Prism: Reducing Arithmetic for High-Recall Approximate Nearest Neighbor Search on Processing-in-Memory

Weihan Kong[1], Shengan Zheng[1*], Yingxue Zhou[1], Yifan Hua[2], Yuheng Wen[1], Cong Zhou[1], Guifeng Wang[1], Linpeng Huang[1*]
[1]Shanghai Jiao Tong University [2]Peking University
{weihankong,shengan,zhouyingxue,wenyuheng,cong258258,wangguifeng,lphuang}@sjtu.edu.cn,{huayifan}@pku.edu.cn

Abstract

Approximate Nearest Neighbor Search (ANNS) at scale is constrained by the memory wall. By moving compute to memory, Processing-in-Memory (PIM) offers ample internal bandwidth but limited on-die compute, making arithmetic reduction crucial. We propose Prism, a PIM-based ANNS system co-optimizing vector pruning, distance evaluation, and host-PIM orchestration. It employs a proximity-aware vector pruner to leverage high intra-PIM bandwidth and dual-cluster affiliations to filter out distant vectors. Prism then performs sensitivity-ordered distance computation, prioritizing high-impact dimension segments and early terminating candidates once exclusion criteria are met. A stall-free host-PIM pipeline overlaps query preparation, PIM execution, and global ranking to sustain high throughput. Experiments show that Prism achieves 2.7-19.8× higher throughput over state-of-the-art systems.

Keywords

Approximate Nearest Neighbor Search, Early Termination, Processing-in-Memory, Vector Database.

1 Introduction

High-dimensional vector representations of complex and unstructured data have become a cornerstone of large-scale AI systems. From personalized recommendation for social media platforms [4, 35] to Retrieval-Augmented Generation (RAG) that retrieves knowledge during LLM inference [9, 31], vector search is the key enabler for finding semantically relevant context to strengthen the quality and robustness of upstream applications [30, 39]. Nevertheless, performing exact nearest neighbor search over high-dimensional datasets at scale is prohibitively expensive due to the curse of dimensionality [18]. Consequently, Approximate Nearest Neighbor Search (ANNS) has emerged as the de facto solution, offering a pragmatic trade-off between search accuracy and efficiency [28, 35].

Despite its wide adoption, ANNS remains bottlenecked by the scale of vector datasets and the limited memory bandwidth of CPU-centric platforms. Serving an ANNS query typically involves fetching numerous vectors from DRAM with random, fine-grained accesses, undermining locality and thus rendering conventional caching and prefetching largely ineffective. Even with ample cores and powerful SIMD processing capabilities, CPUs cannot compensate for this shortfall as DRAM bandwidth scales slowly. The widening processor-memory gap becomes the dominant bottleneck, leading to the notorious "memory wall" problem [11, 27].

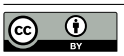
Processing-in-Memory (PIM) [8, 29] has emerged as a promising computing paradigm to address the memory wall by colocating computation with data and reducing data movement across the memory hierarchy. By executing computations inside or near memory, PIM is particularly suitable for workloads that exhibit poor cache locality [12] or high data intensity [15]. With UPMEM's commercialization [34], PIM has progressed beyond prototypes [1, 12, 16], and recent works have demonstrated broad effectiveness across diverse data-intensive domains [2, 22, 23]. For ANNS, high intra-module bandwidth streams large volumes of vector data efficiently, while in-memory execution reduces sensitivity to irregular access patterns. Consequently, PIM provides an unprecedented opportunity for accelerating ANNS at scale.

Despite these advantages, fully unleashing the potential of PIM for ANNS is non-trivial and faces two key challenges. The first is the mismatch between computational capability and memory bandwidth. PIM modules are inherently memory-centric: they offer high intra-module bandwidth, yet their on-die cores are relatively weak and data cannot be shared across modules. Since ANNS entails a vast number of distance calculations among vectors, massive arithmetic pressure can overwhelm the PIM cores. The performance bottleneck thus shifts from memory bandwidth to computation, undermining the bandwidth advantage of PIM.

The second challenge lies in balancing efficiency and accuracy in PIM-based ANNS. Mainstream ANNS approaches fall into two categories: graph-based and cluster-based. Graph-based methods, with high inter-node connectivity [26], incur substantial inter-PIM communication overhead because cross-module data exchanges are mediated by host on UPMEM [25]. Cluster-based approaches, in contrast, are more PIM-friendly by confining search to a handful of nearest clusters, improving locality and minimizing cross-module communication. Yet, the subsequent exhaustive distance computation within those clusters remains substantial. While quantization techniques [10, 20] alleviate the arithmetic pressure, their approximate representation inevitably degrades search recall.

To address these challenges, we propose Prism, a PIM-based ANNS system that strives to reduce PIM-side computation without compromising recall. Prism leverages query-dependent analysis on host to guide PIM-side vector pruning and optimized computation, reducing both the vector volume for distance evaluation and the evaluation cost per vector. Meanwhile, the host orchestrates the end-to-end processing workflow through a stall-free pipeline that overlaps host-side query preparation and global ranking with

*Shengan Zheng and Linpeng Huang are corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804038

PIM execution. By pipelining consecutive query batches, Prism sustains continuous host-PIM collaboration and significantly enhances overall resource utilization.

