimensional GIST1M, the achieved performance of DiskANN, Starling and PipeANN are 18.94

Table 1. Summary of on-disk graph-based index systems

Systems	Computational Efficiency	I/O Efficiency	Search Performance
DiskANN [36]	Low	Median	Low
Starling [57]	Low	High	Median
PipeANN [28]	Low	High	Median
Laser (this work)	**High**	**High**	**High**

GFLOP/s, 18.85 GFLOP/s, and 18.28 GFLOP/s, respectively. All of these solutions are significantly below the peak performance π = 33.6 GFLOP/s. The reason is that the peak performance is derived at the ideal situation, i.e., it only takes 3 cycles for each FLOP as it assumes all the data are in L1 cache. However, the PQ compressed vector of each data vector in 960-dimensional GIST1M is impossible to entirely store on L1 cache, which means it takes more CPU cycles for each FLOP then the peak performance π will be smaller than the maximum 33.6 GFLOP/s in real-world systems.

Summary of the roofline model analysis. As depicted in Figure 4, all three on-disk graph-based index systems are I/O-bound when the vector dataset has moderate dimensionality, e.g., several tens to a hundred dimensions. However, the performance bottleneck of on-disk graph-based index systems for efficient ANNS fundamentally shifts from I/O-bound to compute-bound with the rising of vector dimensionality (e.g., several hundreds or thousands). For example, all existing on-disk graph-based index systems are compute-bound on 960-dimensional GIST1M.

Table 1 summarizes the existing on-disk graph-based index systems from three aspects: (i) computational efficiency, (ii) I/O efficiency, and (iii) search performance. The computational efficiency of all existing solutions is not good enough as all of them are optimizing the I/O bottlenecks and overlooking the computation bottleneck. The I/O efficiency of Starling and PipeANN is better than DiskANN as they improve it by devising specific optimization techniques. Combing the Computational efficiency and I/O efficiency of existing systems, it is safe to conclude that this is a substantial performance improvement space in on-disk graph-based index systems. Hence, the research objective of this work is proposing an efficient on-disk graph-based index system, which achieves high efficiency on all three aspects for large-scale high-dimensional vector retrieval, see the last row in Table 1.

3 Index Layout of Laser

As confirmed by the adapted roofline model in Section 2, the performance bottleneck of existing on-disk graph-based index systems is the expensive computational costs. In this section, we devise an effective index layout in Laser to overcome it. However, it is not trivial to achieve it due to the technical challenges in Section 3.1. We next design the SIMD-friendly layout of Laser in Section 3.2, and present the node caching scheme of Laser in Section 3.3.

3.1 Technical Challenges

We analyze the technical challenges to overcome the compute-bound of existing on-disk graph-based index systems as follows.

C1: Limitation of sequential computation pattern. As discussed in Section 2.2, the computation cost of existing on-disk graph-based index systems (i.e., DiskANN, Starling, and PipeANN) is dominated by the approximate distance calculation from the quantization codewords. In particular, it incurs numerous LUT lookups. Taking Figure 3 as an example, it consumes 12 LUT lookups to derive the approximate distances from the query q to the neighbors of data vector 1. Although LUT is typically small enough to fit within the L1 cache of CPU, which allows low-latency lookups,

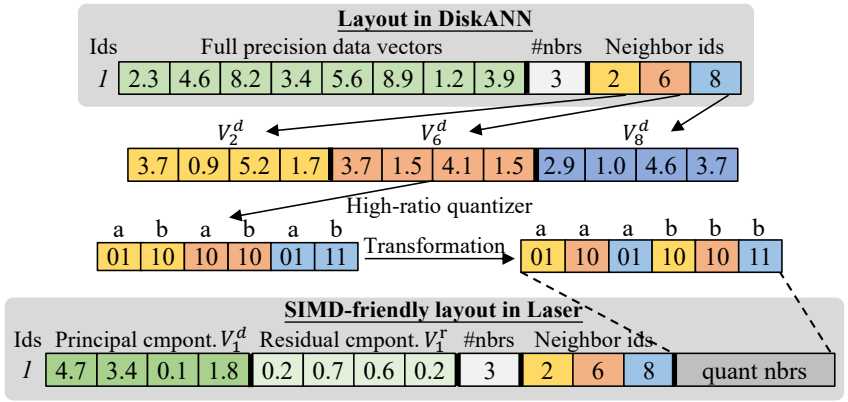


Fig. 5. SIMD-friendly layout in Laser

the lookup process must proceed step by step, with each stage dependent on the completion of the prior one. For example, as shown in Figure 3, 0.9, 2.8, 4.8 and 0.3 are accessed by looking up the LUT step by step then used to estimate the approximate distance between query vector q and data vector 2. Thus, the above computation procedure cannot be effectively parallelized and executed by exploiting SIMD instructions in modern CPUs due to the irregular memory access and inefficient vectorization. Thus, it poses the first technical challenge to overcome the compute-bound of existing on-disk graph-based index systems.

C2: The precision-storage dilemma. Existing in-memory index systems [26, 63] alleviate the computation performance by arranging the quantization codewords in a contiguous, interleaved manner. Thus, it can invoke a single SIMD instruction [27] to perform multiple LUT lookups in parallel. Inspired by these in-memory systems, a straight-forward idea is revising the index layout of DiskANN by appending the quantization codewords vectors of the neighbors after the neighbor ids. However, the core technical challenge in this idea is the precision-storage dilemma as the size of quantization codewords affects the on-disk index size. In particular, if a high-precision compression scheme is applied on the vector dataset and the corresponding quantization codewords are stored to maintain the search accuracy, the size of on-disk graph-based index grows quickly. Moreover, the disk page also will be extremely large as each data vector has tens of neighbors and the quantization codewords of all its neighbors should be contiguously stored in the same page. As a result, the overall performance will be significant degenerated as the bandwidth of SSD is limited. Conversely, aggressively compressing the data vectors into short codewords preserves the small page size, but it fails to maintain the precision of quantization codewords. Thus, a higher number of nodes will be visited via I/O accesses during the search procedure to achieve the high recall, which increases the end-to-end latency.

3.2 SIMD-friendly Layout of Laser

With the discussed technical challenges in Section 3.1, the research problem is how to design a new index layout with limited page size for on-disk graph-based index systems, which enables parallel approximate distance calculation via SIMD instructions on CPU. We address it by devising an SIMD-friendly layout in Laser.

Design of SIMD-friendly layout in Laser. The overview of the SIMD-friendly layout in Laser is illustrated at the bottom of Figure 5, we introduce the designs of it as follows. The key insight behind the design is that we observed that the quantization scheme (e.g., PQ) treats all dimensions

in data vector uniformly and allocates equivalent bits to different dimensions, which results in suboptimal disk page utilization. To overcome this, we first utilize Principal Component Analysis (PCA) to project the original high-dimensional data vectors into a low-dimensional subspace that captures the majority of the data variance. In particular, the Principal Component Analysis (PCA) transformer will generate two parts for each data vector i: a principal component V_i^d, and a residual component V_i^r, the exact distance between the query q and data vector i is preserved when the residual component is also included [7, 10, 38]. Second, we only apply a high-ratio quantizer (e.g., Product Quantization (PQ) [37] or RaBitQ [24]) on the principal component of the neighboring data vectors. For example, the neighboring principal components of data vector 1 are V_2^d, V_6^d and V_8^d. They are compressed via a high-ratio quantizer [24]. We obtain the highly compact quantized codewords, as shown in Figure 5. To minimize the quantization error, we apply an orthogonal rotation to balance the variance distribution across the dimensions of principal components, which allows the quantizer to achieve higher precision for a given code size. To fully exploit the SIMD instructions on CPU, we re-arrange these quantized codewords in an interleaved manner. In particular, we decompose the quantized codeword of each data vector into individual sub-codes, then sequentially store the same position sub-codes of different neighbors. As illustrated in Figure 5, the sub-codes of "a" and "b" from the neighboring data vectors 2, 6, 8 are stored adjacently after the rearrangement transformation. Last, the compact and rearranged quantized codewords of the neighbors are appended.

To sum up, in SIMD-friendly layout, the principal component V_i^d and a residual component V_i^r of a data vector are utilized to rerank and refine the final result set of query vector. The neighbor IDs are used to traverse the graph, and the quantized codewords of neighbors are used to estimate the distance between the query vector and each neighboring data vector. Comparing to the index layout of DiskANN, Laser slightly introduces extra storage overhead on disk to store the quantized codewords of neighbors and removes the PQ compressed vectors from memory.

