ANSMET: Approximate Nearest Neighbor Search with Near-Memory Processing and Hybrid Early Termination

Yiwei Li*
Tsinghua University
Beijing, China
Moonshot AI
Beijing, China
liyiwei@moonshot.cn

Yuxin Jin*
Tsinghua University
Beijing, China
jin-yx21@mails.tsinghua.edu.cn

Boyu Tian
Tsinghua University
Beijing, China
tby20@mails.tsinghua.edu.cn

Huanchen Zhang
Tsinghua University
Beijing, China
Shanghai Qi Zhi Institute
Shanghai, China
huanchen@tsinghua.edu.cn

Mingyu Gao
Tsinghua University
Beijing, China
Shanghai Artificial Intelligence Lab
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China
gaomy@tsinghua.edu.cn

Abstract

Approximate nearest neighbor search (ANNS) is a fundamental operation in modern vector databases to efficiently retrieve nearby vectors to a given query. On general-purpose computing platforms, ANNS is found not only to be highly memory-bound due to the large amount of high-dimensional vectors to access, but also exhibits very low utilization of the fetched data as many memory accesses and computations are wasted on not-so-nearby vectors. To alleviate these two inefficiencies, we propose a hardware-software co-design that integrates near-data processing architectures with a novel hybrid partial-dimension/bit early termination strategy. Distance calculation and comparison in ANNS are offloaded to the near-data processing units at the memory rank level. As a vector is being gradually fetched from memory, we conservatively estimate a lower bound of its distance to the query using the partially fetched data, e.g., a subset of dimensions and/or partial bits of each element. If this lower bound already exceeds a threshold, we could early terminate to avoid future unnecessary data accesses and computations. In the presence of such irregular early termination execution flow, we further optimize the data layouts within a single memory access and across multiple memory ranks in the system, and handle efficient coordination between the near-data units and the host processor that executes the rest of index traversal and result sorting. With all the above optimizations, our design demonstrates an average 5.26× speedup of using near-data processing, and another 1.52× from enabling hybrid early termination on top of it.

*Both authors contributed equally to this research.

ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/25/06
https://doi.org/10.1145/3695053.3731013

CCS Concepts

• **Hardware** → **Memory and dense storage**; • **Information systems** → **Nearest-neighbor search**; • **Computer systems organization** → Special purpose systems.

Keywords

approximate nearest neighbor search, hardware acceleration, near-data processing, early termination

ACM Reference Format:
Yiwei Li, Yuxin Jin, Boyu Tian, Huanchen Zhang, and Mingyu Gao. 2025. ANSMET: Approximate Nearest Neighbor Search with Near-Memory Processing and Hybrid Early Termination. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3695053.3731013

1 Introduction

High-dimensional vectors are powerful representations for text, image, audio, and video data [61, 67]. Consequently, vector databases, which facilitate the insertion and retrieval of high-dimensional vectors, play a crucial role in applications such as search engines [29], e-commerce platforms [53], and retrieval-augmented generation (RAG) [9, 51]. The common operation is to find nearby vectors to a given query vector according to a certain distance metric. However, not only the number of vectors in the database but also the number of dimensions in each vector is quite high in typical application scenarios, hindering the use of direct brute-force search.

As a result, approximate k-nearest neighbor search (ANNS) becomes a prominent approach in modern vector databases. ANNS has many algorithm-level optimizations. One example is to use various vector indexes, including trees [8, 12, 64], hash tables [20, 21, 34], inverted lists [38], and graphs [22, 24, 58, 65], to organize vector data in a structured way according to their similarities, and accelerate the search by effectively identifying potentially nearby vectors to the query. Nevertheless, even with effective indexing, we observe

that ANNS still exhibits a highly memory-bound nature due to the excessive data accesses to high-dimensional vectors and the low arithmetic intensity of distance calculation and comparison. Furthermore, a significant portion of the encountered vectors during the search are actually *not* close, i.e., their distances to the query are larger than the current threshold (e.g., the kth smallest distance), making both data accesses and computations ineffectual.

In this work, we propose a set of hardware and software techniques to improve the performance and efficiency of ANNS processing. At the *hardware* level, we begin by adopting a *DIMM-based near-data processing* (NDP) architecture, which has been extensively studied before for high-dimensional vector aggregation in recommendation systems and graph neural networks [4, 42, 47, 55, 66, 76, 82, 91]. We choose the DIMM-based implementation over other NDP variants because of its relatively less intrusive changes to the DRAM chips and interface, and because of the high capacity (up to terabytes, crucial to large vector databases) it could offer through the commodity DRAM modules. Running the memory-intensive distance comparison tasks on the DIMM-side NDP logic units can benefit from the abundant data bandwidth and memory-level parallelism, thus improving performance and reducing energy.

Furthermore, ANNS exhibits unique algorithmic optimization opportunities compared to other applications. At the *software* level, we propose a novel *hybrid partial-dimension/bit early termination* strategy to save unnecessary data accesses and computations in distance comparison, if the partially fetched vector data and the partially calculated result can already prove this vector is beyond the current distance threshold. This is done by estimating a lower bound on the distance between the vector and the query, using the partial information retrieved so far and conservatively setting the missing data. For example, we can prove that the minimum distance between $(1, x)$ and $(4, -1)$ is obtained when $x = -1$ (partial dimensions), and that between $00\_\__2$ and 0110_2 is when the missing bits are 11_2 (partial bits). By always using lower bounds, our early termination algorithm guarantees no accuracy loss. Existing early termination approaches are at the vector level [13, 52, 87, 90] or dimension level [25, 69, 86]. They either incur accuracy loss due to aggressive predictions or result in limited fetch reduction with conservative strategies. In contrast, we introduce a novel bit-level early termination mechanism in combination with partial dimensions, achieving not only *higher efficiency* than previous early termination schemes, but also *with no accuracy loss*.

We integrate the NDP hardware architecture and the early termination software technique into a coherent design called ANSMET. ANSMET is a heterogeneous CPU+NDP platform. It offloads the memory-intensive distance comparison tasks to its DIMM-based NDP units while keeping the complex and irregular index traversal and result sorting parts on the host CPU to be general for supporting different index structures.

We further address several system-level challenges due to the introduction of early termination. First, to maximize the effectiveness of early termination and save as many subsequent accesses as possible, we need to pick judiciously which partial data of a vector to fetch in each 64 B memory access: either more dimensions but fewer bits per dimension, or more bits of each dimension but fewer dimensions. This affects directly which data should be packed together in the data layout. We propose a lightweight offline sampling-based approach to systematically explore the design space and to effectively decide the optimized data layout.

Second, the global data layout across multiple memory ranks in the system should be examined. Existing DIMM-based NDP designs use a combination of vertical (splitting dimensions) and horizontal (dividing vectors into groups) partitioning on the vector database [76, 82, 91]. Such a hybrid way is still efficient for ANSMET, but early termination shifts the sweet spot between the two schemes and enables a simpler but more effective load-balancing technique.

Third, we propose an adaptive polling scheme to tackle the difficulty of retrieving results from the NDP side by the host CPU under unpredictable processing latencies due to early termination. The best polling period is accurately estimated using the same sampling-based preprocessing above.

We evaluate ANSMET through cycle-accurate simulation on the representative ANNS index, Hierarchical Navigable Small Worlds (HNSW) [58]. Using NDP improves performance by 5.26× on average and up to 6.40× over the CPU baseline, by increasing the theoretical available bandwidth to 8×. Our early termination further brings an average 1.31× speedup on CPU and 1.52× on NDP, thanks to the savings of unnecessary accesses and computations.

2 Background

We introduce the preliminaries of approximate nearest neighbor search and near-data processing in this section.

2.1 Approximate Nearest Neighbor Search

The key operation for high-dimensional vectors in vector databases is k-nearest neighbor (kNN) search, which identifies the k closest vectors to a given query vector. However, because the number of vectors N is usually at billion scales and their dimension D could be tens to hundreds in real workloads [17, 71], brute-force comparison has prohibitively high time complexity of $O(ND)$. Consequently, approximate nearest neighbor search (ANNS) has been widely used to find an approximation of the kNN result, reducing search complexity at the expense of slightly decreased accuracy.

We mainly focus on the search phase (a.k.a., inference) of ANNS in this paper. ANNS algorithms typically accelerate the search process in two ways, by either reducing the number of accessed vectors using *vector indexing*, or reducing the vector dimension using *vector quantization*.

Vector indexing includes two general types of indexes. The *cluster-based* indexes aim to place nearby vectors in the same cluster, so only a few clusters need to be searched. Clusters can be organized in tree structures like kd-trees and octrees [8, 12, 64], by hashing functions like locality-sensitive hashing [20, 21, 34], or as inverted file indexes [38]. For each query vector, a few closest cluster centroids are identified using such indexes, and only the vectors in the corresponding clusters are checked. We maintain a heap to keep the k nearest vectors during the search. The number of searched clusters can be adjusted to trade between accuracy and query time. However, cluster-based indexes may suffer from imbalanced cluster sizes and low accuracy near the cluster boundaries [15].

On the other hand, *graph-based* indexes [22, 24, 58, 65] capture the distance relationship using proximity graphs, where vectors

are represented as graph vertices, and only close vectors are connected by edges. ANNS is conducted through graph traversals. One representative algorithm is Hierarchical Navigable Small Worlds (HNSW) [58]. HNSW constructs multiple layers each associated with a graph. The base layer contains all vectors as the vertices, while each upper layer gradually contains only a subset of vectors of its lower layer. The search starts at a deterministic entry point at the top layer. Within each layer, the nearest neighbors of the entry points are identified, while these neighbors are further used as the entry points to the next layer. Greedy beam search is used to keep the best candidates across layers, involving a search set and a result set. The search set is an unbounded min-heap sorted by distances, containing all vector candidates to search. The result set contains the k' nearest vectors visited thus far. Each time we pop a vector from the search set, find its unvisited neighbors, and add them to both sets if their distances to the query vector are smaller than *the maximum distance in the result set*. Note that k' can be larger than k to allow a looser threshold to keep more candidates at the cost of higher search time [25]. Graph-based indexes improve accuracy by incorporating multi-hop distance information, but they are slower and consume more memory due to the multiple proximity graphs.

Vector quantization compresses the vectors to reduce data size and thus access cost. Scalar quantization converts the elements of the vectors to lower precision, e.g., representing 16-bit floating-point numbers in 8-bit unsigned integers. Product quantization [37] divides the D-dimensional vector space into m subspaces. For each subspace, a separate codebook is generated using clustering techniques. During quantization, each vector is represented by the codewords of the nearest cluster centroids to the sub-vectors in all the subspaces. Since only the centroid codewords are used in the representation, memoization can be used to store the distance between each centroid codeword and the querying sub-vector in every subspace. The final distance computation then becomes selecting from these memorized distances and aggregating them [50]. However, vector quantization may compromise accuracy, particularly for out-of-distribution vectors [77].