Prism employs a proximity-aware vector pruner to reduce the number of vectors subjected to distance computation. Clusters are uniformly sharded into strips and distributed across PIM modules to balance load. Given a query, Prism ranks candidate clusters by centroid-query distances and partitions them into central and peripheral sets. Vectors in central clusters are retained, while vectors in peripheral clusters are filtered using dual-cluster affiliations established during offline clustering. While scanning its local peripheral cluster strips, each PIM module retains a vector only when both of its affiliated clusters are in the candidate set, indicating proximity to the query. The proximity-aware pruning substantially reduces the number of vectors before any PIM arithmetic.

After pruning, Prism performs sensitivity-ordered distance computation. Vector dimensions are split into fixed-length segments, and the distance computation proceeds in descending order of segment sensitivity, prioritizing segments that are expected to contribute most to the distance. As the partial sum accumulates, it is compared with the threshold to early terminate candidates guaranteed to be excluded from the result set, eliminating redundant PIM arithmetic without compromising recall.

To summarize, we make the following contributions:

(1) We design a proximity-aware vector pruner that exploits PIM's high intra-module bandwidth to filter out distant vectors before distance evaluation while preserving likely nearest neighbors.
(2) We propose sensitivity-ordered distance computation that evaluates fixed-size segments in descending significance and terminates a vector once it is guaranteed to be excluded from the result set, thereby reducing redundant computation without compromising recall.
(3) We present a stall-free synergistic pipeline that overlaps host-side preparation and global ranking with PIM execution, ensuring sustained high resource utilization and high throughput.
(4) We implement Prism on the commodity PIM system UPMEM [34]. In our evaluation on Sift100M and SpaceV100M, Prism achieves 2.7-19.8× higher throughput over state-of-the-art ANNS baselines on CPU [7] and UPMEM [3, 37].

2 Background

2.1 Approximate Nearest Neighbor Search

The objective of nearest neighbor search is to retrieve the vectors closest to the query in a high-dimensional space. Exact nearest neighbor search guarantees perfect accuracy, but its linear scan complexity is computationally intractable for massive datasets. Consequently, Approximate Nearest Neighbor Search (ANNS) has become the de facto standard, trading a marginal reduction in accuracy for substantially improved search latency. ANNS therefore serves as a foundational component in applications such as information retrieval [24], recommendation systems [4, 35], and modern Retrieval-Augmented Generation (RAG) [9, 31] systems.

Graph-based methods [14, 19, 26, 36, 38] construct a proximity graph where vertices represent vectors and edges connect neighbors, facilitating ANNS through graph traversal. While offering high recall, this design entails substantial memory overhead due to storing auxiliary links. More importantly, their irregular neighbor expansions incur significant inter-module communication when deployed on PIM architectures. Cluster-based methods [5, 7, 33, 37] partition the dataset by grouping nearby vectors into the same cluster to narrow the search scope. The search follows a two-stage paradigm: first identifying candidate clusters by locating centroids nearest to the query, and then exhaustively scanning all vectors within them. Their primary limitation, however, is the intense computational workload. Since cluster-level pruning is coarse-grained, the system still needs to compute distances for a vast number of candidate vectors, dominating the total query cost. To mitigate this, Product Quantization (PQ) [20] is widely adopted to compress vectors and replace full-precision calculations with table lookups. Nevertheless, such compression inevitably introduces quantization errors, leading to degraded recall.

Regardless of the underlying search paradigm, ANNS remains intrinsically memory-bound for two fundamental reasons. First, ANNS suffers from an extremely low compute-to-memory access ratio [37]. This intrinsic limitation imposes a significant imbalance: while modern CPUs equipped with SIMD support a potential vector processing throughput of 500-1500 GB/s [6], traditional DRAM bandwidth remains a bottleneck at around 200 GB/s [37]. Second, the vectors accessed during search demonstrate negligible temporal locality. Once the distance between a candidate and the query vector is evaluated, the candidate is immediately discarded, thereby precluding effective caching or prefetching. The combination of continuous transfers of large numbers of vectors from DRAM and irregular access patterns leads to poor cache locality and prevents CPU-centric systems from fully exploiting their computational capability.

2.2 Processing-in-Memory

The introduction of Processing-in-Memory (PIM) [8, 29] directly confronts the memory wall by colocating computation with data storage. PIM integrates lightweight processors inside or near memory arrays, allowing direct data operations with reduced latency, making it ideal for memory-bound workloads. A typical PIM system consists of a host CPU and multiple PIM modules working collaboratively. The host, equipped with a powerful CPU, large caches, and ample DRAM, handles task scheduling and manages overall control. Meanwhile, each PIM module contains an on-die processor with limited compute capability, small caches, and local memory, enabling fast and parallel access to in-situ data. By coordinating with the host, PIM modules can receive executable code, perform local computation, and exchange results efficiently.