Determining the graph out-degree R: The graph out-degree R directly determines the disk page size in Laser. The above SIMD-friendly layout of Laser enables the using of the commonly-used out-degrees (e.g., $R = 64$ or 96) in existing on-disk graph-based index systems (e.g., DiskANN) while keeping small page size. The core reason is that Laser only stores the quantization codewords of the principal component V_i^d, instead of the original data vector. For example, on 960-dimensional GIST1M, the page size of Laser is 8KB for out-degree $R = 64$ and it also is only 12KB if the out-degree is $R = 128$. It confirms the design of our SIMD-friendly layout ensures the high I/O performance (i.e., IOPS). In addition, the performance gains diminish rapidly with the increase of graph out-degree R [22]. Thus, the out-degree R does not exceed 128 in Laser.

Discussions: The SIMD-friendly layout used Principal Component Analysis (PCA) to analyze the principal components on data vectors. It is suitable for the query vectors with the similar distribution of data vectors. However, the performance of the proposed SIMD-friendly data layout will be degenerate when processing out-of-distribution queries. We left how to efficiently process out-of-distribution query with on-disk graph-based index system as future work as it is the well-known cross-modal ANNS problem [12], which differs from the ANNS problem in this work as there is a 'modality gap' between the data vectors and query vectors.

Running example: The example in Figure 6 illustrates the distance computation procedure with SIMD-friendly layout. In particular, it loads the quantized codewords of the same position from different neighbors into a SIMD register. For example, it first loads the sub-codes "a" from neighboring data vectors 2, 6, 8 of data vector 1. Then, the corresponding LUT for the sub-codes "a" is loaded into another SIMD register. After that, the distances between the each sub-code of the

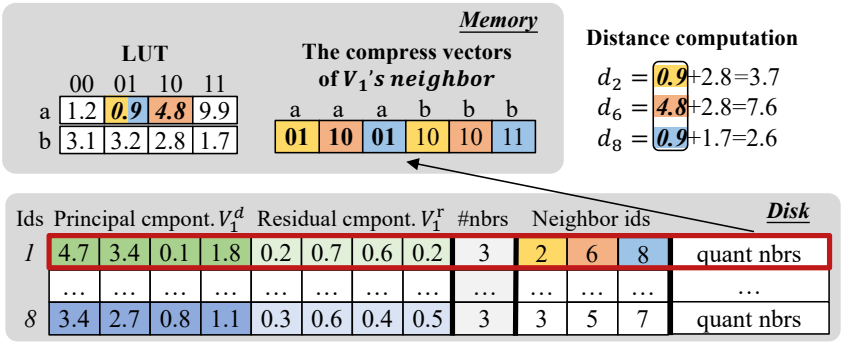


Fig. 6. Distance computation in Laser

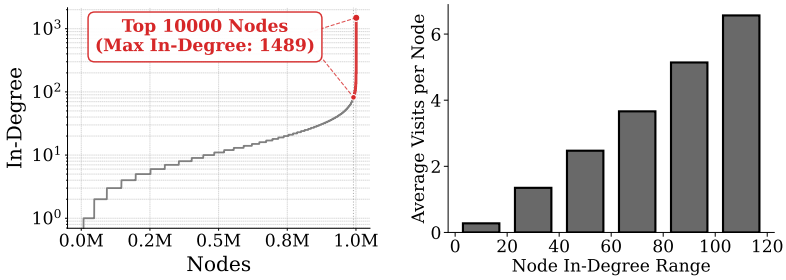
Table 2. Performance analysis of the SIMD-friendly layout

Metric	DiskANN layout	Laser layout
Search Latency (1 thread)	3223.66 μs	2036.44 μs
Computation Cost	1846.55 μs	220.833 μs
Mean I/Os per Query	77.21	131.109
I/O Cost	1254.09 μs	1774.51 μs
Throughput (48 threads)	7652.19	8763.89

query and vectors 2, 6, 8 are obtained via applying SIMD shuffle instruction [27] on these two SIMD registers, which are 0.9, 4.8 and 0.9, respectively, as depicted by colors in LUT "a". Then applying the same procedure on the sub-codes "b", and compute the approximate distance between query vector and data vectors 2, 6 and 8. Compared to DiskANN, Laser transforms the strict serial LUT accesses into a parallel execution procedure via the SIMD-friendly layout, which significantly reduces the computational cost. In particular, as shown in Figure 3, DiskANN computes the approximate distances among the neighboring data vectors 2, 6 and 8 of data vector 1 with query vector q by invoking 24 scalar loads and 12 adds [11]. However, our Laser computes these approximate distances by 4 SIMD loads, 2 SIMD shuffles and 2 SIMD adds. It is worth pointing out that Laser works on the quantized codewords of principal component of data vector, which is smaller than the PQ compressed vectors in DiskANN. Thus, the computation cost of Laser will be further reduced.

3.3 Node Caching Scheme in Laser

Effect of Laser index layout. To investigate the effect of the proposed SIMD-friendly layout in Laser, we measured the performance of DiskANN and a modified version of DiskANN, i.e., it replaces the index layout of DiskANN by our proposed SIMD-friendly layout in Section 3.2 on 960-dimensional GIST1M at Recall@10=0.90. The search algorithm and all other parameters (i.e., the out-degree $R = 64$ and beam search width $BW = 8$) are the same in this experiment. As shown in Table 2, the SIMD-friendly layout yielded a significant 88% reduction of the computation cost, which confirms its effectiveness to resolve the compute-bound of DiskANN. However, the computational efficiency improvement from the SIMD-friendly layout also introduces extra overhead. The 960-dimensional data vector in GIST1M is quantized into a 48-byte codeword in the SIMD-friendly layout. The high compression ratio losses the precision of data vectors, which in turn requires visiting more nodes to guarantee the same recall, i.e., Recall@10=0.90. Thus, the mean I/Os per



(a) The distribution of nodes (b) The number of node visits

Fig. 7. In-degree of node analysis

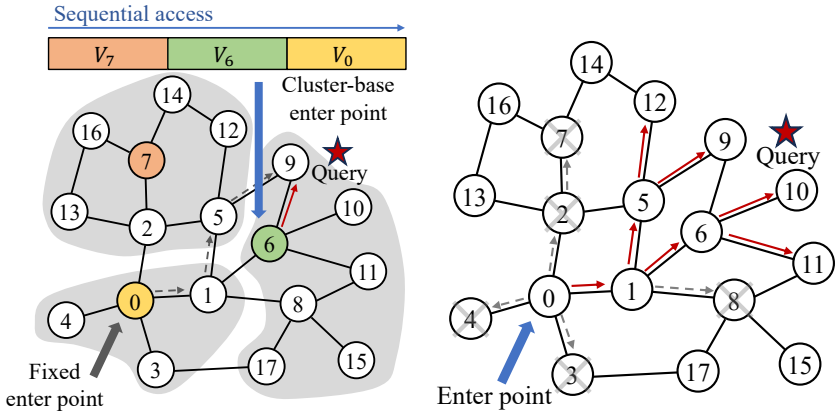
query of the modified DiskANN with SIMD-friendly layout is higher than the original DiskANN. Taking both compute cost and mean I/Os per query (resp. I/O cost in Table 2) into consideration, the corresponding search latency and throughput of the DiskANN with the proposed SIMD-friendly layout only improve 37% and 14.5%, respectively. Thus, our new research question is: how to reduce the mean I/Os per query then fully unlock the potential of the SIMD-friendly layout in Laser?

In-degree based node cache scheme. Unlike DiskANN, the Laser index layout does not store compressed vectors in memory as the quantized codewords of neighboring data vectors are stored on the SIMD-friendly index layout, which brings a unique opportunity of Laser to utilize the available memory budget for search performance improvement. We designed an in-degree based node cache scheme on Laser by the following key observation. In particular, we observed that the high in-degree nodes in the graph-based index are visited more frequent than those with low in-degrees. Figure 7(a) illustrates the highly skewed in-degree distribution of the constructed graph-based index of DiskANN on 960-dimensional GIST1M. Figure 7(b) confirms the strong positive correlation between the in-degree of a node and its average number of visits during the search process. Thus, the in-degree based node cache scheme in Laser first computes in-degree of node in the graph-based index and then caches the top-ranked nodes up to the available memory budget. The performance gain becomes larger with the rising of the available memory budget. we use 80% of the memory budget for the node caching in Laser and the rest 20% are reserved to handle unexpected out-of-memory failures. The cached nodes can be periodically updated when the underlying graph-based index are updated by insertions and deletions. One side-benefit of our cache scheme is that it overlaps CPU costs and I/O costs in Laser. For example, consider a search hop that visits 5 nodes: V_1, V_2, V_3, V_4, V_5, where nodes $\{V_1, V_5\}$ are in the node cache and nodes $\{ V_2, V_3, V_4 \}$ are not in the cache. Laser overlaps the compute cost and I/O cost by executing the following tasks concurrently: (i) issuing asynchronous I/O requests to fetch nodes $\{ V_2, V_3, V_4 \}$; and (ii) computing the distance between the cached data vectors $\{ V_1, V_5 \}$ and query.