Distance definitions. To calculate the distance between two vectors, specific similarity metrics can be used. The Euclidean distance is a typical choice, computed as $d^{(E)} = \sqrt{\sum_{i=0}^{D-1}(a_i - b_i)^2}$. The inner-product distance is derived as $d^{(I)} = -\sum_{i=0}^{D-1} a_i b_i$. The cosine similarity further normalizes the inner product by the magnitudes of the vectors, as $d^{(C)} = d^{(I)} / \sqrt{\sum_{i=0}^{D-1} a_i^2 \sum_{i=0}^{D-1} b_i^2}$. Note that we can do this normalization during preprocessing, after which the inner-product distance and the cosine distance become the same, and we can avoid expensive division and square root at runtime.

2.2 Near-Data Processing

Near-data processing (NDP) brings processing logic closer to data storage locations, e.g., main memory, in order to exploit higher bandwidth and lower latency of data access. Such physical proximity reduces the overheads associated with data movements in memory-intensive applications. NDP systems targeting main memory can be broadly classified into two categories: those that retain the traditional DIMM form factor, and those that utilize 3D-stacked memories like High Bandwidth Memory (HBM) [36]. Although

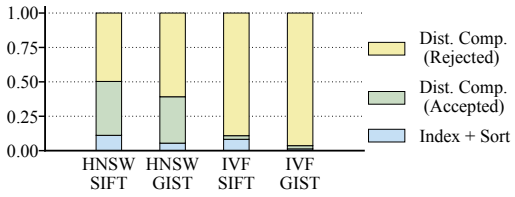

Figure 1: Performance breakdown of IVF and HNSW indexes on SIFT and GIST datasets, running FAISS [23] on CPUs.

3D-stacked NDP systems [2, 10, 18, 26, 27, 43, 45, 74, 84, 93] offer extremely high bandwidth, they currently face limitations in the capacity due to the challenges of die stacking and thermal management. Their typical capacities, e.g., dozens of gigabytes [36, 59], are much smaller than terabyte-scale vector databases.

In contrast, DIMM-based NDP integrates computing logic within the conventional DRAM hierarchy. In each memory channel, several DIMMs are connected to the host CPU via shared data (DQ) and command/address (C/A) buses. Each DIMM is further organized into ranks and banks, with each bank consisting of memory cells. The NDP logic can be placed at the DIMM level in the separate DIMM buffer chip [11, 19, 33, 42, 47, 62, 73], or near the DRAM banks in the DRAM chips [28, 31, 41]. These logic units leverage the rank/bank-level parallelism and the much higher internal bandwidth. Compared to other choices, DIMM-based NDP incurs small modifications to the commodity DRAM architecture and can also support large capacity; e.g., Samsung recently announced a 1 TB DDR5 module [70]. As a result, we argue that DIMM-based NDP systems are more well-suited for ANNS tasks.

3 Motivation and Proposal

We first analyze the performance bottlenecks in ANNS to motivate our design. We execute the popular vector search framework FAISS [23] on a 14-core Intel Xeon 5120 CPU. We focus on representative cluster-based and graph-based indexes, i.e., IVF and HNSW. Table 2 describes the datasets used in the experiments.

The performance breakdown is illustrated in Figure 1. We can see that a significant portion of the execution time is spent on vector distance comparison, including both fetching the vector data and calculating the distance. Previous research has also observed this trend [25, 35]. A closer look reveals that this stage is mainly bound by *the high memory access overheads* caused by two reasons. First, even with efficient indexes, each query still needs to access many vectors. For instance, an average of 617 vectors are fetched per query in HNSW-SIFT. Each vector contains tens to hundreds of elements, leading to heavy data traffic (e.g., > 300 kB per query). Second, the overall distance comparison has low arithmetic intensity with modern SIMD instructions. For example, the Euclidean distance requires one `VSUBPS` and one `VMULPS` instruction on x86 to calculate the squares of differences for four FP16 elements, and several `VDPPS` and one `VSQRTSS` for aggregation and square root, resulting in 0.125 op/byte for 128-dimension vectors. Other distance metrics and wider SIMD instructions exhibit even lower arithmetic intensity, e.g., 0.093 op/byte for the inner-product distance.

We also observe in Figure 1 that 50% to over 90% of the distance comparisons were "rejected", i.e., the fetched vector is beyond the

distance threshold — the maximum distance in the result set for both IVF and HNSW (Section 2.1). As a result, *a large portion of the data accesses and computations are ineffectual*. This calls for algorithms to effectively identify and filter (i.e., early terminate) these unnecessary data accesses and computations.

High-level ideas. We propose techniques in both hardware and software to address the above issues of excessive memory accesses and low fetch utilization, respectively. First, at the **hardware** level, we propose to use a *DIMM-based NDP architecture* for ANNS. We place specialized distance computing logic in the buffer chips of the multiple DIMMs in the system. We offload the memory-intensive distance comparison tasks in ANNS to such NDP logic to leverage high bandwidth and parallelism. To the best of our knowledge, this is the first DIMM-based NDP system specialized to ANNS.

While our NDP system looks similar to previous designs for recommendation systems [4, 42, 47, 55, 66, 82] that also process high-dimensional embedding vectors, the algorithm difference poses a new opportunity. Unlike recommendation systems that must aggregate all required vectors and return the full result, ANNS only needs the labels of the top-k vectors. Therefore, at the **software** level, we propose a novel *hybrid partial-dimension/bit early termination* strategy to reduce unnecessary data fetches and improve data utilization without sacrificing accuracy. We estimate the distance lower bound using the partially fetched data and stop issuing future accesses if this lower bound is already higher than the threshold.

Challenges. With the above high-level ideas, we still need to resolve the following specific problems to realize an efficient system. First, to maximize the effectiveness of early termination, we must carefully decide what part of the vector to fetch first, so that we have the best distance estimation to reject unpromising vectors as early as possible. The choices include fetching a subset of the dimensions of a vector or fetching partial bits from each dimension element. We comprehensively consider the design space and propose an effective sampling-based method to decide the data organization for early termination with maximum savings.

Second, we must optimize the data layout across multiple DRAM ranks to fully exploit the rank-level parallelism for NDP. For example, we can place different vectors across ranks to allow accessing them simultaneously [4, 47], e.g., for all vectors in a cluster, or all neighbors in a proximity graph. Alternatively, we can split the dimensions of a single vector [42, 66]. Assuming that we split a 128-dim FP32 vector into eight chunks and distribute them in different ranks, we can perform eight 64 B accesses in parallel to get the full vector. Moreover, data accesses would become irregular and imbalanced with early termination, which further complicates the decision. We thus revisit the prior hybrid data layouts that partition at both the vector and dimension levels [76, 82, 91] under the new tradeoffs brought by the unique characteristics of early termination.

Third, ANNS requires traversing the structure of clusters (cluster-based) or the proximity graphs (graph-based) and sorting the candidates in the search/result sets. Index traversal and vector sorting usually exhibit high dependencies, limited parallelism, and irregular computation patterns. These characteristics make them ill-suited for the NDP logic and better be executed on the CPU. We therefore use CPU+NDP heterogeneous processing with a carefully managed cooperation between the two sides to harmonize the execution of index traversal and distance comparison.

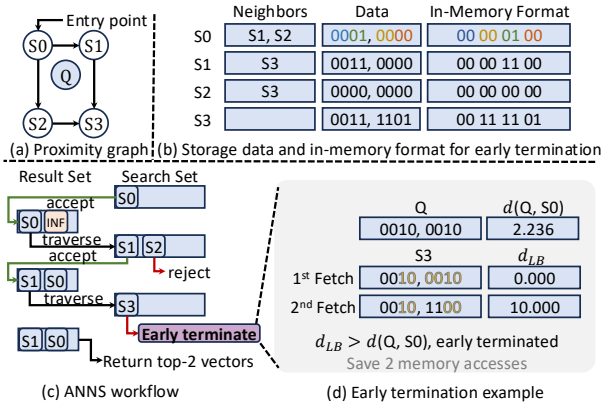

Figure 2: ANNS with early termination. (a) and (b) show an example proximity graph and its storage format. (c) shows the ANNS flow on this graph with query Q, while S3 triggers an early termination in (d).

We integrate our hardware and software techniques into a comprehensive solution named ANSMET. Next, we present our early termination algorithm details in Section 4, and then discuss the architectural designs including data layout optimization and heterogeneous cooperation in Section 5.

4 Hybrid Early Termination Algorithm

A key contribution of ANSMET is the algorithm-level early termination technique that reduces the memory accesses to improve the performance of distance comparisons in ANNS. For a high-dimensional vector, we can estimate a *lower bound* of its distance d_{LB} to the given query vector even with only partial data fetched (e.g., a subset of the dimensions or a subset of the bits in each dimension). For example, given a partial vector of $(1, 2, x_2, x_3)$, the Euclidean distance lower bound to a query $(4, -2, 6, -1)$ would be $d_{LB}^{(E)} = \sqrt{(4-1)^2 + (-2-2)^2} = 5$, when $x_2 = 6$ and $x_3 = -1$. On the other hand, if we know a subset of bits in a vector, e.g., $(00\_\__2, 01\_\__2)$, then its $d_{LB}^{(E)}$ to a query $(0110_2, 0101_2)$ should be 3 if the vector is $(0011_2, 0101_2)$. We call the above two cases *partial dimensions* and *partial bits*, respectively. For ANNS, if d_{LB} already exceeds the current threshold (e.g., the maximum distance in the result set in HNSW), we can safely drop this vector and early terminate its subsequent data fetches.

In this section, we first describe the overall execution flow of our hybrid partial-dimension/bit early termination strategy (Section 4.1). To achieve the best hybrid scheme between the two, we propose a systematic sampling-based approach to determine the optimized data layout in memory, which minimizes the number of fetches needed (Section 4.2). Finally, we discuss more complex layouts and compatibility with vector quantization (Section 4.3).

4.1 Overall Execution Flow

Figure 2 illustrates the algorithm details of ANNS with early termination. We use HNSW, a graph-based index as an example. Note that early termination also applies to other indexes including cluster-based ones. The graph structure and the vector data in Figure 2(a)

are stored in memory as shown in (b). We assume 4 vectors with 2 dimensions each, and each element is 4-bit. While the plain data layout continuously stores the bits of each element and the elements in each vector, efficient early termination requires a transformed layout that enables a hybrid of partial-dimension and partial-bit fetches. In this example, assuming each data fetch obtains 2 bits (in real hardware this is typically the 64-byte cacheline granularity), we store the highest (most significant) 2 bits of both elements in the vector sequentially, and then their rest 2 bits, as in Figure 2(b). We discuss how to determine optimized data layouts in Section 4.2.