Currently, UPMEM [34] is the only commercially available PIM platform. Each PIM module contains a general-purpose in-order DRAM Processing Unit (DPU) paired with 64 MB of local MRAM. The DPU is designed as a 14-stage pipelined processor supporting up to 24 hardware threads [17]. A server can accommodate up to 20 UPMEM DIMMs, providing an aggregate intra-bandwidth of 1.28 TB/s [13]. Despite this impressive bandwidth, the computational capability of each DPU is relatively limited. Even when fully pipelined, the maximum throughput of INT32 ADD/SUB operations is below 60 MOPS [13], whereas a modern CPU with SIMD can achieve theoretical performance measured in TOPS. As a result, for

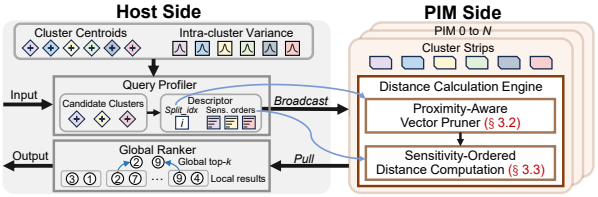


Figure 1: Overview of Prism.

computation- and data-intensive workloads such as ANNS, naively offloading the entire search task to PIM may shift the bottleneck from bandwidth to on-die computation. Without careful host-PIM co-design, the overall system efficiency remains limited.

3 Design

This section presents Prism for batched ANNS, optimizing vector selection, distance calculation, and host-PIM coordination. Candidate vectors from host-selected clusters are first filtered by a PIM-side proximity-aware pruner to trim boundary vectors in peripheral clusters. The remaining vectors are then evaluated with sensitivity-ordered distance computation that triggers early termination when partial sums exceed the threshold. While PIM executes, the host finalizes the top-k results for the previous batch, yielding a stall-free pipeline with sustained throughput.

3.1 Prism Overview

Figure 1 presents the overall architecture of Prism, which consists of three tightly coordinated components: query profiler, distance calculation engine, and global ranker. Upon receiving a query, the host-side query profiler first selects the candidate clusters by ranking centroid-query distances. It then constructs a heuristic descriptor for the query by combining the precomputed intra-cluster variance with the query's segment-wise discrepancy. This descriptor contains two critical components: a *split_idx* that classifies clusters as central or peripheral for vector pruning (Section 3.2); and per-cluster sensitivity orders over dimension segments, which enable early termination (Section 3.3). Once constructed, the query and its descriptor are ready for PIM-side processing.

On the PIM side, Prism shards each cluster into uniformly sized strips and round-robins them across all PIM modules, ensuring each PIM module holds a portion of each cluster. With this balanced layout, queries and their descriptors are simply broadcast to all PIM modules, fully exploiting PIM parallelism without per-module dispatch overhead. On each PIM module, the distance calculation engine spawns multiple threads, each dedicated to a single query. Guided by the descriptor, each thread scans its local cluster strips, calculates vector distances, and maintains a compact heap of nearest local vectors. When local search finishes, the host pulls local results and performs a cache-efficient global ranking, avoiding inter-module synchronization on the PIM side. This guided search eliminates complex decision-making on PIM, enabling it to focus on distance calculations, thereby achieving higher throughput.

Prism accelerates PIM-side execution by curtailing the number of vectors to be evaluated and reducing the arithmetic per vector. To shrink the candidate vector set before distance computation, Prism employs proximity-aware vector pruning (Section 3.2) to filter out distant vectors in peripheral clusters. The candidate clusters are first classified into central and peripheral sets based on their proximity to the query. The vectors in central clusters are searched exhaustively, while those in peripheral clusters are filtered according to their dual-cluster affiliations. Specifically, Prism assigns each vector to its two nearest centroids during clustering, and a vector in a peripheral cluster is retained only if both of its affiliated clusters lie in the candidate set. The proximity-aware vector pruning trims the candidate vector set to avoid unnecessary distance calculations while preserving high recall.

For the remaining candidate vectors, Prism reduces the computational overhead through sensitivity-ordered distance computation (Section 3.3). Vector dimensions are partitioned into equal-sized segments, which are then processed according to the host-generated sensitivity order, so that dimensions with high discrepancy are prioritized. During distance evaluation, the running partial sum is compared with the maximum distance in the local heap; if it exceeds this threshold, the computation terminates immediately. By early rejecting vectors guaranteed to fall outside the results, Prism avoids full distance evaluations and substantially reduces computational burden on PIM cores.

Prism further improves overall throughput by a synergistic host-PIM stall-free execution pipeline (Section 3.4), which tightly coordinates the query profiler, distance calculation engine, and global ranker as one workflow. While PIM processes the current batch of queries, the host concurrently prepares the next batch and finalizes results from the previous batch. By overlapping their execution, Prism effectively eliminates pipeline bubbles and sustains high resource utilization on both host and PIM sides.

