

FlowIVF: Restructuring ANN Search for Spatial Dataflow Accelerators

Seongkyu Lee, Gangmin Lee, Sue Moon, and Gyuyeong Kim

Abstract—Approximate nearest neighbor (ANN) search is a core retrieval primitive in modern ML serving, and much prior work has accelerated it on high-cost GPUs. Emerging spatial dataflow accelerators provide a lower-cost platform with distributed on-chip SRAM and a network-on-chip, creating an opportunity for more cost-efficient ANN serving. However, adapting ANN search to these accelerators is nontrivial. These accelerators are efficient when work is fixed and scheduled across cores in advance, but in inverted-file (IVF) search, which lists to scan and which candidates to retain are known only at query time.

We present FlowIVF, an IVF execution system that restructures both sources of irregularity. FlowIVF converts variable-length inverted lists into fixed-size pages routed through a four-stage pipeline. Throughout this pipeline, centroids and intermediate states are SRAM-resident, and database pages are reused across queries to avoid redundant memory traffic. To resolve top- K 's irregular control, FlowIVF assigns each selection step to the on-core engine and merges results through on-chip reduction. We implement and evaluate FlowIVF end-to-end on a Tenstorrent Blackhole accelerator, achieving $10\text{--}50\times$ higher cost efficiency (QPS/\$) than GPU IVF-Flat across SIFT-1M and GIST-1M and $2\text{--}5\times$ even against compressed IVF-PQ, while maintaining single-digit-millisecond latency at batch size one.

Index Terms—approximate nearest neighbor search, inverted file index, spatial dataflow accelerator

I. INTRODUCTION

Approximate nearest neighbor (ANN) search is the core retrieval primitive over large-scale vector databases, avoiding the cost of exact search over high-dimensional embeddings [1]. It underlies a broad range of ML serving workloads, including retrieval-augmented generation (RAG) [2], semantic search [3], and recommendation [4]. As GPUs have a Single Instruction, Multiple Threads (SIMT) architecture with massive parallelism and high memory bandwidth suited for ML operations, recent work has utilized GPUs for ANN search to achieve higher performance than CPUs [5], [6].

This performance, however, comes at a steep price. Systems that scale ANN search by provisioning more GPU memory pay for it directly, since High-Bandwidth Memory (HBM) is the dominant factor in high-end GPU costs. Recent work addresses

this memory cost challenge by scaling ANN search primarily through inexpensive SSD bandwidth instead of GPU memory, achieving several times higher throughput per dollar [7]. This tension between raw performance and hardware cost has made cost efficiency a central design goal for large-scale ANN serving [8], [9].

The high cost of GPU memory has inspired novel architectures to emerge. CXL-based memory disaggregation extends GPU memory capacity over an external interconnect [8], while near-data processing (NDP) and in-storage accelerators move search computation directly onto storage devices [9], [10]. Another alternative is spatial dataflow accelerators. They replace the GPU's shared cache and expensive HBM with SRAM distributed to cores and connected via a network-on-chip (NoC) throughout the device, achieving high compute density [11]. These accelerators avoid the GPU's costly memory hierarchy by design and have already been deployed at scale for ML serving, offering substantially higher energy efficiency than GPUs [12], [13].

Yet realizing this potential for ANN search is nontrivial. Spatial dataflow accelerators assume that a workload's computation and communication are known in advance, so work can be split into fixed units and scheduled onto cores and the NoC ahead of time. Inverted-file (IVF) search, the most common ANN search approach, does not fit this workflow. When a query arrives, IVF search selects n_{probe} centroids and scans all vectors in the selected centroids' lists to find the top- K candidates. However, which lists a query touches, and how large they are, are only known once the query arrives. Simply porting a GPU-oriented implementation does not resolve this. GPU-based IVF systems execute each stage as a separate, batch-synchronous kernel over global memory. However, spatial dataflow accelerators lack this kind of global memory, and thus re-dispatching every stage requires round-trips to off-chip memory.