During the online search in Figure 2(c), we calculate the distance between the query Q and the vector S*i* popped out from the search set, and then compare it with the current threshold. Specifically, the entry point S0 is within the initial distance threshold (i.e., ∞) and thus brings its neighbors S1 and S2 into the search set. S1 is accepted and brings in its neighbor S3, while S2 is rejected after fetching the entire vector. Next, S3 triggers an early termination as in (d). The current threshold is $d(\mathrm{Q}, \mathrm{S0}) = 2.236$ since S0 is at the tail of the result set. The first fetch gets the highest 2 bits of S3's first element, and estimates $d_{LB}(\mathrm{Q}, \mathrm{S3}) = 0$, which does not exceed the threshold. The second fetch gets 2 more bits and refines d_{LB} to 10.000, which is higher than the threshold. S3 is therefore rejected, and early termination saves half of the data traffic in this example.

Preprocessing cost. Both the layout exploration and the actual data transformation happen offline as preprocessing steps on the vector database similar to vector quantization techniques [37]. The preprocessing cost is modest for several reasons. First, the preprocessing time is usually amortized over long-time online ANNS. Second, the layout transformation can be easily parallelized on individual vectors. Third, the graph construction time in HNSW constitutes the largest offline cost, where we find the vector data layout transformation adds only 1.6% extra overheads.

How to set missing bits. When estimating the distance lower bound, we need to properly set the missing elements/bits in the partially fetched vector. The correct values depend on the distance metric. For each distance metric, we use a unified way to handle integer, fixed-point, and floating-point (FP32, FP16, BF16, etc.) types, as they all follow the fact that, *the bits having more impact on distance calculation are towards the more significant positions and fetched earlier*; e.g., the exponent is fetched before the mantissa. For the Euclidean distance, we compare the already fetched (more significant) bits and the corresponding bits in the query vector. If they match, the missing (less significant) bits should be set to be the same as the remaining bits in the query vector. If the partially fetched bits are larger (resp. smaller), the missing bits should be all 0s (resp. 1s). For example, for a query 0101_2, the partially fetched $01\_\__2$ should be set to 0101_2, $00\_\__2$ to 0011_2, and $11\_\__2$ to 1100_2. Signed numbers are similarly handled. For the inner-product distance, bit 1 should be set for unsigned data and when the sign bits are the same for signed data; otherwise use 0.

Comparison with prior early termination methods. Early termination can be applied to ANNS at various levels. Previous work uses machine-learning-based prediction models to bypass some index traversals and reduce the number of vectors accessed [13, 52, 87, 90], enabling early termination *at the vector level*. However, these prediction-based methods usually incur accuracy loss. Other designs use early termination *at the dimension level*, by scaling

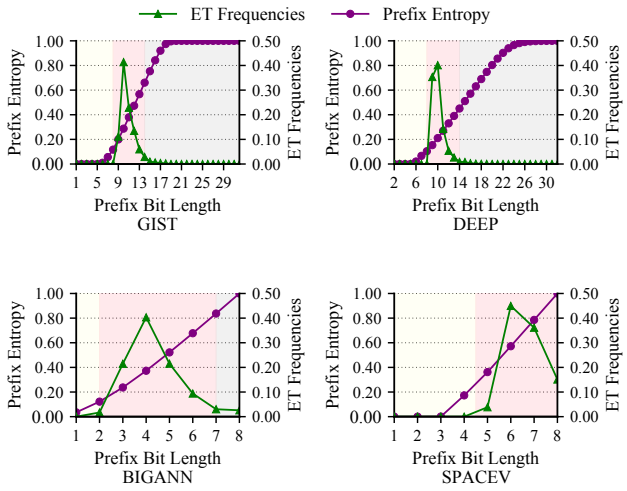

Figure 3: Prefix entropy and early termination (ET) frequency distribution over different prefix lengths. The left-side yellow area shows the low-entropy range. The middle pink area shows the high-termination range.

partial distance results with random projection [25], or through only partial-dimension fetches [69, 86]. Relying only on partial dimensions can only use conservative distance lower bounds. For example, with inner-product or cosine distances, the unfetched dimensions may contribute negative values to the results. Furthermore, opportunities such as common prefixes are not exploited. We are the first to enable early termination *at both the bit and dimension levels* with hardware support. Our approach has no accuracy loss, and can even be used in *accurate* search algorithms like kmeans and kNN. We also combine partial dimensions and partial bits in this work. If accuracy loss is allowed, our design can be further combined with the aforementioned partial-vector methods.

4.2 Data Layout Optimization

In DRAM, each data fetch has a fixed granularity, typically 64 B. Given the choices of partial dimensions and partial bits, we need to decide which of the following two is more effective in terms of minimizing the number of fetches: more bits per element but fewer elements in one fetch, or fewer bits per element but more elements. Within one element, the higher (more significant) bits, including the sign bit for signed data and the exponent bits for floating-point data, are more important and should be fetched first. On the other hand, different dimensions are typically considered of equal importance and are normalized and standardized during feature extraction [56]. Thus, we apply the same partial-bit fetch pattern across all dimensions. With these heuristics, the ith fetch would contain m_i elements with the next high n_i bits of each, where $m_i = \lfloor 64 \times 8 / n_i \rfloor$. The transformed data layout should match the above fetch granularity exactly, with padding if needed. For example, a 64 B chunk may contain the next highest 9 bits from 56 dimensions, with 8 padding bits at the end. The remaining task is to determine the value of n_i for each step.

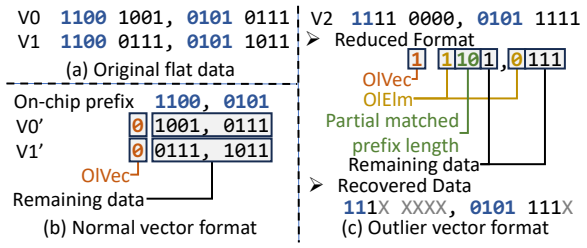

Figure 4: Offline common prefix elimination to save the cost of storing and accessing vector data. Outlier vectors are stored in place as in (c).

Such an optimized fetch scheme is non-trivial to identify. For instance, in datasets like DEEP and GIST, the highest 4 bits after the sign bit of each element are often the same and contribute little discrimination for early termination, making simple sequential fetch suboptimal. While the optimal scheme would generally depend on the dataset characteristics like data distribution, vector formats, and distance metrics, we discover some general trends across datasets. As we fetch from high to low bits, we care about the bit prefix of each element value. For each prefix length, we define the prefix entropy as $-\sum_i P_i \log P_i$ where P_i is the frequency of the ith prefix value appearing in the dataset, so it quantifies how diverse the prefix pattern is. We also define the early termination frequency as how often this prefix length can give a distance lower bound that is higher than the threshold (the threshold selection and other details are described below). Figure 3 illustrates the prefix entropy and early termination frequency for each prefix length on several datasets. We make three observations. First, the highest few bits usually exhibit low entropy (yellow zone), meaning that they tend to be similar, sharing a common prefix. We call it the *low-entropy range*. Second, most early terminations happen in the middle of the bit range (pink zone, early termination frequency > 1%), i.e., in the *high-termination range*. In this region the bits become diverse, leading to noticeably different estimated distance bounds that can be leveraged by early termination. Third, there are fewer early terminations happening within the lowest few bits (grey zone). Although the entropy remains high, these least significant bits have limited impact on the distance results.

Consequently, we have the following heuristics. First, we should quickly skip the common prefix in the low-entropy range, by moving over relatively more bits at the beginning. Second, we should move slowly across the high-termination range, in which every additional bit may trigger a successful termination. The exact number of bits in each fetch step, i.e., n_i, is determined through a *sampling-based* approach, where a small number of vectors in the actual dataset are used for two tasks: (1) deciding the (mostly) common prefix to eliminate when storing data, (2) deciding the bit chunk length in each fetch step. We empirically determine to use 100 vectors to balance sampling cost and accuracy, which is further discussed in Section 7.3. Next, we describe the two tasks in detail.

Offline common prefix elimination. Inspired by the quantization methods that compress vector data, we can directly omit the common prefix bits of each element, without even storing them in the new layout. Instead, a single copy of this common prefix is kept inside the on-chip compute logic, and concatenated to the fetched bits when calculating distances, as shown in Figure 4(b).

We use the sampling set to determine the length of the common prefix. However, the common prefix may not apply to every vector in the full dataset or in a dynamically changing dataset, i.e., there could exist outlier vectors [77]. To distinguish between normal and outlier vectors, we add one `OlVec` bit per vector, thus the actual saved space is (common prefix length × vector dimension − 1) bits.

Outlier vectors are stored in place, using a special format as in Figure 4(c). Since not all elements of a vector are outliers, we further use a per-element `OlElm` bit. Then, for those outlier elements, we use the next log(common prefix length) bits to denote how many bits of this outlier element match the common prefix. The remaining bits are used to store the bits starting from the mismatched position. In the figure, the prefix of V2 (1111_2) has two same bits as the common prefix 1100_2, so the partially matched prefix length is 2, and the remaining space stores the next bit 1. Due to the extra metadata, several lowest bits of the outlier element are dropped. These bits should be recovered similarly to the missing bits as discussed in Section 4.1 when calculating distances, to ensure a conservative lower bound. Solely using this format would slightly sacrifice accuracy (Section 7.3). If we want to ensure no accuracy loss, we can store the non-compressed original vector in a separate place. When the compressed vector gives an in-bound result, we re-check the non-compressed vector to filter out false positives, at the expense of a few additional memory accesses. In this paper we adopt this approach. Alternatively, the offline common prefix elimination is optional and can be disabled to ensure accuracy.

Furthermore, even in the sampling set, we can also allow a small number of outliers for a longer (mostly) common prefix and higher savings for most of the normal vectors. Again we empirically select the fraction of outlier elements in the sampling set as 0.1% (Section 7.3), i.e., the number of individual outlier dimension elements is no more than (0.1% × number of vectors × vector dimension).

Compared to traditional data quantization that inevitably affects accuracy, our common prefix elimination enables flexible tradeoff between efficiency and accuracy, even supporting no accuracy loss, by adjusting the threshold for outliers. In addition, it is designed for our early termination scheme, with which traditional quantization is incompatible.

Dual-granularity fetch. To decide the bit step n_i of each fetch, we again rely on our previous observations and use the following heuristic of *dual granularities*. That is, after the common prefix, we first fetch with a coarse-grained bit step n_C for T_C times (to quickly move over the remaining low-entropy range), and then switch to a fine-grained bit step n_F in the high-termination range (to terminate at the earliest with as few fetched bits as possible). Using such different granularities is more efficient than simple uniform data fetches. We have also explored more complex schemes for n_i, such as an arbitrary non-increasing series of bit steps with more than two granularities, but found limited extra benefits.