3.2 Proximity-Aware Vector Pruner

In cluster-based ANNS, conventional search approaches scan all vectors in the candidate clusters and compute their distances to the query. This exhaustive scanning introduces substantial redundancy because many vectors are geometrically distant from the query and are unlikely to reach the top-k set. The key objective, therefore, is to effectively prune these distant vectors before computation, thereby reducing the search workload while preserving high recall.

However, preserving recall presents a significant challenge, as it strongly depends on the proximity between a vector's assigned cluster and the query. For clusters near the query, all vectors including boundary vectors should be retained to preserve recall. For distant clusters, however, boundary vectors on the side away from the query are unlikely to contribute to the final top-k set. Hence, effective pruning requires both precise partitioning of candidate clusters and reliable indication of vector-query proximity.

Prism addresses this challenge by introducing a proximity-aware pruner. To capture each vector's position relative to its cluster, Prism assigns dual-cluster affiliations to vectors during offline clustering. When processing the query, Prism dynamically partitions candidate clusters into central and peripheral sets by centroid-query distance, and then selectively prunes vectors in peripheral clusters based on their dual-cluster affiliations.

Dual-Affiliation Vector Clustering. Prism begins clustering by performing k-means on the host to assign each vector to an initial cluster centroid. It then affiliates each vector with its second-nearest centroid, forming dual-cluster affiliations. Centroids are updated following the expansion of clusters with their affiliated vectors. Each vector stores its dual-cluster IDs as metadata, co-located with its replicas in PIM to accelerate lookups. This dual assignment

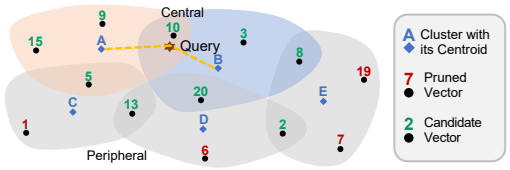


Figure 2: Proximity-aware vector pruning. Non-candidate clusters are omitted for clarity.

positions vectors between clusters to reveal their proximity to the shared boundary, providing a nuanced characterization of spatial position while striking a balance between pruning accuracy and storage overhead.

Central-Peripheral Cluster Partitioning. Accurate cluster partitioning is pivotal since it dictates both the vector-filtering efficiency and overall recall. Given a query, the host first ranks all clusters by their centroid-query distances and selects the n nearest clusters as candidates. It then partitions these candidates into central and peripheral sets based on their proximity to the query.

To determine the partition boundary, these n candidate clusters are first sorted by their centroid-query distances to the query vector, with d_i denoting the distance of the i-th cluster in ascending order ($d_1 \le d_2 \le \cdots \le d_n$). Prism then partitions these clusters based on a distance bound, d_{bound}, computed as a weighted average of these distances:

$$d_{\text{bound}} = \Big(\sum_{i=1}^{n} w_i \cdot d_i\Big) \Big/ \sum_{i=1}^{n} w_i, \tag{1}$$

where the weights w_i are defined iteratively to emphasize clusters that are closer to the query, with $w_1 = 1.0$ as the base weight:

$$w_{i+1} = w_i \cdot (d_i / d_{i+1}), \quad \text{for } i = 1, \ldots, n-1. \tag{2}$$

Once d_{bound} is established, the *split_idx* is set to the last index i where $d_i \le d_{\text{bound}}$, marking the boundary between central and peripheral candidates. Clusters with index up to *split_idx* are deemed central with all their vectors retained, whereas the remainder are peripheral and subject to proximity-aware pruning. This query-adaptive cluster partitioning leverages positional cues to balance recall and pruning efficiency.

Intersection-Based Vector Pruning. After cluster partitioning, Prism applies a proximity-aware pruner on PIM that leverages dual-cluster affiliations to filter each peripheral cluster. For any vector located within a peripheral cluster, if one of its affiliations falls outside the candidate set, it is considered distant and discarded. This indicates that the vector, though in a candidate cluster, is spatially anchored toward a region already deemed far from the query. Conversely, if both affiliated clusters are within the candidate set, the vector is likely to be spatially close to the query and is thus retained. By focusing on the intersection of candidate clusters, the proximity-aware pruning approach yields a compact candidate vector set, significantly reducing downstream distance computations.

Figure 2 illustrates a concrete example of cluster partitioning and vector pruning for a given query. The candidate set consists of 5 clusters (A-E) and the *split_idx* is 2. Hence, clusters A and B are designated as central, retaining all their vectors. For the peripheral clusters C, D, and E, the pruner examines each vector's dual-cluster affiliations. Vectors 1, 6, 7, and 19 are discarded because only one of their affiliated cluster IDs belongs to the candidate set. Conversely, vectors 13 and 2 are retained because both of their affiliations fall within the candidate set.

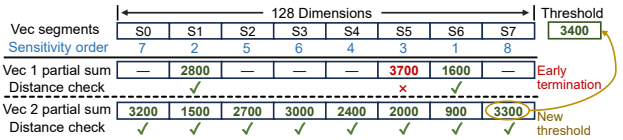


Figure 3: Adaptive early termination during distance calculation based on the sensitivity order.