Until now, it is safe to conclude that the computational improvement brought by the SIMD-friendly layout in Laser changes the game again! Theoretically, the peak performance π in roofline model increases with the SIMD-friendly layout as the cycles to process a FLOP during approximate distance computation will be less than 3 cycles. The maximum operational intensity $\mathcal{I}_{max} = \pi/\beta$, which also will become larger with the improved of peak performance π. When $\mathcal{I}_{max} > \mathcal{I}_{960D}$ (see Figure 4), these on-disk graph-based index systems turn to I/O-bound again.

4 Search Strategy in Laser

To alleviate the re-emergent I/O bound of Laser with SIMD-friendly layout, we design a suite of optimizations to improve the search strategy in it, including (i) cluster-based entry point selection in



(a) Cluster-based EP selection (b) Adaptive beam expansion

Fig. 8. Search strategy optimization techniques in Laser

Section 4.1, (ii) adaptive beam expansion strategy in Section 4.2, and (iii) early dispatch mechanism in Section 4.3.

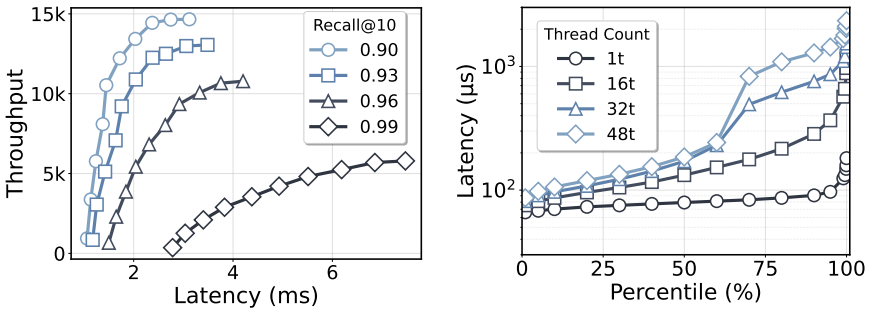
4.1 Cluster-based Entry Point Selection

Conventional graph-based search strategy initiates graph traversal from a fixed or randomly selected entry point to locate the nearest data vectors for the given query. However, the initial entry point often be distant from the actual location of query in the vector space, which results to a long search path. To address it, we propose a cluster-based entry point selection strategy in Laser. The core idea is to identify a region which is close to the query, then commence the search from an entry point in that region. As illustrated in Figure 8(a), during the index building phase, we employ k-means clustering to partition the set of base vectors into several regions, see these three shaded areas in Figure 8(a). The centroid vector of each region is selected as a candidate entry point, and these candidates are stored sequentially in an array, e.g., vectors 0, 7, and 6. During the search phase, for a given query vector, Laser sequentially scans these candidate entry points and computes their corresponding distances to the query. The candidate data vector with the smallest distance to the query is then used as the actual entry point for the graph traversal. The above idea ensures the search procedure begins in a region which is close to the query, then it significantly shortens the search path.

Our cluster-based entry point selection method in Laser differs from existing subgraph-based method [57] by our clustering-based method is “distribution-aware" as it learns the topology of data to maintain a small-yet-effective entry point candidates. It achieves comparable quality to existing methods which sample a large number of starting points randomly. Moreover, the online part of Laser is a lightweight sequential scan of all candidate entry points. It is faster than the costly in-memory graph traversal of subgraph-based method for every query.

4.2 Adaptive Beam Expansion Strategy

Laser follows the beam search strategy of DiskANN to reduce the I/O times during graph traversal. The core of beam search is exploring a batch of nodes at each hop, and the batch size is determined by the beam width BW. Generally, a larger beam width BW increases the likelihood to find a shorter search path, but incurs higher computational and I/O overhead. Existing on-disk graph-based index systems (i.e., DiskANN and Starling) typically configure a fixed beam width BW throughout the



(a) Throughput-latency curve (b) CDF of SSD random read latency

Fig. 9. Long-tail I/O on used SSD

entire search process, which is inefficient as it cannot adapt to the dynamic context of the search. Specifically, in the early stage of search processing, the query vector is far away from the visited nodes in the graph-based index, the fixed wide beam width may waste cost as it visits many unqualified nodes, i.e., which are not be the final result set, by "over expansion", see the gray dashed arrows in Figure 8(b). However, in the final stage of search processing, the fixed beam width BW may lead to insufficient exploration as the search procedure enters a dense region of candidates, which leads to long search path, consequently, results to higher search latency. Inspired by [50], we devise an adaptive beam expansion strategy in Laser to overcome this issue. In particular, the beam width BW at the i-th hop is determined by $min(2^i, BW_D)$, where BW_D is the default beam width and it is 2-4 times larger than that of DiskANN. The core idea of this method is enabling coarse-to-fine search: a narrow initial beam reduces wasteful I/Os in distant regions, while the exponential growth (2^i) enables a rapid transition to intensive exploration when the search converges to the targets, see the red arrows in Figure 8(b).

4.3 Early Dispatch Mechanism

Until now, the total number of I/O operations per query is reduced in Laser. However, we observe that the search latency for a given target recall increases with the rising of concurrency level.

Figure 9(a) shows the throughput-latency curves of Laser on 960-dimensional GIST1M. It is obvious the search latency of Recall10=0.99 is significantly higher than that of Recall10=0.90 with the same throughput. Interestingly, this problem is overlooked by almost all (if not all) existing work. The core reason is that there is a fundamental mismatch between the synchronous nature of the beam search algorithm and the inherent latency variance of modern SSD. At each search hop, the search procedure waits for all I/O requests in the current beam to complete. Consequently, the latency of each step is not determined by the average I/O time, but by the slowest I/O request in the batch, i.e., the long-tail I/O latency. Therefore, the slowest I/O access stalls the entire query, which enlarges the end-to-end search latency. For example, as shown in Figure 9(b), there is a significant long-tail distribution of the search latency. To make the matter worse, the higher concurrency level, the longer search latency. Specifically, with 48 threads, the 99.9th percentile latency is 2.0ms, which is 10.8x higher than the median latency (i.e., 185.3µs). We next design a search process that is robust to the inherent latency variance of modern SSD.

The key idea is leveraging the structural property of Vamana graphs: Path Redundancy. By maintaining a large out-degree (e.g., out-degree R = 64 or 96), the graph index has a high degree of path redundancy. Within any local region of the graph, the neighbor lists of adjacent nodes overlap significantly. This dense connectivity provides multiple alternative paths to the target nearest

Table 3. Vector dataset statistics

Dataset $\mathcal{D}$	$\lvert\mathcal{D}\rvert$	d	Data size	Query size
GIST1M [34]	1,000,000	960	3.57GB	1,000
Cohere [16]	10,123,929	768	28.96GB	10,000
BigCode [4]	10,403,628	768	28.98GB	10,000
DPR [1]	101,000,000	768	288.96GB	10,000
MSMARCO [16]	113,520,750	1,024	433.04GB	1,677

neighbors, thus, the search process is more robust. Crucially, a high-quality search direction on the graph can be obtained by using a large-enough subset of the neighbors. Thus, it does not need to wait for the long tail I/O access in the current beam search. Hence, Laser decouples the search process from the slowest I/O accesses. In particular, instead of synchronous waiting, it employs an asynchronous processing and early dispatch mechanism. At each hop, Laser continuously polls for completed I/O requests. As soon as a request finishes, it immediately begins the distance computations for the neighbors. The key idea is that Laser does not wait for all beam width nodes to be processed. Instead, after a predetermined ratio (e.g., 0.5, targeting the median latency) of the fast-responding nodes have been processed, it decides the next hop and dispatches the next batch of I/O requests immediately. Fortunately, this mechanism also overlaps CPU and I/O costs effectively.