To find the optimized values of n_C, n_F, T_C parameters, we again use the sampling set. More specifically, for a carefully selected threshold (see below), we find the early termination position p_{ET} (i.e., the first bit position at which using all bits before it would trigger an early termination) of each vector in the sampling set. Then for a specific set of n_C, n_F, T_C values, we calculate the amount

of data needed to fetch. For example, if p_{ET} is in the fine-grained fetch range, the access cost is

$$64 \times \left(\left\lceil \frac{D}{m_C} \right\rceil \times T_C + \left\lceil \frac{D}{m_F} \right\rceil \times \left\lceil \frac{p_{\text{ET}} - n_C \times T_C}{n_F} \right\rceil \right) \quad (1)$$

where $m_{C,F} = \lfloor 64 \times 8 / n_{C,F} \rfloor$; D is the vector dimension. The ceiling functions consider the data layout padding overheads. We sum up the access cost over all the vectors in the sampling set. The n_C, n_F, T_C values that minimize this total access cost are chosen. Such parameter selection is done fully offline.

The final question is which threshold to use when doing the above parameter exploration. Strictly speaking, the threshold is determined by both the query vector and the index structure, and would vary along the search process. Nevertheless, in reality, the query vector should be close to some vectors in the database (e.g., in HNSW, it should be close to the entry point of the graph). So we use the distance distribution between pairs of vectors in the sampling set to approximate this threshold. Specifically, we empirically use the 10% largest distance (90% percentile) in this distribution, following the experimental results in Section 7.3.

4.3 Discussion

Packing multiple vectors in one fetch. Currently, we only do re-layout of bits within one vector. We may also consider packing multiple vectors in one fetch, which could further improve efficiency if the number of dimensions is small and/or the optimized fetch bit step is small. However, if not all the packed vectors are needed during index traversal, we may waste accessing unnecessary data in this case. Note that here only when all the packed vectors trigger early termination could we stop fetching, because they are in one single memory fetch. For example, in HNSW graphs, neighbors of a searched vector are usually all accessed next. But top-layer graph vertices typically have fewer neighbors than bottom-layer vertices. Even in the bottom layers with many neighbors for each vertex, there could be multiple traversal paths getting to a vector, i.e., it is a co-neighbor of multiple vertices, and may be packed with different sets of other neighbors. This makes it hard to statically do multi-vector packing during preprocessing. We thus do not explore it further in this work.

Compatibility with vector quantization. Early termination can also work under vector quantization, albeit with less efficiency. For scalar quantization, we can still estimate the missing bits/elements for the quantized data type, but quantization reduces the effectiveness of prefix elimination. For product quantization, partial bits of the codewords are not useful, but partial elements are beneficial. We can look up a subset of the memorized subspace distances for the partial elements and aggregate them for a distance lower bound.

5 Hardware Architecture

Next, we introduce the hardware architecture of ANSMET, which leverages DIMM-based NDP for ANNS and supports early termination. We first describe the overall architecture and execution flow in Sections 5.1 and 5.2. Then we discuss two specific issues, about how to partition the vector data across multiple ranks (Section 5.3), and how to synchronize the results between the host CPU and the NDP units (Section 5.4).

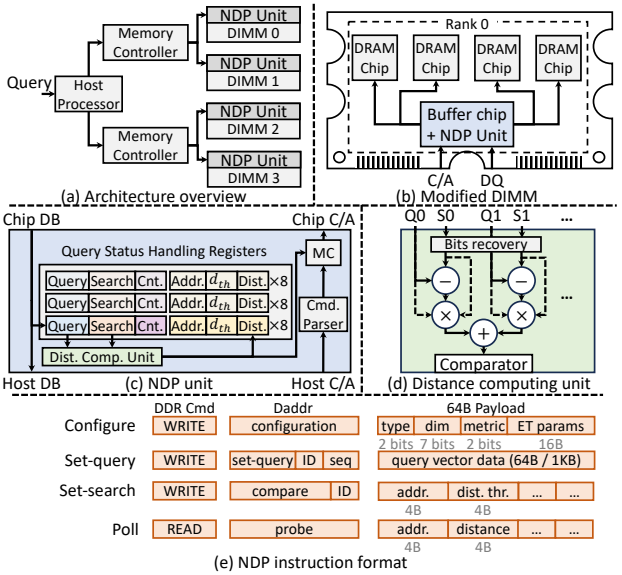

Figure 5: ANSMET hardware architecture. (a) Heterogeneous processing on host CPU and NDP units. (b) Modified DIMM with extra NDP unit. (c) NDP unit design. (d) Distance computing unit design. (e) NDP instruction format.

5.1 Architecture Overview

The search phase of ANNS includes two stages, index traversal and distance comparison. From Section 3 we see distance comparison is memory-bound and has high parallelism, and is thus well-suited to memory-side NDP. In contrast, index traversal typically exhibits limited parallelism and requires frequent control flow synchronization. Furthermore, in ANNS there are various types of index structures, including trees, hash tables, inverted lists, and graphs (Section 2.1). To support such flexibility, index traversal should use a general-purpose processor. Therefore, we run index traversal on the host CPU, and only execute distance comparison on the NDP units at the DIMM side, as in Figure 5(a). Such heterogeneity provides both generality and efficiency. We also separate the index data (the graph topology, the inverted list, etc.) and the vector data into different ranks to make host CPU accesses and NDP computations independent. Alternative ways that support concurrent accesses between host and NDP [16, 19, 33] may also be applicable.

The NDP units for distance comparison are added to the DIMM buffer chip, as in Figure 5(b). Figure 5(c) further shows the detailed design of an NDP unit, which contains a set of query status handling registers (QSHRs), a distance computing unit, and corresponding controllers for command parsing and generation. We use 32 QSHRs in each NDP unit. Each QSHR stores the metadata and status for the distance comparison tasks between one query vector and multiple vectors from the database under search (we call them "search vectors" hereafter). Specifically, each QSHR contains the query vector data, an array of 8 comparison tasks for this query (each including the search vector address, the distance threshold, and the result distance), the current vector data under comparison, and a fetch counter. We describe their usage in Section 5.2. The query and

search vector data fields are both 1 kB, which supports 256-dim FP16 or 512-dim UINT8 data types. Longer vectors are partitioned as described in Section 5.3. The other distance and address values are 4 B each. The overall capacity of QSHRs is 2148 bytes × 32 = 67.125 kB, a hardware cost similar to or smaller than previous designs [19, 73].

The distance computing unit in Figure 5(d) calculates the (lower-bound) distance between the query vector and a (partially fetched) search vector. We use 16 adders and multipliers, each up to 32-bit wide, to process multiple dimensions in parallel for each 64 B fetch. Fewer resources can be used to reduce area at the expense of longer compute latency, but we find this unnecessary for the DIMM buffer chip. Section 6 shows the NDP unit area is minor compared to the DIMM buffer chip total area. This unit is designed to support multiple distance metrics, akin to prior work [35]. For example, the adders are used to calculate the differences for the Euclidean distance but skipped for the inner product.

For implementation, our NDP units assume a unified buffer chip in the rank, consistent with prior NDP designs [42, 47]. To improve signal integrity, Load-Reduced DIMMs in both DDR4 and DDR5 remove the global buffer chip and instead use separate data buffers (DBs) and a register clock driver (RCD). We follow MEDAL [33] to implement NDP on this DIMM type. Specifically, a distance computing unit is added to each DB to compute a partial distance on the sub-vector from its DRAM chip. The result is sent through the new inter-chip hierarchical data bus added by MEDAL to the centralized RCD, which aggregates them and decides early termination.

5.2 Execution Flow

During ANNS, the index traversal runs on the host CPU. When it comes to the point at which multiple search vectors need to be compared with the query vector, the host CPU offloads distance comparison tasks to the NDP units on the DIMMs, through specially encoded DDR commands as the NDP instructions summarized in Figure 5(e). Initially, a *configure* instruction is issued to all NDP units, specifying the element type, the vector dimension, the distance metric, and the early termination parameters including the common prefix length and the n_C, n_F, T_C values. It is encoded as a DDR `WRITE` to a reserved address. To offload distance comparison, we use two instructions. A *set-query* instruction writes the query vector data (up to 1 kB) to a specific QSHR, with up to 16 DDR `WRITE` commands that each transfers 64 B. The QSHR ID and the sequence number are encoded in the address, with a reserved prefix. A *set-search* instruction sends to the same QSHR up to 8 comparison tasks, each with a 4-byte vector address and a 4-byte distance threshold, in one 64 B DDR `WRITE`. A QSHR that has received both instructions could start its processing. An optimization here is to first issue *set-search* and then *set-query*, so the NDP unit can start fetching the search vector data while receiving the query data.

The comparison tasks in the same QSHR are processed one after one, sequentially. However, different QSHRs can issue memory accesses in parallel to maximize bandwidth utilization. For each task, the NDP unit fetches each 64 B chunk from the transformed layout (Section 4.2) of its search vector stored in the local DRAM rank. The fetched data are restored to the original layout according to the early termination parameters configured previously, and written to the current vector data field of the QSHR. After each fetch, we compute the distance lower bound and decide whether to early terminate. The fetch counter is split into two sub-fields, tracking both which task is being processed and how many fetches have been done in the current task. If the final distance is within the threshold, we write it to the result field, which was initialized to an invalid `MAX` value. The host CPU polls the QSHR with the *poll* instruction, as a DDR `READ` to a specific QSHR ID. More details about polling are discussed in Section 5.4. Note that it is the host program's responsibility to allocate/free and keep track of QSHR usage, using their IDs explicitly in the above NDP instructions.

5.3 Hybrid Partitioning for Vector Data

In a typical memory system with multiple ranks, the vector data should be partitioned and placed across these ranks in an optimized manner, to fully utilize their NDP units in parallel. Prior DIMM-based NDP systems for recommendation systems have studied different schemes that can also be used by our system. The *vertical partitioning* method splits the dimensions of each vector and places each subset of dimensions into a different rank [42, 66]. To get one vector, all the ranks are accessed simultaneously, each calculating a partial distance result, and they are aggregated by the host CPU to derive the final distance. Vertical partitioning maximizes memory throughput, but it has several drawbacks. First, it has limited scalability as the partitioned subset of each vector becomes shorter and may not fully occupy the 64 B fetch granularity. Second, the host CPU needs to do additional partial result collection and aggregation, which is serialized across each rank. In contrast, the *horizontal partitioning* scheme splits different vectors into different ranks while all the dimensions of each vector stay in the same rank [4, 47]. Each distance comparison only accesses one rank, with a large access granularity of the full vector and without extra host CPU aggregation. However, when there are few on-the-fly comparison tasks, the other ranks may be underutilized. When the vectors in some ranks are hot, severe load imbalance could also limit the performance.