Overall, the proximity-aware vector pruner filters out a large fraction of distant vectors at negligible cost. The query-vector proximity is approximated through host-side lightweight computations and PIM-side efficient memory lookups instead of costly distance calculation, exploiting high intra-PIM bandwidth without stressing compute. By shrinking the candidate vector set passed to distance computation (Section 3.3), Prism substantially reduces the arithmetic burden on PIM cores.

3.3 Sensitivity-Ordered Distance Computation

In ANNS, distance computation constitutes a substantial arithmetic workload, which involves calculating Euclidean distances between the query and each candidate vector. Despite applying proximity-aware vector pruning (Section 3.2), a substantial portion of vectors still require full distance calculation. Given PIM's constrained on-die compute resources (Section 2.2), such arithmetic becomes the dominant cost. Hence, identifying and eliminating unnecessary computations is essential for ANNS throughput.

During intra-cluster search, conventional distance calculation is performed via sequential, dimension-wise partial sum accumulation. Meanwhile, result selection compares the partial sum against a dynamic threshold, which is the current maximum distance in the result heap. This sequential method, however, often proves inefficient because many dimensions contribute disproportionately to the final distance [10], with only a few dominating the total. Prioritizing high-impact dimensions can accelerate the accumulation of the partial sum, meeting the termination condition earlier and reducing redundant computations.

To accelerate distance computation, Prism employs an adaptive early termination scheme guided by dimension sensitivity. For each query-cluster pair, the host ranks fixed-size dimension segments by sensitivity, reflecting their potential distance contribution. The sensitivity order is broadcast to PIM, which evaluates distances as ordered and halts once the partial sum exceeds the local threshold.

Dimension Segment Sensitivity Order. Prism partitions each vector's dimension into fixed-sized, non-overlapping segments, making segment-wise subvectors the granularity for Euclidean distance accumulation. To generate the segment-based sensitivity order for a given query and cluster, the host scores each segment by combining intra-cluster variance with query-centroid variance to estimate its expected impact on the query-vector distance. Prism then reorders the distance computation by their impact, prioritizing sensitive segments to trigger early termination once the partial sum exceeds the threshold.

Intra-cluster variance quantifies vector dispersion within a cluster on a per-segment basis. For a given segment, the variance is computed by averaging the squared segment deviations, where each deviation is the Euclidean distance between a vector's subvector and the centroid's subvector. Segments exhibiting higher variance tend to be more discriminative, amplifying Euclidean distance differences. Prism precomputes these variances offline during

clustering stage, storing them on the host side for retrieval at query time without additional runtime cost.

Query-centroid variance measures how far the query deviates from a cluster centroid on a per-segment basis. For each segment, it is the squared Euclidean distance between the corresponding segmented subvectors of the query and the cluster's centroid. A higher variance indicates greater segment-level discrepancy, suggesting a larger potential contribution to the final distance.

For each query-cluster pair, the host computes a per-segment sensitivity score by retrieving the intra-cluster variance, computing the query-centroid variance, and adding the two. It then sorts the segments by descending score to form the sensitivity order. The resulting order reflects the segments' expected contribution to query-vector distances of the cluster, and steers PIM to prioritize high-discrepancy segments during distance computation.

Distance Computation with Adaptive Early Termination. Following the sensitivity order, each PIM module's distance calculation engine evaluates candidate vectors through the segment-wise accumulation of their Euclidean distances to the query. After processing each segment, it compares the running partial sum against the maximum distance in its local result heap. If the partial sum exceeds this threshold, the vector is immediately discarded, avoiding further computation. Since the segments are processed in accordance with the sensitivity order, high-impact segments are evaluated first, increasing the likelihood of early termination. Otherwise, if the full distance ends below the threshold, the vector is inserted into the local heap, potentially updating the threshold for subsequent comparisons. The adaptive early termination thus dynamically trims unnecessary computations while preserving accuracy, significantly reducing arithmetic workload on PIM cores.

For example, as shown in Figure 3, with 128-dimensional vectors partitioned into 8 segments (S0-S7), the current maximum distance in the local result heap is 3400. Following the sensitivity order, vector 1 exceeds this threshold after the third processed segment (S5) and is discarded immediately, avoiding most of its remaining arithmetic. In contrast, vector 2 finishes at 3300 without early termination and enters the local result heap. Its insertion further tightens the threshold applied to subsequent comparisons.

To summarize, the sensitivity-ordered distance computation trims the arithmetic cost aggressively yet safely atop the candidate vector set already filtered by the proximity-aware vector pruner (Section 3.2). Its efficiency stems from tight host-PIM co-design that exploits the complementary strengths of both sides. The host undertakes the complex, query-specific analysis to build the per-segment sensitivity order, leveraging its powerful CPU and ample memory. PIM cores, in contrast, execute a lightweight, threshold-checking loop over local cluster strips following that order and terminate once the running partial sum exceeds the heap threshold. This adaptive early termination mechanism thus substantially reduces on-die pressure without compromising accuracy.