Building on this observation, we propose FlowIVF, which restructures IVF execution around distributed on-chip SRAMs. To resolve query-time list access, FlowIVF converts variable-length inverted lists of each centroid into fixed-size pages and streams them as bounded messages over the NoC through a persistent pipeline that consists of coarse-search, routing, fine-search, and reduction stages. This turns unpredictable candidate access into deterministic, core-local streaming. Centroids, metadata, and query state stay resident in SRAM to avoid DRAM access, and database pages are buffered and reused across a query window to amortize DRAM fetch over many candidate evaluations. To address the data-dependent control flow of top- K selection, FlowIVF splits selection

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (No. RS-2025-00522990) and by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02263378, Development of Heterogeneous AI Chip Virtualization Technology). (Corresponding author: Gyuyeong Kim.)

Seongkyu Lee, Gangmin Lee, and Sue Moon are with the School of Computing, KAIST, Daejeon 34141, South Korea (e-mail: {lsk108, lgm9, sbmoon}@kaist.ac.kr).

Gyuyeong Kim is with the School of Computing, Yonsei University, Seoul 03722, South Korea (e-mail: gyuyeong.kim@yonsei.ac.kr).

into stages assigned to different dedicated cores and merges results through on-chip reduction. Together, these mechanisms convert IVF's two sources of irregularity into bounded, on-chip-resident, and pipelined execution that aligns with the spatial hardware's execution model.

We profile IVF search on a spatial dataflow accelerator and show that, once distance computation is parallelized across cores, selection and coordination become the dominant performance bottlenecks. We propose FlowIVF, an SRAM-centric IVF execution architecture that keeps search state on-chip, communicates directly core-to-core, and reuses streamed database pages across queries. FlowIVF bounds irregular list access with fixed-size pages routed through a persistent pipeline and resolves top- K 's data-dependent control by assigning each selection stage to the execution unit best suited to it. We implement and evaluate FlowIVF end-to-end on a Tenstorrent Blackhole device, showing that it achieves higher cost efficiency than GPU baselines. These results suggest that reducing irregular control and maximizing on-chip locality is the practical objective for implementing IVF on spatial dataflow accelerators.

II. BACKGROUND

A. IVF-based Approximate Nearest Neighbor Search

IVF indexing partitions the database vectors into clusters, each stored as an inverted list of its member vectors [14]. When a query arrives, a coarse search compares the query against every cluster centroid and selects a small number of the nearest lists. The candidate vectors in these lists are then scanned, and the K vectors with the smallest distances to the query are returned. As candidate scanning is the dominant cost in traditional IVF, reducing the number of scanned candidates is the main optimization objective [15]. IVF performance is therefore generally determined by the trade-off between the cost of locating relevant lists and the cost of scanning their candidates.

B. ANN Search on Accelerators

Research on accelerating ANN search has been focusing on GPUs. Systems such as FAISS and CAGRA [5], [6] utilize SIMT parallelism, high memory bandwidth, and the cache/shared-memory hierarchy. TPU-KNN takes a different direction from indexed ANN methods, observing that fractional search has low data reuse and is inefficient on accelerators; it removes search-space pruning and instead performs a dense scan over the entire database [16]. Rather than storing the full distance computation in external memory, it retains only a small set of top candidates as distances are computed. This results in a reduction of the coefficient-wise overhead of generic top- K selection while exploiting the accelerator's dense compute throughput. This shows that TPU-KNN's efficiency comes not just from removing pruning, but from redesigning the reduction primitive itself to fit the hardware. Rather than removing pruning as TPU-KNN does, this work keeps IVF's fractional search and instead addresses the problem of mapping it efficiently onto a spatial dataflow accelerator.