Example. considering a search hop which is accessing 4 nodes V_1, V_2, V_3, V_4 on disk. When any node (e.g., V_1) arrives, its computation begins immediately and—overlaps the ongoing I/O for other nodes (e.g., V_2, V_3, V_4). Once a predetermined ratio of nodes are processed (i.e., 0.5, it processed V_1 and V_2), it then submits I/O requests for the next hop candidates V_5, V_6, V_7, V_8. Consequently, the computation for the late-arriving V_3 and V_4 runs asynchronously, which overlaps their CPU costs with the I/O cost of fetching V_5, V_6, V_7, V_8 effectively. Finally, the results from these deferred nodes V_3, V_4 are used to update the candidate set asynchronously, and maintain the same accuracy of the search procedure.

5 Experimental Evaluation

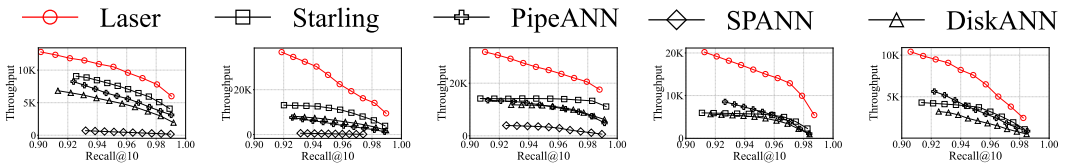
In this section, we first introduce the experimental setup in Section 5.1, then evaluate the overall performance in Section 5.2, and conduct the effectiveness study in Section 5.3.

5.1 Experimental Setup

Platform. All experiments in this work are conducted on a server with the following configurations:

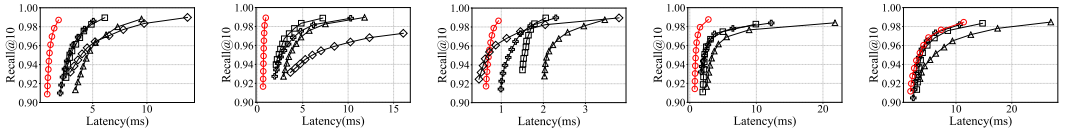
- CPU: Intel(R) Xeon(R) Gold 5318Y @ 2.10GHz, 24 cores/48 threads.
- Memory: 128 GB DDR4 @ 2933 MT/s (2 x 64 GB)
- Storage: 3 x 1.92 TB SAMSUNG PM9A3 NVMe SSDs
- Operating System: Ubuntu 22.04 LTS with Linux kernel 5.15.0

Datasets and query sets. We used 5 high-dimensional vector datasets in the experiments. Table 3 provides the detailed statistics of them. All of these 5 datasets are publicly available at HuggingFace [33] and BIGANN Benchmarks [53]. For the query set of each dataset, we directly used the provided query set for GIST1M and MSMARCO. For the rest three datasets, we randomly sampled 10,000 queries from the corresponding dataset to form the query sets as they do not include the query sets. We obtain the ground truth of the queries in each dataset via linear scan method.



(a) GIST1M (b) Cohere (c) BigCode (d) DPR (e) MSMARCO

Fig. 10. Throughput-Recall@10 evaluation on all on-disk index systems



(a) GIST1M (b) Cohere (c) BigCode (d) DPR (e) MSMARCO

Fig. 11. Recall@10-Latency evaluation on all on-disk index systems

Compared methods. We compare our proposed Laser with two categories of competitors: (i) on-disk systems, and (ii) in-memory solutions, which are introduced as follows:

On-disk systems. We evaluate 4 representative on-disk systems: 3 graph-based index systems and 1 cluster-based index system.

- **DiskANN** [36] is the first on-disk graph-based index system that stores the full graph structure and vectors on SSD, maintains PQ compressed vectors in memory to guide the search.
- **Starling** [57] improves the I/O efficiency of DiskANN by optimizing its on-disk data layout via "block shuffling" and using an in-memory navigation graph to shorten the search path.
- **PipeANN** [28] is an algorithmic optimization that pipelines I/O and computation of DiskANN during the execution, which hides the disk latency and reduces the query latency.
- **SPANN** [14] is the cluster-based index, which stores cluster centroids in memory and posting lists with vectors on disk.

In-memory solutions. We compare the performance of our Laser against 3 in-memory index solutions.

- **HNSWlib** [46] is a widely adopted in-memory graph-based index, which is a *de facto* in-memory solution for ANNS.
- **Vamana** [36] is the underlying graph structure of DiskANN, we evaluate its in-memory performance by storing the entire Vamana graph of DiskANN in memory.
- **SymphonyQG** [26] is the state-of-the-art in-memory graph-based index for ANNS. It leverages a compact layout to minimize random memory accesses and exploits SIMD instructions to accelerate distance computations.

Parameter configurations. The source code of the compared methods are obtained from the original authors. All the parameters of the evaluated methods are configured by following the settings in existing work [14, 28, 36, 57]. The memory budget $\mathcal{B}$ of all these on-disk systems are set as follows: 1GB for GIST1M, 10GB for Cohere and BigCode, and 128GB for DPR and MSMARCO. To bypass the effect of operating system cache, we use the libaio library with the O_RDONLY and O_DIRECT flags on all on-disk methods as the setting in [14, 28, 36, 57]. For all graph-based index

construction, the maximum out-degree is $R = 64$ and the construction list size is 200. For entry point selection, Starling and PipeANN constructed an in-memory navigation graph from 1% data samples, which is built with maximum out-degree $R = 48$ and the construction list size 128. In Laser, we store 300 candidate entry points for GIST1M, Cohere and BigCode, and 500 for DPR and MSMARCO by default. The dimension of the principal components in Laser for all five tested datasets is 256. During the search phase, the search beam width is 8, 8, 32 and 16 for DiskANN, Starling, PipeANN, and Laser, respectively. We vary the size of the candidate list (i.e., search_L) for all compared graph-based methods to achieve the targeted recall.

Measured metrics. We measure both search performance and result accuracy on all compared methods. In particular, the search performance metrics include: throughput (i.e., queries per second, QPS) and latency (time cost to process a query). In all our experiments, the throughput of a method is measured by using all available threads (i.e., 48 threads) and the latency is measured on a single thread. We also report the mean I/Os per query, which measures the average number of disk I/O operations to process a query. The result accuracy is measured by Recall@K, which shows the fraction of the true K-nearest neighbors found within the top-K results returned by the algorithm. It is a standard metric for ANNS.

5.2 Overall Performance Evaluation

We plot (i) throughput-recall@K curve (where top-right is better), and (ii) recall@K-latency curve (where top-left is better) to evaluate the overall search performance of all compared systems.

Compared with on-disk systems. Figure 10 depicts the throughput-recall@10 curves of all compared on-disk index systems on 5 tested datasets. Visually, our proposal Laser outperforms all existing on-disk systems at all tested cases. The throughput of Laser is 2.0x-3.6x, 1.4x-2.7x, 1.6x-4.2x, 7.9x-60.6x faster than DiskANN, Starling, PipeANN and SPANN among all these tested datasets, which confirmed the superiority of the designs in Laser. In particular, SPANN has the lowest throughput as it is cluster-based index, which needs to access a large amount of data vectors to achieve a high recall. We ignored SPANN at DPR and MSMARCO, see Figures 10(d) and (e), as the index construction of SPANN is out-of-memory at large vector datasets. Starling outperforms DiskANN as it employs an in-memory navigation graph to shorten the search path and reduce overall computation. PipeANN performs better than DiskANN as it effectively hides the latency in DiskANN via asynchronous I/O. However, it is slightly worse than Starling in some cases. The major reason is that the PIPESearch strategy in PipeANN may visit more nodes when comparing with the visited nodes in Starling.

Figure 11 shows the recall@10-latency curves of all evaluated on-disk index systems. It is no doubt that the latency of Laser is the overall winner among all these tested systems on all 5 datasets. In particular, the search latency of Laser is up to 91% lower than that of DiskANN on Cohere where recall@10 is 0.99. In all these tested cases, the search latency of Laser is approximately 24-70% and 22-74% lower than that of Starling and PipeANN, respectively. The performance gain of Laser are from three-fold: (i) the SIMD-friendly layout reduced the approximate distance computation time in memory, (ii) the in-degree based node cache scheme avoids I/O accesses as the frequently accessed nodes are cached in memory, and (iii) the optimization techniques in search strategy of Laser, e.g., the asynchronous early dispatch strategy mitigates the effect of long-tail I/Os. Besides, the search latency of PipeANN is comparable to Starling as it overlaps computation and I/O cost.