3.4 Stall-Free Execution Pipeline

The proximity-aware vector pruner (Section 3.2) and sensitivity-ordered distance computation (Section 3.3) substantially reduce the arithmetic performed in PIM, shifting the bottleneck to host-PIM coordination. During batched query processing, orchestration bubbles between stages can leave either side intermittently idle. To address this, we adopt a stall-free execution pipeline to effectively overlap host preparation, PIM computation, and host ranking, as illustrated in Figure 4. By overlapping these stages across adjacent batches, the pipeline eliminates inter-batch stalls and sustains high utilization on both host and PIM.

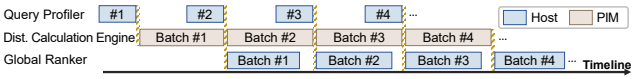


Figure 4: Stall-free pipeline coordinating host and PIM.

Stage 1: Host-side preparation. As PIM processes the current batch, the host-side query profiler proactively prepares the next batch. It identifies the candidate clusters and profiles each query to build a descriptor. The descriptor carries the *split_idx* and the sensitivity orders to guide PIM-side pruning and computation. Once the descriptors are ready, the host broadcasts the next batch of queries with their descriptors to PIM.

Stage 2: PIM-side execution. Upon receiving queries and their descriptors, each PIM module spawns multiple threads. For each query, a thread scans its local cluster strips, applying the proximity-aware vector pruner to filter out distant vectors in peripheral clusters. The thread then computes distances for the remaining vectors, following the sensitivity order to prioritize high-impact segments, and maintaining a local result heap. Partial sums are checked against the heap's maximum to trigger early termination. Upon completion, Prism signals the host for result retrieval. This parallelism fully exploits PIM's architecture to accelerate vector search.

Stage 3: Host-side ranking. In parallel with ongoing PIM execution, the host pulls each PIM module's local result from the prior batch. It then consolidates the vectors in the local heap from all PIM modules into a global result pool and sorts it by query-vector distance to produce the global top-k results.

In Prism's stall-free pipeline, the host alternates between the query preparation for the next batch and global vector ranking for the previous batch, while PIM executes distance calculation for the current batch. The division of labor keeps both host and PIM sides continuously occupied, maximizing inter-batch overlap and minimizing pipeline stalls. Combining vector pruning with optimized distance computation, Prism fully leverages the complementary strengths of both host and PIM, thereby substantially improving overall system throughput.

4 Evaluation

4.1 Experiment Setup

Testbed. Experiments are conducted on a dual-socket server with two Intel Xeon Silver 4216 CPUs (2.10 GHz, 22 MB L3 cache), 256 GB DDR4-3200 DRAM, and 20 UPMEM DIMMs. Each DIMM integrates 128 PIM modules, providing 2560 modules in total. Each PIM module integrates a DPU operating at 350 MHz with 24 hardware threads. The server runs Ubuntu 22.04 and Linux kernel 5.15.

Datasets. We conduct experiments on two datasets: Sift100M, which contains 100 million 128-dim uint8_t vectors sampled from SIFT1B [21], and SpaceV100M, which comprises 100 million 100-dim int8_t vectors sampled from SPACEV1B [32]. Prism partitions vectors into 8 segments for Sift100M and 10 segments for SpaceV100M.

Compared systems. We use Faiss-CPU [7] as the CPU baseline, a widely adopted ANNS library from Meta that represents a highly optimized CPU implementation. For PIM-based methods, we evaluate DRIM-ANN [3], which leverages batch processing and Product

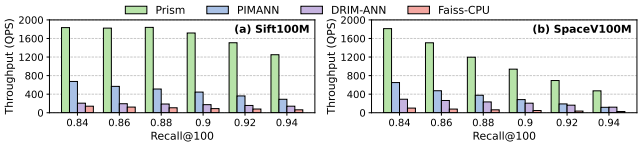


Figure 5: Throughput under different recalls.

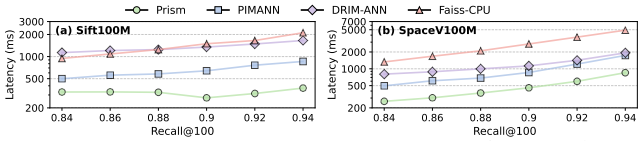


Figure 6: Latency under different recalls (log scale).

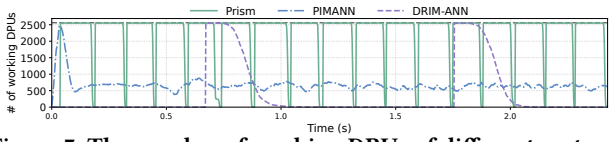


Figure 7: The number of working DPUs of different systems.

Quantization (PQ) [20], and PIMANN [37], which employs a fine-grained, per-DPU scheduling strategy, also built upon PQ [20]. We set the number of clusters to 4096, batch size to 128. For Prism, each PIM module produces 2 local results per query. All PIM-based systems use the same hardware configuration as our system, while the CPU components and Faiss-CPU both exploit SIMD parallelism.