Given these tradeoffs, state-of-the-art solutions use *hybrid partitioning* that combines the two [76, 82, 91]. We follow this approach, which first splits each vector by dimensions into sub-vectors of size S and assigns them to a subset of ranks in one rank group (by vertical partitioning), and then partitions by different vectors among different rank groups (by horizontal partitioning). Previous designs use a relatively small sub-vector size of S = 64 B [82]. In our case, the use of early termination has unique impact on the above tradeoff. Specifically, early termination prefers horizontal partitioning, because with a longer (sub-)vector, early termination could save more future fetches of more dimensions. If the local rank only has a few dimensions, the partial distance result on them may not be enough to trigger early termination. Even if an early termination happens, it can only save future fetches on this local rank, but cannot affect other ranks due to lack of fast cross-rank communication. As a result, ANSMET should use a larger sub-vector size than previous designs. We find that S = 1 kB achieves the best balance (Section 7.3). We emphasize that we do not need to keep each full vector in a single rank; our design is compatible with vertical partitioning. Each sub-vector in each rank results in a partial distance that is eventually merged by the host CPU. It is still possible to

do early termination locally in each rank, by comparing its partial distance of the sub-vector to the full distance lower bound, albeit with reduced effectiveness.

Finally, we also replicate a small number of hot vectors to all rank groups, in order to alleviate load imbalance, which is the main drawback of horizontal partitioning with a larger sub-vector size. ANNS has a unique advantage over recommendation systems, which is also one of our novel insights. While hot vectors are somewhat arbitrary and hard to identify in recommendation systems, the index structure of ANNS can provide clear hints, e.g., the entry points and vertices in the top-layer graphs in HNSW, and the centroids in IVF. So hot vector replication is more effective in ANSMET.

Specifically, we evaluate the GIST dataset with the above replication technique. Without replication, the query amount ratio between the most loaded NDP unit and the average is 1.49×. By replicating the top four layers of the HNSW index, which are just 5.27 MB and 0.14% of the total data, this ratio reduces to 1.05×, indicating well balanced loads. We also generate a more skewed query set with the zipf distribution ($a = 2.0$). Replication can effectively reduce the above imbalance ratio from 2.19× to 1.09×.

5.4 Adaptive Result Polling

ANSMET intentionally separates distance comparison task offloading and result collection. *Polling* is used by the host CPU to retrieve the results of distance comparisons from the NDP units [28, 73, 92], due to several reasons. First, the NDP unit supports simultaneously processing multiple queries, which may affect each other's timing. Second, even for a single query, early termination may cause unpredictable latency that is shorter than full processing.

The key challenge is to determine when to poll, to balance between bandwidth overheads and unnecessary delays. In ANSMET, we observe that the early termination frequency in Figure 3 also serves as an indication for the probability distribution of distance computing time. Therefore we leverage this distribution obtained from the sampling-based preprocessing (Section 4.2) to calculate the expected latency of each offloaded task. For multiple tasks, we use the addition of their distributions, further considering their issue time differences. This significantly improves the polling efficiency.

Note that some previous designs proposed to modify the DDR protocol to support variable timing [3, 7, 48], and MEDAL [33] used additional RFU (Reserved for Future Use) pins to proactively notify the host CPU with a modified host memory controller. Our adaptive probing method does not need such modifications.

6 Methodology

Simulation setup. We build a cycle-accurate simulator for ANSMET through modifying Ramulator 2.0 [57]. The detailed configurations are summarized in Table 1. We use 4 memory channels and 2 DIMMs per channel. Each DIMM contains 4 ranks for parallel accesses. This configuration corresponds to previous DIMM-based designs [19, 33, 73]. The memory energy is derived from the Ramulator 2.0 DRAM power model. Other energy parameters are detailed in Table 1. The latency and area for the QSHR are derived from CACTI [5] under a conservative 22 nm technology node. Each NDP unit consumes 0.06 mm^2, which is acceptable compared to a 100 mm^2 typical area budget of the DIMM buffer chip according to existing DIMM-based NDP designs [19]. We have open-sourced the implementation of our simulator, including early termination and data layout optimizations, at https://github.com/tsinghua-ideal/ANSMET.

Table 1: System configurations.

Host CPU	16 out-of-order cores, of 3.2 GHz, 7 W per core 64 kB private L1-D/I cache, 8-way, 64 B cachelines, LRU 1 MB private L2 cache, 8-way, 14-cycle, LRU 8 MB shared LLC, 16-way, 60-cycle, LRU
Memory	DDR5-4800, 4 channels × 2 DIMMs × 4 ranks, 8 bank groups × 4 banks, RCD-CAS-RP: 40-40-40
NDP units	32 NDP units in total, one per rank, 1.2 GHz 32-entry QSHRs, 1-cycle lookup latency 16-wide 32-bit multipliers & adders, 300 mW

Table 2: ANNS datasets.

Dataset	Distance	Datatype	# Dims	# Vectors	# Queries
SIFT	L2	UINT8	128	1M	10K
BigANN	L2	UINT8	128	1B	10K
SPACEV	L2	INT8	100	1B	1K
DEEP	L2	FP32	96	1B	10K
GloVe	IP	FP32	100	1.2M	1K
Txt2Img	IP	FP32	200	1B	10K
GIST	L2	FP32	960	1M	1K

Evaluated designs. We select the following designs for comparison. CPU-Base uses the host CPU to process all operations and accesses the 4-channel memory system in the conventional way. NDP-Base adds 32 NDP units in the memory for distance comparison while keeping index traversal on the host CPU. We adopt the existing early termination schemes that only use partial-dimension fetches on NDP as NDP-DimET. We also compare with BitNN [32], which was designed for 3D point cloud kNN search and used bit-serial computations with bit-level early termination. We adapt their design to accelerate ANNS with a fixed 1-bit step (i.e., bit-serial), denoted as NDP-BitET.

We then enable our hybrid partial-dimension/bit early termination design, as NDP-ET. It uses a simple heuristic data layout strategy, where integers use 4-bit chunks and floating-point numbers use 8-bit chunks. This strategy does not need any sampling-based optimization but still needs to transform the data layout. We then apply dual-granularity fetch in NDP-ET+Dual. Finally, we add offline common prefix elimination in NDP-ETOpt, which becomes the full design of ANSMET. Similarly, we also evaluate CPU-ET and CPU-ETOpt. Hybrid partitioning is used for the vector data with the best sub-vector size of 1 kB for all the above designs.

Datasets. We conduct our evaluation on several billion-scale real-world datasets from public benchmarks [37, 38, 60, 67, 71]. The dataset characteristics are summarized in Table 2, ordered by their data element types and numbers of dimensions. We focus on the HNSW index algorithm in our evaluation [23, 58]. To construct the HNSW graph indexes, we set its parameter `efConstruction` to 500 and limit the maximum degree to 16 [25, 58]. We tune the other

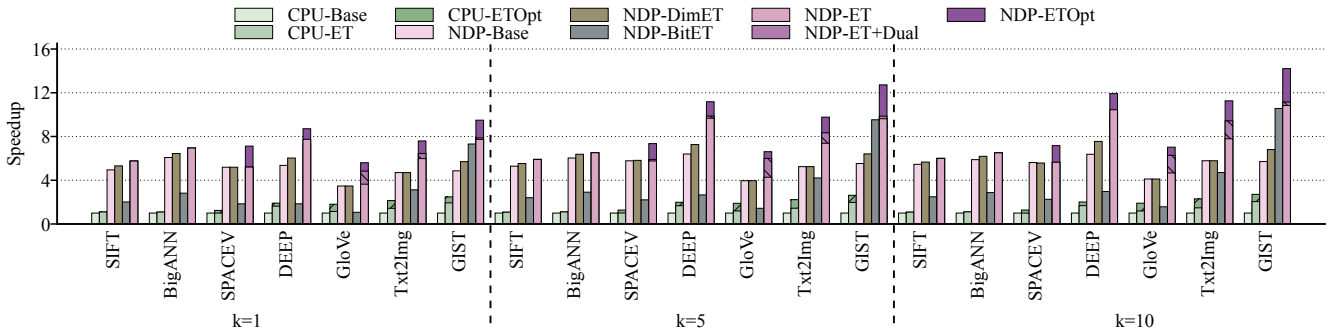

Figure 6: Performance comparison between ANSMET and the baselines with different datasets and result counts (k). Normalized to CPU-Base.

parameter `efSearch` (a.k.a., k' in Section 2.1) until the recall rate is over 80%, as recommended by previous work [25]. We further study the impact of k' in Figure 8.

7 Evaluation

In this section, we first compare the overall performance and energy consumption between ANSMET and the baselines in Section 7.1. Then we conduct detailed analysis of other performance metrics in Section 7.2. Finally we study the impact of various design parameters in ANSMET in Section 7.3.

7.1 Overall Comparison

We evaluate the CPU and NDP designs for HNSW-based ANNS workloads that find the k nearest neighbors, where we choose k as 1, 5, 10 for different use cases corresponding to previous work [35]. Figure 6 shows the speedups normalized to CPU-Base. Different k values exhibit similar results. For simplicity, in the following analysis, we use $k = 10$ as suggested in the BigANN benchmark [71].

Compared to the CPU, simply offloading distance comparison tasks to the NDP could significantly improve performance by leveraging intrinsic memory access parallelism, which brings a 5.26× speedup on average, and up to 6.40× for DEEP. This is expected as our rank-level NDP design has a theoretical 8× bandwidth increase over the CPU. NDP-DimET, the existing early termination using partial dimensions only, does not work for the datasets with the inner-product metric (GloVe and Txt2Img). This is because unfetched dimensions could contribute negative values, and thus a stable distance lower bound cannot be obtained. Even on the other datasets, NDP-DimET achieves only a 5.9% speedup compared to NDP-Base. NDP-BitET uses fixed single-bit early termination. It is only effective for datasets with sufficiently large dimensions, e.g., GIST. Otherwise, e.g., for SIFT with 128 dimensions, each fetch only contains 128 bits, which wastes 75% of a 64-byte access and performs even worse than NDP-Base.

Our hybrid partial-dimension/bit early termination, together with all the optimized data layout techniques (ETOpt), could save unnecessary memory accesses and computations, obtaining a 1.64× speedup on the CPU and 1.52× on the NDP. Note that the CPU early termination results are somewhat optimistic, as we assume similar dedicated logic to Figure 5(d) exists to avoid any bit recovery overheads. Below we mainly discuss the NDP designs. Among all the datasets, GIST achieves the highest speedup of 2.24× in NDP-ETOpt over NDP-Base, in which 87.3% distance comparisons are early terminated, and 53.0% memory accesses are saved. Both SIFT and BigANN are UINT8 datasets with low dimensionality, so our advanced bit-level optimizations including dual-granularity fetch and common prefix elimination are not very effective. But fetching partial bits in the simple way still improves performance by 10% compared to 5% in NDP-DimET. Generally, ANSMET shows greater improvements on datasets with 1) long bit widths (e.g., FP32) and high dimensions; 2) same or similar prefixes followed by high-entropy ranges, to enable dual-granularity fetch (e.g., DEEP and GIST as in Figure 3); 3) uniform query loads to avoid imbalance.