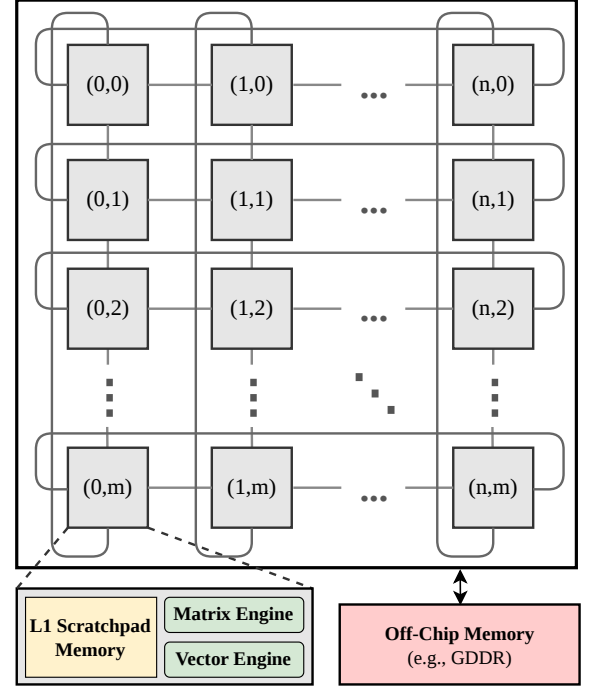


Fig. 1: Spatial dataflow accelerator abstraction.

C. Spatial Dataflow Accelerators

As illustrated in Figure 1, modern spatial dataflow accelerators including Tenstorrent Blackhole [17], AWS NeuronCore [18], Cerebras [19], DOJO [20], Graphcore IPU [21], Groq [22], MTIA [13], SambaNova [23], and Simba [24] consist of a grid of compute cores, each pairing matrix and vector units with a local L1 scratchpad, connected by an on-chip NoC and shared off-chip memory. This local scratchpad can hold persistent working sets resident near the cores that use them, and the NoC allows hierarchical reductions to be built directly between cores without storing intermediate data in off-chip memory [25]. This architecture is already an active target of systems research, including compiler frameworks [26], [27] and virtualization mechanisms [28]. However, fine-grained scalar control, irregular gather/scatter, and data-dependent selection do not map well onto this regular dataflow execution model. IVF includes these kinds of irregular operations, such as query-dependent list selection, variable-length inverted lists, candidate routing, and top- K selection.

III. FLOWIVF DESIGN

A. FlowIVF Overview

Figure 2 illustrates the overall execution flow of FlowIVF. FlowIVF organizes the four execution stages of coarse-search, routing, fine-search, and global reduction into a spatial core pipeline. Each stage executes persistently on a dedicated core group. The coarse-search cores identify the nearest centroids for each query; the routing cores convert the corresponding inverted lists into page-granularity tasks; the fine-search cores scan the target database pages to generate local top- K results; and the reduction cores merge these results into the final top- K results.

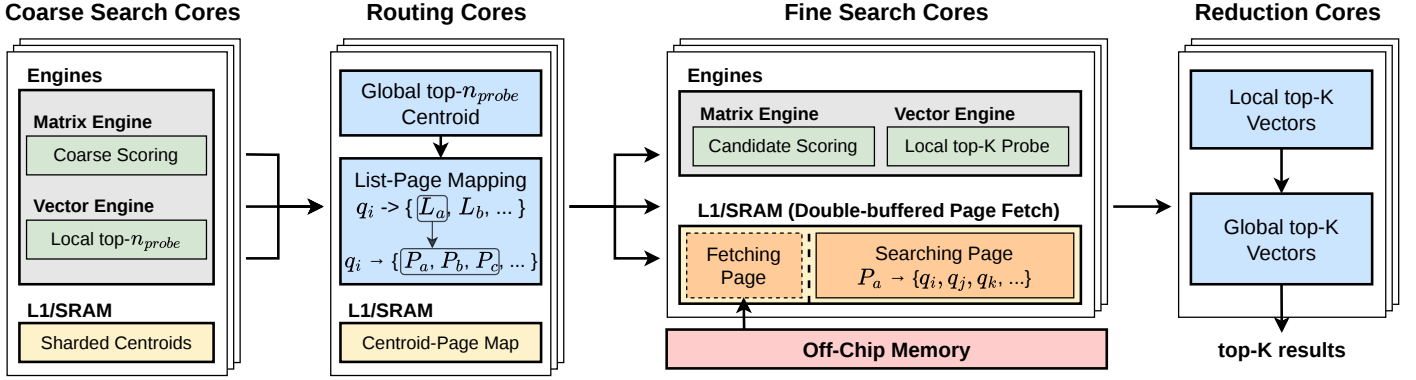


Fig. 2: Overview of FLOWIVF, a persistent spatial pipeline that performs SRAM-resident coarse search, page-based work distribution, candidate scan, and hierarchical top- K reduction entirely on the accelerator.