Metrics. We report throughput (QPS) and latency at equivalent recall@100. For fair comparison, PQ-based methods are configured to have the same number of candidate clusters as Prism under high-recall settings. Their recall is measured by the overlap between their expanded result set and the ground-truth top-100 items.

4.2 Performance Under Different Recall

Figure 5 presents the throughput comparison among Prism, PIMANN, DRIM-ANN, and Faiss-CPU. Prism consistently outperforms all other systems across different recall levels. Compared with Faiss-CPU, Prism achieves a 13.0-19.8× speedup by effectively leveraging the high internal bandwidth of PIM. Against DRIM-ANN, Prism delivers a 3.9-9.9× speedup, attributed to the stall-free execution pipeline and the reduced computational burden from vector pruning and sensitivity-ordered distance computation. Compared with PIMANN, Prism attains a 2.7-4.3× gain. The fine-grained, per-DPU scheduling in PIMANN creates heavy coordination overhead, and its PIM-side result merging adds further pressure, ultimately constraining end-to-end throughput. In contrast, Prism simply broadcasts each batch of queries to all PIM modules and assigns dedicated reduced-computation to each PIM core, with negligible management overhead and consequently higher throughput.

Figure 6 presents the average query latency of Prism, PIMANN, DRIM-ANN, and Faiss-CPU. Prism achieves the lowest latency across all settings, because its evenly sharded data layout allows all PIM modules to participate in each query, fully exploiting parallelism and reducing batch-level latency. Compared with Faiss-CPU, it reduces latency by 65%-84%, as PIM's massive parallelism handles distance computations concurrently. Relative to DRIM-ANN, Prism achieves a 56%-80% reduction through a reduced on-die computational pressure and a highly load-balanced data placement strategy. Compared with PIMANN, latency is reduced by 34%-58%. The persistent-kernel design of PIMANN relies on a low-bandwidth control interface for data exchange. In addition, due to the accuracy loss from PQ, it must transfer more intermediate results to the host side to achieve the target recall. These two factors jointly lead to significant delays.

Notably, under the Sift100M configuration at Recall@100=0.90-0.94 in Figure 6(a), latency is unexpectedly lower than at lower recall levels. This is because the increased number of candidate clusters results in a higher computational load on PIM, which enhances PIM utilization. Meanwhile, the host-side load remains relatively stable, allowing for a more balanced host-PIM pipeline. This balanced workload enables the host to sustain higher PIM utilization, which in turn leads to lower end-to-end latency.

4.3 Working DPU Utilization

We define a working DPU as a PIM module's processing unit that is actively performing distance computation, excluding overheads like synchronization, scheduling, or data transfer. Unlike system-level QPS/latency metrics, which can be skewed by such non-computational overheads, this metric provides a more accurate reflection of PIM-side search efficiency.

We take the case on SpaceV100M at Recall@100 = 0.90 as a representative example, as shown in Figure 7. For PIMANN, working DPUs are initially abundant but fall off quickly as execution proceeds. This decline stems from its polling-based fine-grained scheduler: the host repeatedly scans for idle DPUs that hold replica slices of the query's candidate clusters, incurring long dispatch latency and severe load imbalance. Consequently, DPUs holding popular slices become overloaded, while the majority of others remain idle. DRIM-ANN, however, shows a gradual decline in working DPUs within each batch, followed by complete idleness between batches. Its cluster-based data distribution leads to intra-batch imbalance; meanwhile, the lack of a host-PIM pipeline leads to inter-batch imbalance, both leaving many DPUs underutilized over time. In Prism, working DPUs remain consistently high throughout execution. Our uniform cluster sharding and broadcast-based query dispatch together ensure a balanced load among DPUs, eliminating intra-batch idleness. Additionally, the stall-free execution pipeline eliminates inter-batch stalls, as DPUs begin computation immediately upon receiving a prepared batch of queries.

5 Conclusion

In this paper, we present Prism, a PIM-based ANNS engine that reduces arithmetic while preserving recall. Prism employs a proximity-aware vector pruner to filter out distant vectors before distance evaluation, and sensitivity-ordered distance computation that terminates vectors once exclusion is certain. A stall-free host-PIM pipeline overlaps host preparation, PIM execution, and global ranking, eliminating orchestration bubbles and sustaining high throughput. Evaluations demonstrate that Prism achieves significant performance improvements over state-of-the-art ANNS systems.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and insightful suggestions. This work is supported by the National Key Research and Development Program of China (Grant No. 2024YFB4505203), the National Natural Science Foundation of China (NSFC) (Grant Nos. 62227809, 62332012, 62302290), the Fundamental Research Funds for the Central Universities, and the Natural Science Foundation of Beijing (Grant No. 4264104).

References

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd annual international symposium on computer architecture*. 105–117.