Compared with in-memory solutions. Even though Laser is an on-disk graph-based index system, we compare the performance of Laser with existing in-memory graph-based index methods in Figure 12. We ignore two large vector datasets (i.e., DPR and MSMARCO) in this experiment as their data sizes exceed the memory size of our machine (128GB). First of all, as shown in

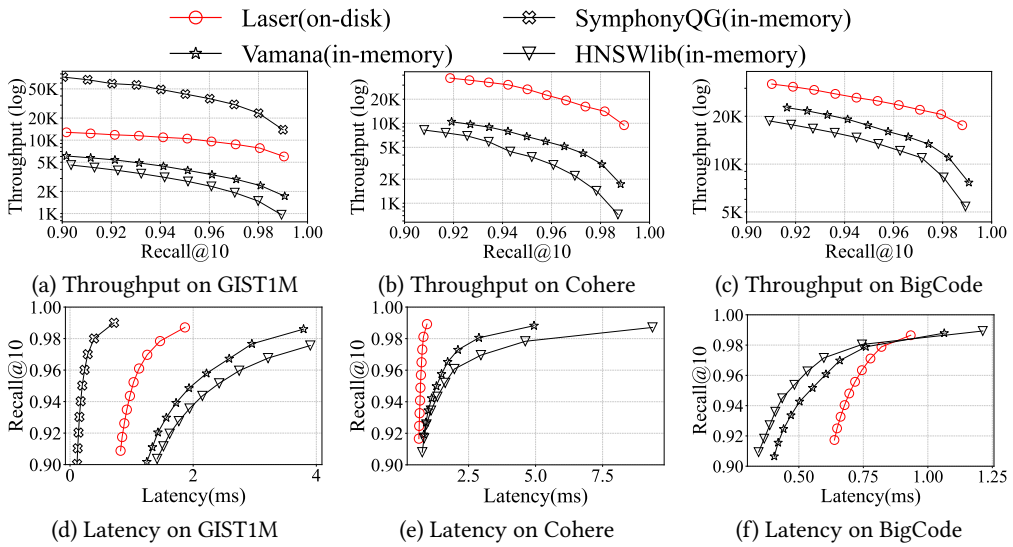
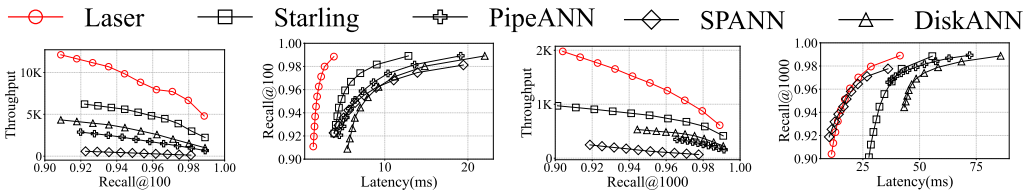


Fig. 12. In-memory index solutions evaluation

Figures 12(a), (b) and (c), the throughput-recall@10 of Vamana (in-memory) and HNSWlib (in-memory) are obviously below the corresponding curve of Laser (on-disk). It confirms that Laser not only outperforms its own in-memory graph-based index version, i.e., Vamana (in-memory), but also the widely-used in-memory index HNSWlib (in-memory) on these three datasets. Secondly, the state-of-the-art in-memory index SymphonyQG [26] is better than Laser on GIST1M as it is careful-designed for in-memory scenario. However, the index construction of SymphonyQG is failed on Cohere (29GB) and BigCode (29GB) due to out-of-memory (OOM) errors on our 128GB server. The reason is that SymphonyQG stores the quantized codewords of the neighbors in each data vector to fully exploit SIMD instructions for superior performance. However, it also has the precision-storage dilemma, as we discussed in Section 3.1. In particular, the index size of SymphonyQG is 4 times larger than the original dataset on Cohere (29GB) and BigCode (29GB) with the settings in [26], which incur OOM during SymphonyQG index construction on both datasets. The limitation of the scalability of SymphonyQG further confirms the efficiency of Laser as it is an on-disk system. Thirdly, considering the search latency, Laser also demonstrates strong competitiveness against Vamana (in-memory) and HNSWlib (in-memory) in all three datasets, as shown in Figures 12(d), (e) and (f). The performance of Laser is consistently better than its of Vamana and HNSWlib on GIST1M and Cohere. However, its performance is worse than its of Vamana and HNSWlib when Recall@10 is smaller than 0.98 on BigCode. The major reason is that Vamana and HNSWlib achieve the target recall (i.e, ≤ 0.98) by visiting much few nodes than Laser on BigCode. However, both Vamana and HNSWlib need to visit more nodes to achieve a high recall (e.g., ≥ 0.98), which enabling Laser to surpass them by unlocking its superior computation ability. As illustrated in Figure 12(f), the latency of Laser is smaller than Vamana and HNSWlib when the Recall@10 is higher than 0.98 on BigCode.

Varying K at Recall@K. We evaluate the performance of all on-disk systems by varying K in this experiment. Figure 13 shows the measured result of Recall@100 and Recall@1000 on Cohere, respectively. We omit the experimental results on other datasets as they share similar trend. It is no doubt our Laser is the overall winner among all these compared methods with different Recall@Ks. For example, the throughput of Laser is nearly 2x of Starling, and 4x of DiskANN when Recall@100 is 0.95. Interestingly, the latency of cluster-based index SPANN is better than existing graph-based



(a) Thr. with K=100 (b) Lat. with K=100 (c) Thr. with K=1000 (d) Lat. with K=1000

Fig. 13. Performance evaluation by varying K, Cohere

Table 4. Index construction cost (minutes)

	GIST1M	Cohere	BigCode	DPR	MSMARCO
DiskANN	22.8	68.8	51.7	1,464.5	1,532.4
Starling	18.4	70.6	71.0	1,420.5	1453.7
PipeANN	19.4	78.2	68.3	1,472.8	1,448.3
Laser	**14.2**	**52.3**	**49.8**	**1,532.4**	**1,436.4**

Table 5. Index size (GB)

	GIST1M	Cohere	BigCode	DPR	MSMARCO
DiskANN	7.7	39	40	386	867
Starling	7.7	39	40	386	867
PipeANN	7.7	39	40	386	867
Laser	**7.7**	**78**	**80**	**771**	**867**

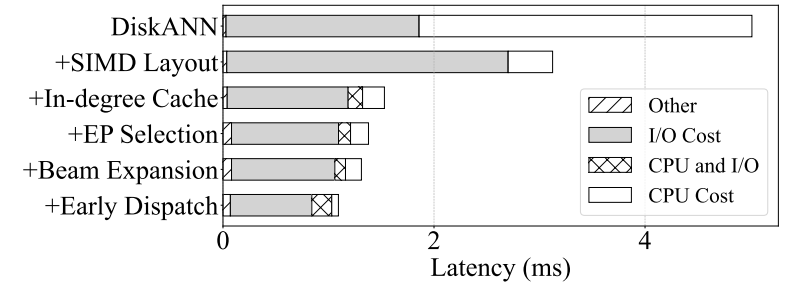
indexes (e.g., DiskANN, Starling, PipeANN) when K is 1000, see Figure 13(d). The core reason is that SPANN only needs 332 I/Os to visit clusters and compute a high quality top-1000 result set, but the on disk graph-based methods incur thousands I/Os (e.g., DiskANN uses 1115 I/Os) to achieve the similar recall at top-1000 result set. However, our Laser is the only on disk graph-based index method, which performs better than SPANN w.r.t latency when the Recall@1000 is larger than 0.95. This is because the computation cost of Laser is significantly smaller than SPANN to compute top-1000 results with high recall (i.e., Recall@1000 $\geq$ 0.95) as SPANN computes all the candidates data vectors in these accessed clusters and Laser obtains a tight yet high quality candidate set via the fast-computed approximate distances.

Index construction cost. Table 4 presents the Vamana index construction cost of the compared on-disk graph-based index systems on the 5 datasets. Comparing with DiskANN, Starling, and PipeANN, Laser introduces extra overhead for PCA transformation and data clustering during index construction. However, it significantly reduces cost during quantization phase as it only quantizes the principal components of data vector, but the other index methods need to quantize the original vectors. For example, the dimensionality of MSMARCO is 1,024, which is significantly larger than the used dimension of principal component 256. Interestingly, the gained benefit is slightly outweighs the introduced overhead. Thus, the index construction cost of Laser is slightly faster than its of existing on-disk graph-based index systems on all datasets.