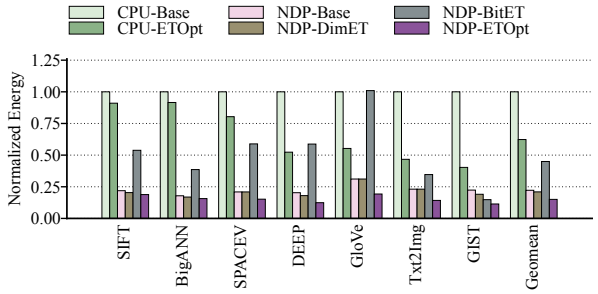

Figure 7: System energy comparison between ANSMET and the baselines on different datasets. Normalized to CPU-Base.

We further show the benefit breakdown from our data layout optimization techniques for early termination (Section 4.2). The simple heuristic in NDP-ET can reduce memory accesses by 25.1% on average. With dual-granularity fetch applied in NDP-ET+Dual, we can use adaptive coarse/fine-grained bit chunks at different ranges. It further reduces memory accesses based on the learned distribution of the dataset, and brings an 8.2% average speedup over the simple method. Eventually in NDP-ETOpt, we enable outlier-aware common prefix elimination to further increase fetch utilization, which brings another 18.8% speedup on average.

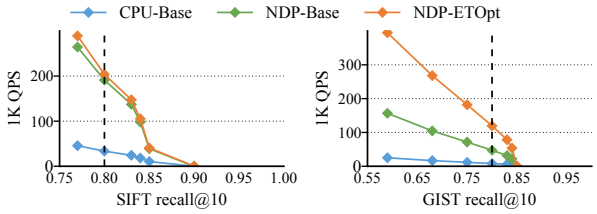

Figure 8: Recall vs. QPS curves for the SIFT and GIST datasets. Y-axis is the number of queries per second (QPS). X-axis is recall@10, i.e., the ratio of the exact k nearest neighbors included in the k ANNS results when $k = 10$. The vertical dash lines correspond to the cases in Figure 6 with 80% recalls.

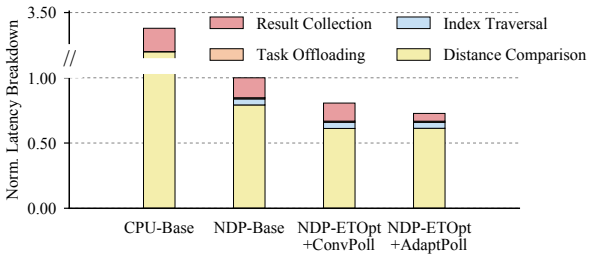

Figure 9: Latency breakdown in CPU-Base, NDP-Base, and NDP-ETOpt with conventional and adaptive polling. With the SIFT dataset. Normalized to NDP-Base.

Figure 7 shows the overall system energy comparison. NDP-Base is more energy-efficient than CPU-Base with 77.8% less energy. With fewer memory accesses, our simple early termination effectively reduces the system energy by 18.3%, and dual-granularity fetch and common prefix elimination further reduce the energy by 4.6% and 13.2%, respectively. We have also found that in the DEEP, Txt2Img, and GIST datasets, NDP-Base consumes higher memory energy than CPU-ET (not shown in the figure), because without early termination, the NDP design needs to access and process more data than the CPU despite having higher bandwidth. But the overall energy is lower because the NDP is more performant.

Figure 8 shows the tradeoff between accuracy and search time with different result queue sizes k'. We use the recall rate [35], i.e., how many true nearest neighbors are included in the approximate output. ANSMET consistently outperforms NDP-Base and CPU-Base at different accuracies. Besides the classic tradeoff between accuracy and time, a smaller k' also leads to a smaller distance threshold and makes early termination more effective. This explains the larger gaps between NDP-ETOpt and NDP-Base on the left side of the figure. Note that early termination improves performance without hurting accuracy.

7.2 Detailed Analysis

Figure 9 illustrates the latency breakdown of each query, in four designs: CPU-Base, NDP-Base, NDP-ETOpt with conventional polling of a simple fixed 100 ns interval, and with our adaptive polling (Section 5.4). NDP-Base significantly reduces the total per-query

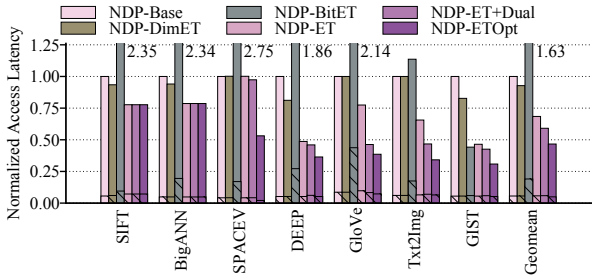

Figure 10: Normalized access latency breakdown into effectual (shaded regions) and ineffectual data fetches.

Table 3: Speedups of ANSMET over CPU-Base with different numbers of NDP units.

	CPU-Base	8	16	32	64
Speedup	1×	1.94×	3.72×	6.04×	7.60×

latency by 72.8% over CPU-Base, mainly from the faster distance comparison. Moreover, the early termination algorithm effectively reduces the distance comparison latency by saving unnecessary computations and data accesses, resulting in a 20% end-to-end latency reduction. However, the conventional polling method spends 13% time on collecting the results from the NDP units. By using our adaptive result polling, this overhead is reduced by 62% and the overall performance improves by another 6.8%. It has only 5.9% overheads compared to an ideal method with zero result collection and task offloading costs.

To see how early termination reduces unnecessary data fetches, Figure 10 illustrates the data fetch utilization improvements by attributing the total access latency to effectual (i.e., for accepted vectors) and ineffectual fetches. The fetch utilization improves from 6.0% in NDP-Base to 9.0% in NDP-ET, and eventually achieves 11.1% in NDP-ETOpt. The existing partial-dimension-only scheme obtains a 6.6% utilization. We see that even with NDP-ETOpt, there are still significant ineffectual data fetches. This gap is mainly due to the over-estimated distance bounds at the beginning of each query, during which the temporary nearest neighbors are still not close enough, and the early termination thresholds derived from them are too loose, yet to gradually converge.

ANSMET can adopt different numbers of parallel NDP units in the memory system. To investigate the scalability, we scale the NDP units and the corresponding ranks from 8 to 32 in Table 3. Overall, the performance scales linearly within 32 NDP units, obtaining speedups from 1.94× for 8 units to 6.04× for 32 units, compared to CPU-Base which has 4 channels. However, from 32 to 64 units, the speedup only slightly increases, due to the limited parallelism in the index algorithm itself, such as the average number of neighbors in the HNSW proximity graphs. Optimizations on the index structure, e.g., allowing more neighbors per node in HNSW, are outside the scope of this paper. At the architecture level, both horizontal and vertical partitioning schemes have scalability bottlenecks. With horizontal partitioning, load imbalance becomes more severe with

Table 4: Preprocessing time of different datasets.

Dataset	Preproc. time (s)	Graph constr. time (s)
SIFT	1.28	240.91
BigANN	44.95	>5000
SPACEV	0.38	4393.14
DEEP	24.48	1350.62
GloVe	12.67	3305.69
Txt2Img	27.66	4784.04
GIST	2.99	287.61

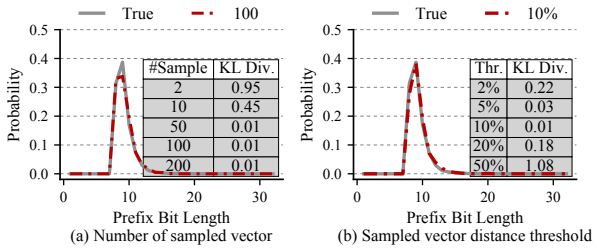

Figure 11: Impact of sampling parameters in preprocessing of early termination. (a) Number of sampled vectors. (b) Distance threshold as a percentile of all pair-wise distances. With the DEEP dataset. The figures compare the sampling distribution of early termination under the default parameter to the true distribution of the full dataset. The tables list the KL divergences between the true distribution and the distributions under different parameters.

more NDP units. With vertical partitioning, the final reduction at the CPU needs to collect more partial results from more NDP units. We leave optimizations for large-scale NDP units to future work.

Table 4 shows that the extra preprocessing time for data layout transformations in ANSMET is negligible, especially compared to the existing graph construction time, adding a small < 1% cost.

7.3 Parameter Sensitivity and Selection

Sampling parameters in preprocessing of early termination. We study the selection of two parameters in the sampling-based preprocessing of our early termination strategy: the number of sampled vectors, and the distance threshold (as a percentile in the distribution of all pair-wise distances in the sampling set). Their usage is described in Section 4.2. We compare the sampled distribution of early termination (similar to Figure 3) and the true distribution (obtained by performing real query vectors on the full dataset), using the Kullback–Leibler (KL) divergence which is a mathematical indicator to measure the difference between two probability distributions [46]. In Figure 11(a), we see that using more sampled vectors can get more close to the true case, and 50 to 100 vectors are sufficient for billion-scale benchmarks (sampling rate = 1e−7). Larger datasets may need slightly more samples. ANSMET uses 100 sampled vectors as default. In (b), we find that using the distance at the 10% percentile as the threshold shows the most similarity to the true case, with the smallest KL divergence. Here the threshold should be set as close to the true case as possible; either smaller or larger values would lead to more divergence. We also plot these distributions and see visually they are similar to the true distributions.

Table 5: Impact of outlier-aware common prefix elimination. With the SPACEV dataset at $k = 10$.