Intermediate tasks and partial results are transferred directly between core-local SRAMs via the NoC without accessing off-chip memory. This enables the entire search pipeline to execute persistently within the accelerator without host intervention or per-stage off-chip synchronization.

B. SRAM-Resident Centroids and Page-Based Work Distribution

During offline indexing, FlowIVF divides each inverted list into fixed-size pages. Since inverted lists vary in length and an individual list can exceed a single core's SRAM capacity, using an entire list as a monolithic execution unit is impractical. Fixed-size pages normalize the scan workload into a uniform granularity, enabling simple memory address calculation and predictable page-level data transfer. We discuss the choice of page size together with the fine-search execution in Sec. III-C.

Each page is processed exclusively by a designated core determined by a round-robin policy, so the scan workload of a single long list is distributed across multiple cores for parallel execution. Since the interleaving is independent of list identity, per-core page counts differ by at most one within every list, mitigating load concentration from skewed list access.

During runtime, as illustrated in Figure 2, centroids are sharded across the local SRAMs of multiple coarse-search cores and remain resident during the search. Each core computes the distance scores between the incoming query and its resident centroid shard to generate local top- n_{probe} candidates. This eliminates the need to repeatedly fetch centroids from off-chip memory for every query.

Each routing core aggregates partial results from multiple coarse-search cores to determine the global top- n_{probe} inverted lists for its assigned queries. It then looks up the constructed list-to-page mapping to construct query-major page tasks that enumerate all required pages for each query. Each page task is dispatched to the corresponding fine-search core based on the static page-to-core mapping. In this process, query-dependent inverted-list selections are converted into fine-grained page-level tasks while keeping database page placement static. For example, as shown in Figure 2, the list L_a for query q_i is converted into a list of pages such as P_a , P_b , and P_c .

The following section details how fine-search cores restructure query-major page tasks into page-major order, enabling multiple queries to share the same page within local SRAM.

C. Page-Reuse Fine Search

The tasks generated by the routing stage are arranged in query-major order, where all pages required by a given query are queued consecutively before proceeding to subsequent queries. Processing these tasks naively in this order leads to redundant off-chip memory accesses, as identical database pages are repeatedly transferred from off-chip memory. FlowIVF does not process incoming (query, page) tasks immediately. Instead, it buffers them within a multi-query window at each fine-search core. Each core groups the tasks within the window by page ID, effectively reconstructing the query-major stream into a page-major execution order.

In page-major execution order, queries referencing the same page are grouped together. For example, if q_i , q_j , and q_k all require page P_a , the fine-search core brings P_a into its local SRAM from off-chip memory only once per window and reuses the page across all three queries. A higher degree of page sharing across queries thus allows the accelerator to more effectively amortize database memory traffic.

This reuse requires a window to be complete before scanning begins, since a page is fetched once per window and every query referencing it must be known in advance. As the pipeline is persistent, routing for the next window overlaps with the current window's scan, so this cost is exposed only for the first window of a batch.

The number of queries covered by one window is the window size W , which determines the scope of page reuse. A larger W exposes more opportunities for multiple queries to share the same page and amortizes each page fetch over more query scans. To overlap page movement with candidate scoring, each fine-search core uses double buffering. While the matrix engine processes the current page, the next page is fetched into a second SRAM buffer. Consequently, the window size W and page size s (the number of vectors per page) draw on the same per-core SRAM budget. For vector dimension D , the dominant SRAM requirement is proportional to

$$2sD + WD, \quad (1)$$

where the first term corresponds to the two page buffers and the second to the query vectors retained for the current window.

This coupling creates a trade-off between reuse scope and page granularity. Increasing W expands the set of queries over which page sharing can be exploited, but also consumes more SRAM for per-query state. Increasing s reduces the number of page-level tasks and fixed per-page control operations, but consumes more SRAM for double-buffered page storage, leaving less capacity for query aggregation and thereby limiting the scope of cross-query page reuse. Since both terms in Eq. (1) scale with D , this trade-off becomes tighter for higher-dimensional datasets.