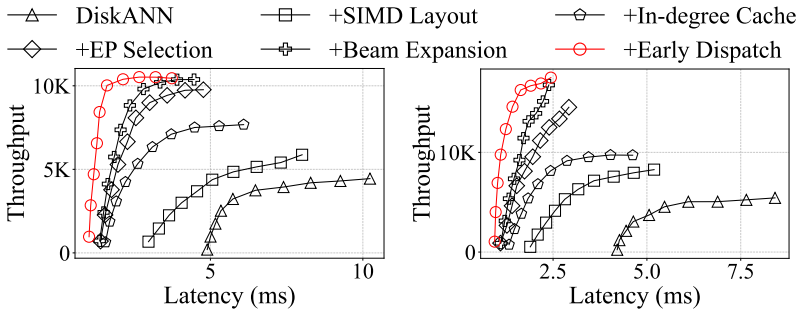
Index size. Table 5 reports the on-disk index size of all compared on-disk graph-based index methods on five datasets. As expected, Laser is typically larger than other on-disk graph-based methods as it stores quantized codewords in the SIMD-friendly layout. In particular, the node size of Laser is $S_{node} = 4d + 4R + R(\frac{d_{PCA}}{8} + 12)$ bytes, where d, R, and d_{PCA} are the dimensionality of data vectors, the out-degree of graph, and the dimensionality of principal components. However,

Table 6. Peak memory footprint during search phase

	GIST1M	Cohere	BigCode	DPR	MSMARCO
DiskANN	518MB	5.0GB	4.9GB	77GB	108GB
Starling	797MB	8.1GB	7.7GB	78GB	112GB
PipeANN	734MB	9.7GB	8.0GB	78GB	110GB
Laser	**769MB**	**7.7GB**	**7.7GB**	**89GB**	**98GB**



(a) Latency evaluation on GIST1M



(b) GIST1M (c) Cohere

Fig. 14. Laser ablation study, Recall@10=0.95

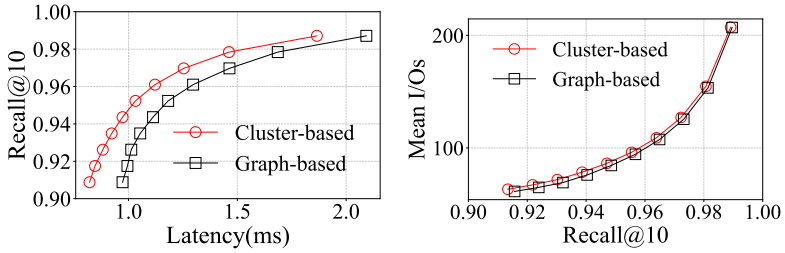
the node size of DiskANN (resp. Starling and PipeANN) is $S_{node} = 4d + 4R + 4$ bytes. Consequently, Laser exhibits the largest index size on Cohere, BigCode, and DPR. However, on GIST1M and MSMARCO, Laser has the same size as the other methods as Laser effectively utilizes the space wasted in existing methods due to I/O alignment padding.

Memory footprint. Table 6 presents the peak memory footprint of each on-disk graph-based index system during the search phase. The peak memory consumption is measured by using the **psutil** library [51]. The results confirm that Laser effectively utilizes the available memory budget. In particular, the SIMD-friendly layout of Laser eliminates the need to store a full set of compressed vectors in memory, and Laser utilized the available memory budget by proposing in-degree based node cache scheme, which significantly reduced the I/O cost, see the third bar in Figure 14(a).

5.3 Effectiveness Study

We conduct effectiveness study on Laser in this section.

Ablation study. To verify the effectiveness of Laser, we measure the latency of Laser by including each proposed technique step by step. Figure 14(a) illustrates the measured latencies on GIST1M with Recall@10=0.95. As analyzed in Section 3.3, the SIMD-friendly layout of Laser successfully addresses the compute-bound of DiskANN by slightly introducing I/O overheads, see the second



(a) Latency (b) Mean I/Os

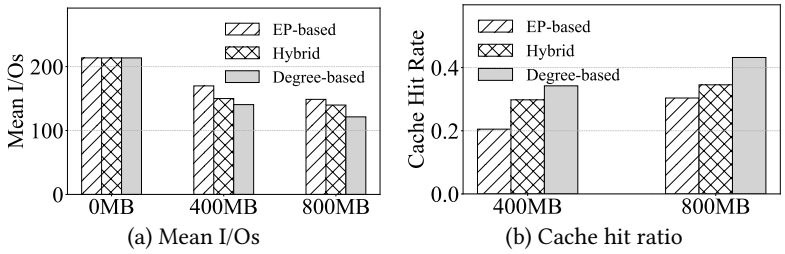
Fig. 15. Effect of entry point selection, GIST1M

bar in Figure 14(a). With SIMD-friendly layout, the in-degree node cache scheme reduces almost half of the I/O cost and overlaps computation and I/Os, as the third bar shown in Figure 14(a). The overall latency of Laser is further optimized by the proposed cluster-based entry point selection method, adaptive beam expansion strategy and early dispatch idea, as the last three bars shown in Figure 14(a).

We next evaluate the effectiveness of each proposed technique in Laser by measuring the corresponding throughput-latency curves. In particular, we tune the candidate list size parameter (*search_L*) to maintain the Recall@10 at 0.95. For each curve, we vary the number of used threads from 1 to 48. The experimental results on GIST1M and Cohere are plotted in Figures 14(b) and (c), respectively. It is clear that each proposed technique moves the throughput-latency curve of Laser closer to the top-left corner, which means Laser achieves the higher throughput and the lower latency by integrating the proposed techniques one by one. One interesting observation is that the early dispatch method on Laser mitigates the impact of long-tail I/O in multi-thread setting, as the red curves shown in Figures 14(b) and (c).

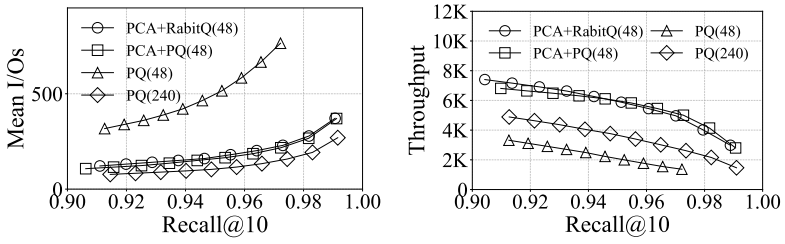
Effect of entry point selection. We investigate the effectiveness of the cluster-based entry point selection method of Laser by comparing it with the graph-based entry point selection method in [28, 57]. In particular, we replace the cluster-based method by the existing graph-based method in Laser. Figure 15(a) demonstrates that the cluster-based method of Laser achieves an approximate 10% latency reduction over the graph-based method. The performance gain is from that the cluster-based method only computes 300 candidates but the graph-based method computes 10,000 nodes in the in-memory navigation graph to identify the top-1 entry point for ANNS query processing on GIST1M. We next measure the mean I/Os during query processing to quantify the quality of the identified entry points by both methods. As shown in Figure 15(b), the mean I/Os curve of the cluster-based method is slightly larger than that of the graph-based method at the same recall. It confirms the effectiveness of our cluster-based method as it visits few nodes (i.e., 300 nodes) and achieves comparable quality of graph-based method, which visits more nodes (i.e., 10,000 nodes).

Effect of cache scheme. To fully utilize the available memory budget, different cache schemes have been proposed in existing on-disk graph-based solutions. In particular, DiskANN [36] caches the frequently accessed entry points and their neighbors. The hybrid cache scheme is devised in [68] which combines static cache (with 20% budget) and dynamic cache (with 80% budget). The static cache stores the same content as DiskANN and the dynamic cache employs Least Frequently Used (LFU) policy to dynamically update the cache budget by the visited nodes during query processing. We compare these existing cache schemes against our proposed in-degree based cache scheme for Laser in this experiment. Specifically, we replace the degree-based cache scheme of Laser by (i) EP-based cache scheme in [36] and (ii) hybrid cache scheme in [68], respectively. The tested cache budgets are 0% (0MB), 5% (400MB) and 10% (800MB) of the size of GIST1M. Figures 16(a) and



(a) Mean I/Os (b) Cache hit ratio

Fig. 16. Effect of cache scheme, Recall@10=0.95 on GIST1M



(a) Mean I/Os (b) Throughput

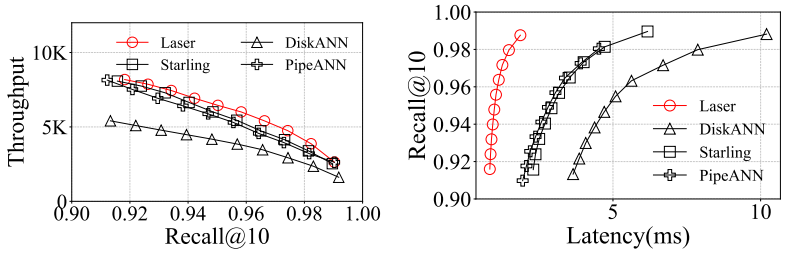
Fig. 17. Effect of quantization method, GIST1M

(b) reported the mean I/Os and cache hit ratio of tested cache schemes. We can observe: (i) the larger cache budget, the lower mean I/Os and the higher cache hit ratio of all cache schemes; and (ii) in-degree-based node cache strategy in Laser consistently outperforms the alternative cache schemes (e.g., EP-based and Hybrid).