	Outlier %	0%	0.01%	0.1%	1%	20%
	Speedup	11.4%	11.4%	32.0%	32.0%	−1.1%
	Saved space	25.0%	25.0%	37.5%	37.5%	50%
(a)	**Extra space**	0%	0%	1.1%	1.1%	50%
	Extra accesses	0.0%	0.0%	1.4%	1.4%	68.3%
(b)	**Accuracy loss**	0%	0%	−34.7%	−34.7%	−76.5%

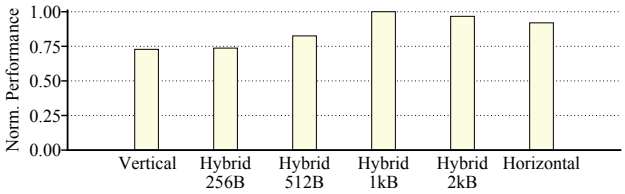

Figure 12: Impact of vector data partitioning schemes. With the GIST dataset. Normalized to the default of Hybrid 1 kB**.**

Outlier-aware common prefix elimination. Table 5 evaluates the benefits and overheads of being outlier-aware in the common prefix elimination process for different allowed fractions of outliers, compared to not using common prefix elimination. We use the SPACEV dataset as an example with $k = 10$. The rows of (a) use our default setting that stores non-compressed outlier vectors as backup to ensure no accuracy loss, while (b) allows accuracy loss to save more space. A lower fraction results in fewer outliers and fewer extra backup accesses, but also eliminates fewer bits and saves less memory space and data traffic. Nevertheless, even not allowing any outliers can still provide a 11.4% performance improvement, because 2 prefix bits of each element are eliminated in this case. On the other extreme, an overly high outlier fraction aggressively prunes the prefix, but there is a high probability of more extra accesses to backup outlier vectors. For example, with 20% outliers, the performance decreases by 1.1% because of 68.3% more extra accesses. In addition, the extra space to store these backup vectors could be substantial, totally offsetting the space saving. Our 0.1% choice has good space saving and performance speedup. The extra space and access overheads to backup vectors are small. However, if we do not keep the backup data, the accuracy could drop significantly, by 34.7%.

Hybrid partitioning of vector data. Figure 12 shows the impact of vector data partitioning schemes. Clearly, neither purely vertical nor purely horizontal partitioning achieves the optimal performance because they bias either parallelism with low fetch utilization, or high fetch utilization with long sequential accesses. The hybrid scheme with a 1 kB granularity works best for ANSMET, and is used as the default.

8 Related Work

NDP for recommendation systems. There have been numerous existing proposals using NDP architectures, mostly DIMM-based ones, to accelerate high-dimensional vector aggregation in recommendation system workloads [4, 42, 47, 55, 66, 82]. Our hardware architecture is similar to these designs. However, ANNS differs from recommendation systems in terms of its unique opportunity of early termination, which is leveraged by ANSMET with additional software and hardware optimizations.

ANNS acceleration. Some prior designs used commodity hardware to execute ANNS. For example, GPUs have been used to provide higher access bandwidth and exploit more parallelism [30, 40, 65, 88, 89]. DiskANN [72] and SpANN [14] utilized disk storage to provide sufficient capacity. However, GPU memory is usually too small, while disks are too slow. ANSMET uses DIMM-based NDP that achieves a good balance between data capacity and access performance for ANNS. FPGAs are also a promising platform for acceleration. ANNA [50] and Abdelhadi et al. [1] accelerated the product quantization implementation of ANNS using FPGAs. DF-GAS [83] and Falcon [39] proposed FPGA frameworks for end-to-end graph-based ANNS acceleration.

On the NDP regime, SSAM [49] was the first HMC-based NDP solution for ANNS, but 3D memory has limited capacity. CXL-ANNS [35] proposed a CXL-based system and offloaded computation tasks to the underlying CXL devices. Some designs, including VStore [54], Starling [78], NDSearch [79], Proxima [81], SmartANNS [75], and Kim et al. [44], utilized near-SSD computation. Nevertheless, none of these NDP designs exploited the opportunity of early termination as ANSMET does.

Other similarity search algorithms. Some designs [6, 12, 32, 68, 80] also accelerated large-scale low-dimensional vector searching with various indexes. BitNN [32] adopted bit-serial computations with bit-level early termination, targeting 3D point cloud applications. Vector databases have much more (hundreds of) dimensions than 3D, which motivates our hybrid partial-dimension/bit early termination and hybrid partitioning techniques. Some work incorporated vector search with conventional attribute matching named hybrid search [63, 85]. The distance computation can still be accelerated using ANSMET.

9 Conclusions

We proposed ANSMET, the first DIMM-based NDP system to accelerate ANNS with a novel early termination strategy. Using NDP alleviates the memory bottleneck of ANNS at the hardware level, and early termination further saves unnecessary data accesses and computations by identifying ineffectual vectors as early as possible at the algorithm level. We also optimize the data layout and CPU+NDP coordination carefully. ANSMET exhibits 5.26× and 1.52× speedups from NDP and early termination, respectively. These benefits are multiplicative to enable higher combined improvements.

Acknowledgments

The authors thank the anonymous reviewers for their valuable suggestions, and the Tsinghua IDEAL group members for constructive discussion. Mingyu Gao is the corresponding author.

References

[1] Ameer M. S. Abdelhadi, Christos-Savvas Bouganis, and George A. Constantinides. 2019. Accelerated Approximate Nearest Neighbors Search Through Hierarchical Product Quantization. In *2019 International Conference on Field-Programmable Technology (FPT)*. IEEE, 90–98. doi:10.1109/ICFPT47387.2019.00019

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *42nd International Symposium on Computer Architecture (ISCA)*. ACM, 105–117. doi:10.1145/2749469.2750386

[3] Mohammad Alian and Nam Sung Kim. 2019. NetDIMM: Low-Latency Near-Memory Network Interface Architecture. In *52nd International Symposium on Microarchitecture (MICRO)*. ACM, 699–711. doi:10.1145/3352460.3358278

[4] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung Kyu Lim, and Hyesoon Kim. 2021. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In *27th International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 908–920. doi:10.1109/HPCA51647.2021.00080

[5] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 14:1–14:25. doi:10.1145/3085572

[6] Pedro Henrique Exenberger Becker, José-María Arnau, and Antonio González. 2023. K-D Bonsai: ISA-Extensions to Compress K-D Trees for Autonomous Driving Tasks. *arXiv preprint arXiv:2302.00361* (2023).

[7] Brian Beeler. 2019. Intel Optane DC persistent memory module (PMM). *Retrieved March* 11 (2019), 2021.

[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517. doi:10.1145/361002.361007

[9] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving Language Models by Retrieving from Trillions of Tokens. In *39th International Conference on Machine Learning (ICML)*, Vol. 162. PMLR, 2206–2240.

[10] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 316–331. doi:10.1145/3296957.3173177

[11] Dan Chen, Haiheng He, Hai Jin, Long Zheng, Yu Huang, Xinyang Shen, and Xiaofei Liao. 2023. MetaNMP: Leveraging Cartesian-Like Product to Accelerate HGNNs with Near-Memory Processing. In *50th International Symposium on Computer Architecture (ISCA)*. ACM, 56:1–56:13. doi:10.1145/3579371.3589091

[12] Faquan Chen, Rendong Ying, Jianwei Xue, Fei Wen, and Peilin Liu. 2023. ParallelNN: A Parallel Octree-Based Nearest Neighbor Search Accelerator for 3D Point Clouds. In *29th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 403–414. doi:10.1109/HPCA56546.2023.10070940

[13] Patrick H. Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. 2023. FINGER: Fast Inference for Graph-Based Approximate Nearest Neighbor Search. In *32nd ACM Web Conference (WWW)*. ACM, 3225–3235. doi:10.1145/3543507.3583318

[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. In *35th Annual Conference on Neural Information Processing Systems (NeurIPS)*. 5199–5212.

[15] Rongxin Chen, Yifan Peng, Xingda Wei, Hongrui Xie, Rong Chen, Sijie Shen, and Haibo Chen. 2024. Characterizing the Dilemma of Performance and Index Size in Billion-Scale Vector Search and Breaking It with Second-Tier Memory. *arXiv preprint arXiv:2405.03267* (2024).

[16] Benjamin Y. Cho, Yongkee Kwon, Sangkug Lym, and Mattan Erez. 2020. Near Data Acceleration with Concurrent Host Access. In *47th International Symposium on Computer Architecture (ISCA)*. IEEE, 818–831. doi:10.1109/ISCA45697.2020.00072

[17] Kenneth L. Clarkson. 1994. An Algorithm for Approximate Closest-Point Queries. In *10th Symposium on Computational Geometry (SCG)*. ACM, 160–164. doi:10.1145/177424.177609

[18] Guohao Dai, Tianhao Huang, Yuze Chi, Jishen Zhao, Guangyu Sun, Yongpan Liu, Yu Wang, Yuan Xie, and Huazhong Yang. 2019. GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 4 (2019), 640–653. doi:10.1109/TCAD.2018.2821565

[19] Guohao Dai, Zhenhua Zhu, Tianyu Fu, Chiyue Wei, Bangyan Wang, Xiangyu Li, Yuan Xie, Huazhong Yang, and Yu Wang. 2022. DIMMining: Pruning-Efficient and Parallel Graph Mining on Near-Memory-Computing. In *49th International Symposium on Computer Architecture (ISCA)*. ACM, 130–145. doi:10.1145/3470496.3527388

[20] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast Locality-Sensitive Hashing. In *17th Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 1073–1081. doi:10.1145/2020408.2020578

[21] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *20th Symposium on Computational Geometry (SCG)*. ACM, 253–262. doi:10.1145/997817.997857

[22] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures. In *20th International World Wide Web Conference (WWW)*. ACM, 577–586. doi:10.1145/1963405.1963487

[23] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[24] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With the Navigating Spreading-Out Graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474. doi:10.14778/3303753.3303754

[25] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 2 (2023), 137:1–137:27. doi:10.1145/3589282

[26] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *24th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE Computer Society, 113–124. doi:10.1109/PACT.2015.22

[27] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 751–764. doi:10.1145/3093336.3037702

[28] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. 2021. Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture. *arXiv preprint arXiv:2105.03814* (2021).

[29] Otis Gospodnetic, Erik Hatcher, and Michael McCandless. 2010. *Lucene in action*. Simon and Schuster.

[30] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P. A. Lensch. 2023. GGNN: Graph-Based GPU Nearest Neighbor Search. *IEEE Transactions on Big Data* 9, 1 (2023), 267–279. doi:10.1109/TBDATA.2022.3161156

[31] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie. 2020. iPIM: Programmable In-Memory Image Processing Accelerator Using Near-Bank Architecture. In *47th International Symposium on Computer Architecture (ISCA)*. IEEE, 804–817. doi:10.1109/ISCA45697.2020.00071

[32] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Tianhao Cai, Jiale Xu, Yibo Wu, Chenhao Zhang, and Xiangrong Xu. 2024. BitNN: A Bit-Serial Accelerator for K-Nearest Neighbor Search in Point Clouds. In *51st International Symposium on Computer Architecture (ISCA)*. IEEE, 628–643. doi:10.1109/ISCA59077.2024.00095

[33] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. 2019. MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm. In *52nd International Symposium on Microarchitecture (MICRO)*. ACM, 587–599. doi:10.1145/3352460.3358329

[34] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *30th Annual ACM Symposium on the Theory of Computing (STOC)*. ACM, 604–613. doi:10.1145/276698.276876

[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 *2023 USENIX Annual Technical Conference (ATC)*. USENIX Association, 585–600.

[36] JEDEC. 2023. High Bandwidth Memory (HBM3) DRAM. https://www.jedec.org/standards-documents/docs/jesd238a.