[2] Shuangyu Cai, Boyu Tian, Huanchen Zhang, and Mingyu Gao. 2024. Pimpam: Efficient graph pattern matching on real processing-in-memory hardware. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[3] Mingkai Chen, Tianhua Han, Cheng Liu, Shengwen Liang, Kuai Yu, Lei Dai, Ziming Yuan, Ying Wang, Lei Zhang, Huawei Li, et al. 2024. DRIM-ANN: An Approximate Nearest Neighbor Search Engine based on Commercial DRAM-PIMs. *arXiv preprint arXiv:2410.15621* (2024).

[4] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, et al. 2022. Approximate nearest neighbor search under neural similarity metric for large-scale recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3013–3022.

[5] Sitian Chen, Amelie Chi Zhou, Yucheng Shi, Yusen Li, and Xin Yao. 2024. MemANNS: Enhancing Billion-Scale ANNS Efficiency with Practical PIM Hardware. *arXiv preprint arXiv:2410.23805* (2024).

[6] Rongxin Cheng, 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).

[7] 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).

[8] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical near-data processing for in-memory analytics frameworks. In *2015 International Conference on Parallel Architecture and Compilation*. IEEE, 113–124.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2, 1 (2023).

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

[11] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. Ai and memory wall. *IEEE Micro* 44, 3 (2024), 33–39.

[12] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2021. Syncron: Efficient synchronization support for near-data-processing architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 263–276.

[13] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. 2021. Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware. In *2021 12th International Green and Sustainable Computing Conference (IGSC)*. IEEE, 1–7.

[14] Hao Guo and Youyou Lu. 2025. Achieving Low-Latency Graph-Based Vector Search via Aligning Best-First Search Algorithm with SSD. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 171–186.

[15] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1718–1725.

[16] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. 2016. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *2016 IEEE 34th International Conference on Computer Design*. IEEE, 25–32.

[17] Bongjoon Hyun, Taehun Kim, Dongjae Lee, and Minsoo Rhu. 2024. Pathfinding future pim architectures by demystifying a commercial pim technology. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 263–279.

[18] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[19] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS:Software-Hardware collaborative memory disaggregation and computation for Billion-Scale approximate nearest neighbor search. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 585–600.

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

[21] Hervé Jégou, Laurent Amsaleg, Cordelia Schmid, and Patrick Gros. 2013. SIFT1B Dataset. http://corpus-texmex.irisa.fr/. Accessed: 2025-10-21.

[22] 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 *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*. 1–14.

[23] Weihan Kong, Shengan Zheng, Yifan Hua, Ruoyan Ma, Yuheng Wen, Guifeng Wang, Cong Zhou, and Linpeng Huang. 2025. PimBeam: Efficient Regular Path Queries Over Graph Database Using Processing-in-Memory. *IEEE Transactions on Parallel and Distributed Systems* (2025).

[24] Pau Perng-Hwa Kung, Zihao Fan, Tong Zhao, Yozen Liu, Zhixin Lai, Jiahui Shi, Yan Wu, Jun Yu, Neil Shah, and Ganesh Venkataraman. 2024. Improving embedding-based retrieval in friend recommendation with ann query expansion. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2930–2934.

[25] Ruoyan Ma, Shengan Zheng, Guifeng Wang, Jin Pu, Yifan Hua, Wentao Wang, and Linpeng Huang. 2024. Accelerating regular path queries over graph database with processing-in-memory. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[26] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[27] Sally A McKee. 2004. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*. 162.

[28] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.

[29] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2022. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*. Springer, 171–243.

[30] Ioannis Papadimitriou, Ilias Gialampoukidis, Stefanos Vrochidis, et al. 2024. RAG Playground: A Framework for Systematic Evaluation of Retrieval Strategies and Prompt Engineering in RAG Systems. *arXiv preprint arXiv:2412.12322* (2024).

[31] Derrick Quinn, Mohammad Nouri, Neel Patel, John Salihu, Alireza Salemi, Sukhan Lee, Hamed Zamani, and Mohammad Alian. 2025. Accelerating retrieval-augmented generation. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 15–32.

[32] Microsoft SPTAG Team. 2021. SPACEV1B: A Billion-Scale Vector Dataset for Text Descriptors. https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B Accessed: 2025-09-20.

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

[34] UPMEM. 2025. UPMEM Technology. https://www.upmem.com/technology/. Accessed: 2025-08-18.

[35] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).

[36] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, Yiran Chen, et al. 2024. NDSEARCH: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 368–381.

[37] Puqing Wu, Minhui Xie, Enrui Zhao, Dafang Zhang, Jing Wang, Xiao Liang, Kai Ren, and Yunpeng Chai. 2024. Turbocharge ANNS on Real Processing-in-Memory by Enabling Fine-Grained Per-PIM-Core Scheduling. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[38] 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).

[39] Hailin Zhang, Yujing Wang, Qi Chen, Ruiheng Chang, Ting Zhang, Ziming Miao, Yingyan Hou, Yang Ding, Xupeng Miao, Haonan Wang, et al. 2023. Model-enhanced vector index. *Advances in Neural Information Processing Systems* 36 (2023), 54903–54917.