Effect of quantization method. In this experiment, we investigate the effect of different quantization methods on the performance of Laser on GIST1M. In particular, we test 4 different quantization methods on Laser: (i) PQ with 240 bytes, (ii) PQ with 48 bytes, (iii) PCA+PQ with 48 bytes, and (iv) PCA+RabitQ with 48 bytes, which is the default setting of Laser. Figure 17 reports the mean I/Os and throughput of different quantization methods on Laser at different Recall@10. Firstly, the mean I/Os of PQ (48) is the largest as it severely loses precision due to high compression ratio. Secondly, PQ (240) has the smallest mean I/Os among all these 4 tested methods but its throughput is obvious below our PCA based methods. Thirdly, PCA-based methods are better than PQ only method and PCA+RabitQ(48) slightly outperforms PCA+PQ(48), which demonstrates (i) the generality of our PCA-based method and (ii) the efficiency of RabitQ.

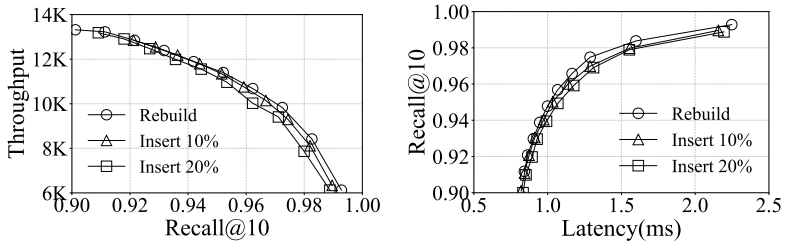
Effect of hardware configurations. We agree that the roofline model analysis is sensitive to the CPU and SSD configurations. To evaluate the effectiveness of our designs in Laser, we compare all on-disk graph-based index systems on a machine with small I/O bandwidth in this experiment. In particular, the I/O bandwidth is 3.7GB/s (i.e., we only use a single 1.92 TB SAMSUNG PM9A3 NVMe SSD), which is 11.1GB/s in our default setting. As depicted in Figures 18(a) and (b), Laser outperforms all existing on-disk graph-based index systems in terms of throughput and latency, respectively. Thus, it is safe to conclude Laser enjoys high I/O efficiency, computational efficiency and search performance simultaneously, which does not rely on any specific hardware configurations. We also agree that the performance improvement times of Laser over existing on-disk graph-based index systems may be different with different hardware configurations.

Effect of data updates. To evaluate the robustness of Laser, we divide GIST1M into two subsets: base set and insertion set. Laser constructs the on-disk graph-based index with the base set. Then Laser updates the index with the vectors in insertion set by following the same PCA matrices. The



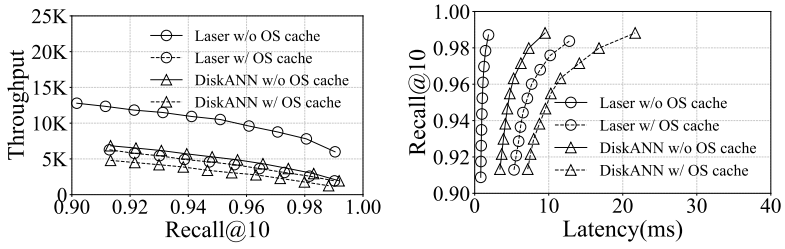
(a) Throughput (b) Latency

Fig. 18. Effect of hardware configuration, GIST1M



(a) Throughput (b) Latency

Fig. 19. Effect of data updates, GIST1M



(a) Throughput (b) Latency

Fig. 20. Effect of OS cache, GIST1M

Vamana graph structure of Laser is updated with the HNSW insertion strategy in [46]. Figure 19 depicts the throughput and latency of Laser in three cases: (i) rebuild index, (ii) 10%$|\mathcal{D}|$ inserts, and (iii) 20%$|\mathcal{D}|$ inserts. Firstly, we recommend periodic rebuilding the index of Laser to offer optimal performance, as rebuild index enjoys the best throughput and latency. Secondly, Laser is robust to data updates as the performance of 10%$|\mathcal{D}|$ inserts and 20%$|\mathcal{D}|$ inserts are slightly worse the rebuild index.

Effect of OS cache. We evaluate the effect of OS cache in Figure 20. Specifically, the effect of OS cache is disabled in our default setting. In this experiment we enable it on Laser and DiskANN by removing O_DIRECT flag. As illustrated in Figure 20, both Laser and DiskANN without OS cache perform better than with OS cache. The major reason is that the OS cache is ineffective (i.e., low cache hit rate) due to the random access pattern of graph-based search on both Laser and DiskANN.

6 Related Work

In-memory graph-based ANNS solutions. In-memory graph-based index solutions, such as HNSW [46], NSG [22], and Vamana [36], are the state-of-the-art solutions for high-performance approximate nearest neighbor search. These methods leverage various optimization strategies: HNSW enhances search speed through hierarchical small-world graphs, NSG ensures strong

connectivity via monotonic search paths, and Vamana improves robustness with two-phase graph pruning. Other notable approaches, such as KGraph [42] and DPG [41], focus on optimizing graph construction through neighborhood propagation and angle-based diversification, respectively, while SPTAG [13] adopts a divide-and-conquer strategy for better scalability. However, a fundamental limitation of these approaches is their requirement to store the entire index—comprising the graph structure and the original vectors—in main memory, which becomes a critical scalability bottleneck for large-scale high-dimensional vector data.

On-disk ANNS solutions. Besides the graph-based indexing adopted in DiskANN [36], Starling [57] and PipeANN [28], there are also solutions following a two-level architecture. In particular, GRIP [67] and FusionANNS [55] employ a two-stage "preview and validate" strategy: they first use in-memory quantized indexes to identify candidate vectors, and then use the original vectors stored on disk to rerank. While this approach ensures high final accuracy, the reranking step incurs significant random I/O costs, as fetching numerous small, non-contiguous vectors from an SSD leads to severe read amplification. SPANN [14] mitigates this overhead by clustering vectors and enabling sequential access to relevant clusters. However, maintaining high recall in SPANN still incurs high latency as it incurs substantial I/O accesses.

Acceleration techniques. To accelerate vector similarity search, two approaches are widely used. First, many studies compress vectors to reduce memory access costs. The representative techniques include Product Quantization (PQ) [37], Scalar Quantization [1], and RaBitQ [24]. Second, many methods have been studied, such as Quick ADC [2], Bolt [11], Quicker ADC [3], and SymphonyQG [26], which utilized SIMD instructions for the distance computation acceleration. Some algorithms further utilize modern hardwares like GPU [48, 61, 65] and CXL [35] to accelerate vector search. However, these techniques require excessive data storage and are specifically designed for in-memory solutions.

7 Conclusion

In this work, we revisited the design paradigm of on-disk graph-based index system for efficient large-scale, high-dimensional vector similarity search. We revealed a key paradigm shift, contrary to the prevailing wisdom, proving both theoretically and empirically that as vector dimensionality increases, the primary performance bottleneck transitions from disk I/O to CPU computation. To address this new compute-bound paradigm, we propose Laser, which consists of a novel SIMD-friendly data layout and an optimized search strategy. In particular, the novel SIMD-friendly index layout of Laser first resolves the computational bottleneck through parallelized computation with SIMD instructions, then enables in-degree based node cache scheme in Laser. The optimized search strategy addresses the re-emergent I/O bottleneck via a suite of optimization techniques, including cluster-based entry point selection, adaptive beam expansion strategy, and early dispatch mechanism. Extensive experiments demonstrate that the performance of Laser not only significantly surpasses existing on-disk index systems but also, is comparable to or slightly better than that of in-memory index solutions, e.g., HNSWlib. The promising directions for further work is (i) processing out-of-distribution queries on on-disk graph-based index system efficiently, and (ii) accelerating the search performance of on-disk index system on heterogeneous computing hardware.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (Grant Nos. 62422206, 62532007, 62532001) and a research gift from AlayaDB.AI.