Once a page is resident in SRAM, the matrix engine computes candidate scores between the page vectors and the queries grouped for that page. The resulting score and index pairs are sorted by the vector engine, and a bounded top- K per query is maintained from these sorted runs. This keeps dense distance computation on the matrix datapath and completes selection locally, so only the sorted local top- K is forwarded over the NoC to the reduction cores.

D. Hierarchical Top- K Reduction

Each fine-search core produces its local top- K sorted by score. A reduction core merges multiple local top- K lists, discarding the remaining candidates once they can no longer enter the current top- K . If a fine-search core processed no pages for a given query, its contribution is skipped after inspecting only the header.

The reduction tree consists of two stages. First, reduction cores merge the local top- K lists from one group of fine-search cores. Next, the final core merges these partial results into the global top- K . The first stage is further split by query parity, so that each reduction core is responsible for one fine-search core group and one query parity. This limits both the number of queries each reduction core must process and the number of inputs the final core must merge per query.

Retaining only the local top- K at each fine-search core does not affect the correctness of the global top- K . Since a single core contributes at most K candidates to the final result, any candidate outside that core's local top- K cannot appear in the global top- K . Consequently, performing an exact merge at every reduction stage yields the exact global top- K over the generated candidate scores, without introducing additional approximation.

The final core translates each selected candidate's in-page position into its original vector ID using an offline-generated ID mapping, and writes the resulting neighbor IDs directly to off-chip memory, eliminating the need for host-side ID reconstruction or additional lookups after search completes.

Overall, this hierarchical reduction avoids collecting the full candidate score set in one place. Each fine-search core first reduces its results to a local top- K , and this bounded partial result is transferred over the NoC. This reduces both the storage required for a large intermediate score matrix and the volume of inter-core communication, allowing the entire pipeline to complete within the accelerator.

IV. EVALUATION

A. Experimental Setup

Experimental Platform. We evaluate FlowIVF on a single Tenstorrent Blackhole p150a accelerator, which provides 120 Tensix cores. For the GPU baseline, we use a single NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition and evaluate cuVS 26.02 IVF-Flat and IVF-PQ using `cuvs_bench`. Table I summarizes the key characteristics of the two platforms.

TABLE I: Evaluation platform comparison.

	Blackhole p150a	RTX PRO 6000
Price	\$1,399 [29]	\$16,000 [30]
Compute units	120 Tensix cores	188 SMs
On-chip SRAM	1.5 MB/core	128 KB/SM
Device memory	32 GB GDDR6	96 GB GDDR7
Memory bandwidth	512 GB/s	1,792 GB/s

Workloads and Measurement. We evaluate SIFT-1M ($D=128$, 10,000 queries) and GIST-1M ($D=960$, 1,000 queries), each containing one million base vectors under Euclidean distance. Recall@10 is the average percentage of the true top-10 nearest neighbors contained in the returned top-10 results [31]. We report query throughput in queries per second (QPS) and cost efficiency as QPS per dollar using the hardware prices in Table I.

Throughput and cost efficiency are measured using the complete native query set as a single batch across different recall levels, while batch scaling is evaluated at Recall@10 of approximately 0.90 with batch sizes from 8 to 512. Online latency is reported as the mean per-query execution time at a batch size of one. For both platforms, the index and queries reside in device memory, and host-to-device transfers are excluded from all measurements.

Parameter Setting. We set $nlist = 1024$ for both datasets, following the common $O(\sqrt{N})$ guideline for IVF indices [32]. At query time, we vary n_{probe} to obtain different trade-offs between query speed and search quality. For IVF-PQ, we set $M_{PQ} = D/2$ and $nbits_{PQ} = 4$, with re-ranking of the top $K \times k_factor$ candidates ($k_factor = 10$). For FlowIVF, the window size is the smaller of the batch size and the largest window that fits the SRAM budget of Eq. (1).