[37] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57

[38] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in One Billion Vectors: Re-Rank with Source Coding. In *36th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 861–864. doi:10.1109/ICASSP.2011.5946540

[39] Wenqi Jiang, Hang Hu, Torsten Hoefler, and Gustavo Alonso. 2024. Accelerating Graph-Based Vector Search via Delayed-Synchronization Traversal. *arXiv preprint arXiv:2406.12385* (2024).

[40] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. doi:10.1109/TBDATA.2019.2921572

[41] Hongbo Kang, Yiwei Zhao, Guy E. Blelloch, Laxman Dhulipala, Yan Gu, Charles McGuffey, and Phillip B. Gibbons. 2023. PIM-trie: A Skew-Resistant Trie for Processing-in-Memory. In *35th Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 1–14. doi:10.1145/3558481.3591070

[42] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim M. Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *47th International Symposium on Computer Architecture (ISCA)*. IEEE, 790–803. doi:10.1109/ISCA45697.2020.00070

[43] Duckhwan Kim, Jaeha Kung, Sek M. Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *43rd International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 380–392. doi:10.1109/ISCA.2016.41

[44] Ji-Hoon Kim, Yeo-Reum Park, Jaeyoung Do, Soo-Young Ji, and Joo-Young Kim. 2023. Accelerating Large-Scale Graph-Based Nearest Neighbor Search on a Computational Storage Platform. *IEEE Transactions on Computers (TC)* 72, 1 (2023), 278–290. doi:10.1109/TC.2022.3155956

[45] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies. *BMC Genomics* 19, S2 (2018). doi:10.1186/s12864-018-4460-0

[46] Solomon Kullback and Richard A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86.

[47] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *52nd International Symposium on Microarchitecture (MICRO)*. ACM, 740–753. doi:10.1145/3352460.3358284

[48] Changmin Lee, Wonjae Shin, Dae Jeong Kim, Yongjun Yu, Sung-Joon Kim, and Taekyeong Ko. 2020. NVDIMM-C: A Byte-Addressable Non-Volatile Memory Module for Compatibility with Standard DDR Memory Interfaces. In *26th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 502–514. doi:10.1109/HPCA47549.2020.00048

[49] Vincent T. Lee, Amrita Mazumdar, Carlo C. del Mundo, Armin Alaghi, Luis Ceze, and Mark Oskin. 2018. Application Codesign of Near-Data Processing for Similarity Search. In *32nd International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 896–907. doi:10.1109/IPDPS.2018.00099

[50] Yejin Lee, Hyunji Choi, Sunhong Min, Hyunseung Lee, Sangwon Beak, Dawoon Jeong, Jae W. Lee, and Tae Jun Ham. 2022. ANNA: Specialized Architecture for Approximate Nearest Neighbor Search. In *28th International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 169–183. doi:10.1109/HPCA53966.2022.00021

[51] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *34th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[52] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *45th International Conference on Management of Data (SIGMOD)*. ACM, 2539–2554. doi:10.1145/3318464.3380600

[53] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-Based Product Retrieval in Taobao Search. In *27th Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 3181–3189. doi:10.1145/3447548.3467101

[54] Shengwen Liang, Ying Wang, Ziming Yuan, Cheng Liu, Huawei Li, and Xiaowei Li. 2022. VStore: In-Storage Graph Based Vector Search Accelerator. In *59th Design Automation Conference (DAC)*. ACM, 997–1002. doi:10.1145/3489517.3530560

[55] Haifeng Liu, Long Zheng, Yu Huang, Chaoqiang Liu, Xiangyu Ye, Jingrui Yuan, Xiaofei Liao, Hai Jin, and Jingling Xue. 2023. Accelerating Personalized Recommendation with Cross-Level Near-Memory Processing. In *50th International Symposium on Computer Architecture (ISCA)*. ACM, 66:1–66:13. doi:10.1145/3579371.3589101

[56] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110. doi:10.1023/B:VISI.0000029664.99615.94

[57] Haocong Luo, Yahya Can Tugrul, F. Nisa Bostanci, Ataberk Olgun, Abdullah Giray Yaglikçi, and Onur Mutlu. 2024. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 23, 1 (2024), 112–116. doi:10.1109/LCA.2023.3333759

[58] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473

[59] Micron. 2018. Hybrid Memory Cube – HMC Gen2. https://www.micron.com/-/media/client/global/documents/products/data-sheet/hmc/gen2/hmc_gen2.pdf.

[60] Microsoft. 2021. SPACEV1B: A billion-Scale vector dataset for text descriptors. https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B.

[61] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations (ICLR)*.

[62] Hadi Asghari Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *49th International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 50:1–50:13. doi:10.1109/MICRO.2016.7783753

[63] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-Throughput Vector Similarity Search in Knowledge Graphs. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 2 (2023), 197:1–197:25. doi:10.1145/3589777

[64] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 36, 11 (2014), 2227–2240. doi:10.1109/TPAMI.2014.2321376

[65] 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 *40th International Conference on Data Engineering (ICDE)*. IEEE, 4236–4247. doi:10.1109/ICDE60146.2024.00323

[66] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory. In *54th International Symposium on Microarchitecture (MICRO)*. ACM, 268–281. doi:10.1145/3466752.3480080

[67] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 1532–1543. doi:10.3115/V1/D14-1162

[68] Reid Pinkham, Shuqing Zeng, and Zhengya Zhang. 2020. QuickNN: Memory and Performance Optimization of k-d Tree Based Nearest Neighbor Search for 3D Point Clouds. In *26th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 180–192. doi:10.1109/HPCA47549.2020.00024

[69] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. 2012. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. In *18th Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 262–270. doi:10.1145/2339530.2339576

[70] Samsung. 2023. Samsung Unveils Industry's First 32Gbit DDR5 Memory Die: 1TB Modules Incoming. https://www.anandtech.com/show/20039/samsung-unveils-industrys-first-32gbit-ddr5-die-1tb-modules-incoming.

[71] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2021. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *NeurIPS 2021 Competitions and Demonstrations Track*, Vol. 176. PMLR, 177–189.

[72] Suhas Jayaram Subramanya, Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. 2019. Rand-NSG: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node. In *33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*. 13748–13758.

[73] Weiyi Sun, Zhaoshi Li, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. ABC-DIMM: Alleviating the Bottleneck of Communication in DIMM-Based Near-Memory Processing with Inter-DIMM Broadcast. In *48th International Symposium on Computer Architecture (ISCA)*. IEEE, 237–250. doi:10.1109/ISCA52012.2021.00027

[74] Boyu Tian, Qihang Chen, and Mingyu Gao. 2023. ABNDP: Co-optimizing Data Access and Load Balance in Near-Data Processing. In *28th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 3–17. doi:10.1145/3582016.3582026

[75] Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. 2024. Scalable Billion-Point Approximate Nearest Neighbor Search Using SmartSSDs. In *2024 USENIX Annual Technical Conference (ATC)*. USENIX Association, 1135–1150.

[76] Teng Tian, Xiaotian Wang, Letian Zhao, Wei Wu, Xuecang Zhang, Fangmin Lu, Tianqi Wang, and Xi Jin. 2022. G-NMP: Accelerating Graph Neural Networks with DIMM-Based Near-Memory Processing. *Journal of Systems Architecture* 129 (2022), 102602. doi:10.1016/J.SYSARC.2022.102602

[77] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions. *IEEE Data Engineering Bulletin* 47, 3 (2023), 39–54.

[78] 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. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 1 (2024), 1–27. doi:10.1145/3639269

[79] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, Hai (Helen) Li, and Yiran Chen. 2024. NDSEARCH: Accelerating Graph-Traversal-Based Approximate Nearest Neighbor Search through Near Data Processing. In *51st International Symposium on Computer Architecture (ISCA)*. IEEE, 368–381. doi:10.1109/ISCA59077.2024.00035

[80] Tiancheng Xu, Boyuan Tian, and Yuhao Zhu. 2019. Tigris: Architecture and Algorithms for 3D Perception in Point Clouds. In *52nd International Symposium on Microarchitecture (MICRO)*. ACM, 629–642. doi:10.1145/3352460.3358259

[81] Weihong Xu, Junwei Chen, Po-Kai Hsu, Jaeyoung Kang, Minxuan Zhou, Sumukh Pinge, Shimeng Yu, and Tajana Rosing. 2023. Proxima: Near-storage Acceleration for Graph-Based Approximate Nearest Neighbor Search in 3D NAND. *arXiv preprint arXiv:2312.04257* (2023).

[82] Sungmin Yun, Hwayong Nam, Kwanhee Kyung, Jaehyun Park, Byeongho Kim, Yongsuk Kwon, Eojin Lee, and Jung Ho Ahn. 2024. CLAY: CXL-Based Scalable NDP Architecture Accelerating Embedding Layers. In *38th International Conference on Supercomputing (ICS)*. ACM, 338–351. doi:10.1145/3650200.3656595

[83] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, and Yu Wang. 2023. DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-Based Approximate Nearest Neighbor Search. In *56th International Symposium on Microarchitecture (MICRO)*. ACM, 283–296. doi:10.1145/3613424.3614292

[84] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. 2018. GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition. In *24th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, 544–557. doi:10.1109/HPCA.2018.00053

[85] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th Symposium on Operating Systems Design and Implementation(OSDI)*. USENIX Association, 377–395.

[86] Yaodong Zhang and James R. Glass. 2011. An Inner-Product Lower-Bound Estimate for Dynamic Time Warping. In *36th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 5660–5663. doi:10.1109/ICASSP.2011.5947644

[87] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. 2023. Fast, Approximate Vector Queries on Very Large Unstructured Datasets. In *20th Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 995–1011.

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

[89] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *36th International Conference on Data Engineering (ICDE)*. IEEE, 1033–1044. doi:10.1109/ICDE48307.2020.00094

[90] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. 2023. Learned Probing Cardinality Estimation for High-Dimensional Approximate NN Search. In *39th International Conference on Data Engineering (ICDE)*. IEEE, 3209–3221. doi:10.1109/ICDE55515.2023.00246

[91] Zhe Zhou, Cong Li, Xuechao Wei, Xiaoyang Wang, and Guangyu Sun. 2022. GNNear: Accelerating Full-Batch Training of Graph Neural Networks with Near-Memory Processing. In *31st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 54–68. doi:10.1145/3559009.3569670

[92] Zhe Zhou, Cong Li, Fan Yang, and Guangyu Sun. 2023. DIMM-Link: Enabling Efficient Inter-DIMM Communication for Near-Memory Processing. In *29th International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 302–316. doi:10.1109/HPCA56546.2023.10071005

[93] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. 2019. GraphQ: Scalable PIM-Based Graph Processing. In *52nd International Symposium on Microarchitecture (MICRO)*. ACM, 712–725. doi:10.1145/3352460.3358256