References

[1] Cecilia Aguerrebere, Ishwar Bhati, Mark Hildebrand, Mariano Tepper, and Ted Willke. 2023. Similarity search in the blink of an eye with compressed indices. *arXiv preprint arXiv:2304.04759* (2023).

[2] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2017. Accelerated nearest neighbor search with quick adc. In *ICMR*. 159–166.

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2019. Quicker adc: Unlocking the hidden potential of product quantization with simd. *TPAMI* 43, 5 (2019), 1666–1677.

[4] Argilla. 2024. PersonaHub-FineWeb-Edu-4-Embeddings. Hugging Face dataset. https://huggingface.co/datasets/argilla-warehouse/personahub-fineweb-edu-4-embeddings Accessed: 2025-10-06.

[5] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[6] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. Elpis: Graph-based similarity search for scalable data science. *PVLDB* 16, 6 (2023), 1548–1559.

[7] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *TPAMI* 37, 6 (2014), 1247–1260.

[8] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. 2014. Neural codes for image retrieval. In *ECCV*. 584–599.

[9] Zheng Bian, Man Lung Yiu, and Bo Tang. 2025. IGP: Efficient Multi-Vector Retrieval via Proximity Graph Index. In *SIGIR*. 2524–2533.

[10] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[11] Davis W Blalock and John V Guttag. 2017. Bolt: Accelerated data mining with fast vector compression. In *SIGKDD*. 727–735.

[12] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *PVLDB* 17, 11 (2024), 2735–2749.

[13] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search.* https://github.com/Microsoft/SPTAG Accessed: 2025-5-12.

[14] 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. *NIPS* 34 (2021), 5199–5212.

[15] Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, et al. 2025. Retroinfer: A vector-storage approach for scalable long-context llm inference. *arXiv preprint arXiv:2505.02922* (2025).

[16] Cohere. 2025. msmarco-v2.1-embed-english-v3. Hugging Face dataset. https://huggingface.co/datasets/Cohere/msmarco-v2.1-embed-english-v3 Accessed: 2025-10-06.

[17] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RECSYS*. 191–198.

[18] Yangshen Deng, Zhengxin You, Long Xiang, Qilong Li, Peiqi Yuan, Zhaoyang Hong, Yitao Zheng, Wanting Li, Runzhong Li, Haotian Liu, Kyriakos Mouratidis, Man Lung Yiu, Huan Li, Qiaomu Shen, Rui Mao, and Bo Tang. 2025. AlayaDB: The Data Foundation for Efficient and Effective Long-context LLM Inference. In *SIGMOD*. New York, NY, USA, 364–377.

[19] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *SIGKDD*. 6491–6501.

[20] C Fu. 2016. Efanna: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. *arXiv preprint arXiv:1609.07228* (2016).

[21] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *TPAMI* 44, 8 (2021), 4139–4150.

[22] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[23] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *SIGMOD* 1, 2 (2023), 1–27.

[24] Jianyang Gao and Cheng Long. 2024. RaBitQ: quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *SIGMOD* 2, 3 (2024), 1–27.

[25] Albert Gordo, Jon Almazán, Jerome Revaud, and Diane Larlus. 2016. Deep image retrieval: Learning global representations for image search. In *ECCV*. 241–257.

[26] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. 2025. SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. *SIGMOD* 3, 1 (2025), 1–26.

[27] Part Guide. 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: system programming guide, Part* 2, 11 (2011), 0–40.

[28] Hao Guo and Youyou Lu. 2025. Achieving Low-Latency Graph-Based Vector Search via Aligning Best-First Search Algorithm with SSD. In *OSDI*. 171–186.
[29] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: a cloud native vector database management system. *PVLDB* 15, 12 (2022), 3548–3561.
[30] Gabriel Haas, Michael Haubenschild, and Viktor Leis. 2020. Exploiting Directly-Attached NVMe Arrays in DBMS.. In *CIDR*, Vol. 20. 2.
[31] 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. *PVLDB* 16, 9 (2023), 2090–2102.
[32] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*. 2333–2338.
[33] Hugging Face Inc. 2025. Hugging Face Datasets: A Community Library of Natural Language Data. https://huggingface.co/datasets. Accessed: 2025-10-08.
[34] IRISA. 2025. Corpus Tex-Mex: Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/. Accessed: 2025-10-06.
[35] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS: Software-Hardware Collaborative Memory Disaggregation and Computation for Billion-Scale Approximate Nearest Neighbor Search. In *ATC*. 585–600.
[36] 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. *NIPS* 32 (2019).
[37] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *TPAMI* 33, 1 (2010), 117–128.
[38] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. 2010. Aggregating local descriptors into a compact image representation. In *CVPR*. 3304–3311.
[39] Yinsicheng Jiang, Yeqi Huang, Liang Cheng, Cheng Deng, Xuan Sun, and Luo Mai. 2026. ContextPilot: Fast Long-Context Inference via Context Reuse.
[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NIPS* 33 (2020), 9459–9474.
[41] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *TKDE* 32, 8 (2020), 1475–1488.
[42] Wei Ling. 2025. kgraph: A Fast Approximate k-NN Graph Construction Library. https://github.com/aaalgo/kgraph. Accessed: 2025-03-09.
[43] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. 2024. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516* (2024).
[44] Fuyu Lv, Taiwei Jin, Changlong Yu, Fei Sun, Quan Lin, Keping Yang, and Wilfred Ng. 2019. SDM: Sequential Deep Matching Model for Online Large-scale Recommender System. In *CIKM*. 2635–2643.
[45] Vasilis Mageirakos, Bowen Wu, and Gustavo Alonso. 2025. Cracking Vector Search Indexes. *PVLDB* 18, 11 (2025), 3951–3964. doi:10.14778/3749646.3749666
[46] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI* 42, 4 (2018), 824–836.
[47] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition* 96 (2019), 106970.
[48] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *ICDE*. 4236–4247.
[49] Keiron O'shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).
[50] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. 2023. iQAN: Fast and Accurate Vector Search with Efficient Intra-Query Parallelism on Multi-Core Architectures. In *PPoPP*. 313–328.
[51] Giampaolo Rodolà. 2024. psutil—System and Process Utilities for Python. GitHub repository. https://github.com/giampaolo/psutil Accessed: 2025-10-06.
[52] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. In *EMNLP*. 9248–9274.
[53] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, et al. 2024. Results of the Big ANN: NeurIPS'23 competition. *arXiv preprint arXiv:2409.17424* (2024).

[54] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. 2022. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*. 177–189.

[55] 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 *FAST*. 171–185.

[56] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD*. 2614–2627. doi:10.1145/3448016.3457550

[57] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An i/o-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. *SIGMOD* 2, 1 (2024), 1–27.

[58] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *PVLDB* 14, 11 (2021), 1964–1978.

[59] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[60] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2024. A survey on large language models for recommendation. *WWW* 27, 5 (2024), 60.

[61] Long Xiang, Bo Tang, and Chuan Yang. 2019. Accelerating exact inner product retrieval by cpu-gpu systems. In *SIGIR*. 1277–1280.

[62] Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities. *SIGMOD* 3, 1, Article 82 (2025), 28 pages.

[63] Yahoo Japan Corporation. 2024. NGT: Nearest Neighbor Search with Neighborhood Graph and Tree for High-dimensional Data. https://github.com/yahoojapan/NGT. Accessed: 2025-5-12.

[64] Artem Babenko Yandex and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *CVPR*. 2055–2063.

[65] Yuxiang Yang, Shiwen Chen, Yangshen Deng, and Bo Tang. 2025. ParaGraph: Accelerating Graph Indexing through GPU-CPU Parallel Processing for Efficient Cross-modal ANNS. In *DaMoN*. Article 7, 10 pages.

[66] Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Rankrag: Unifying context ranking with retrieval-augmented generation in llms. *NIPS* 37 (2024), 121156–121184.

[67] Minjia Zhang and Yuxiong He. 2019. Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In *CIKM*. 1673–1682.

[68] Yijie Zhou, Shengyuan Lin, Shufeng Gong, Song Yu, Shuhao Fan, Yanfeng Zhang, and Ge Yu. 2025. GoVector: An I/O-Efficient Caching Strategy for High-Dimensional Vector Nearest Neighbor Search. *arXiv preprint arXiv:2508.15694* (2025).

Received October 2025; revised January 2026; accepted February 2026