B. Throughput and Cost Efficiency

Fig. 3 compares purchase-price-normalized throughput (QPS/\$) across the evaluated Recall@10 range. We use IVF-Flat as the direct comparison for uncompressed IVF search, and IVF-PQ as a stronger practical baseline that reduces memory traffic through vector compression. FlowIVF consistently achieves higher cost efficiency than both cuVS baselines on SIFT-1M and GIST-1M. At Recall@10 targets between 0.95 and 0.99, FlowIVF achieves approximately 2 to 4 $\times$ higher QPS/\$ on SIFT-1M and approximately 5 $\times$ on GIST-1M than IVF-PQ. The margin over IVF-Flat is substantially larger, reaching 10 to 12 $\times$ on SIFT-1M and more than 50 $\times$ on GIST-1M. These advantages persist across the evaluated recall range.

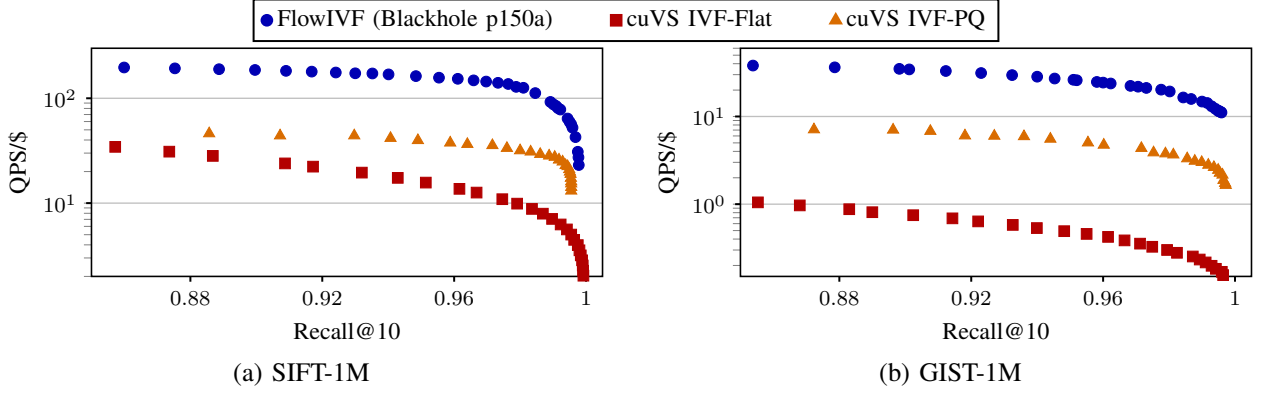


Fig. 3: Cost efficiency (QPS per dollar of hardware cost), under various recall rates.

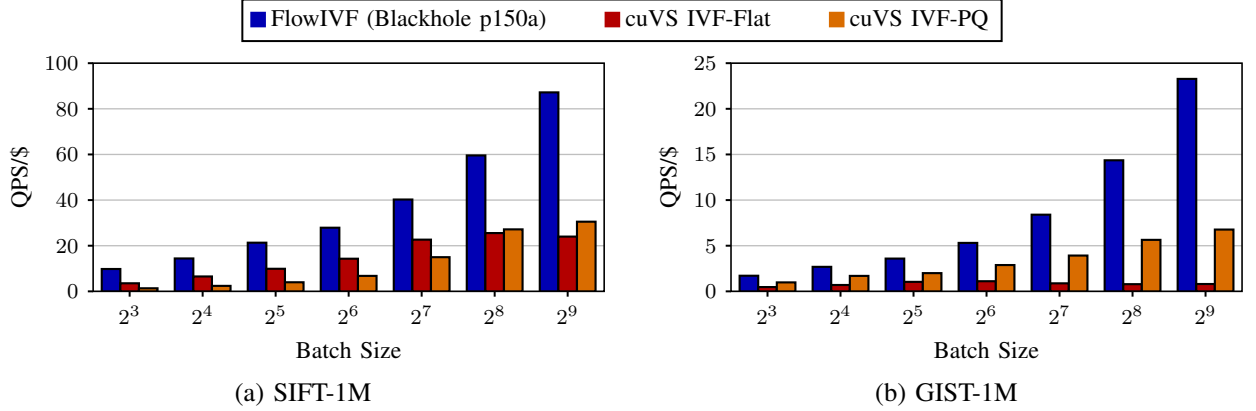


Fig. 4: Cost efficiency (QPS/\$) versus query batch size at approximately Recall@10 = 0.90 on (a) SIFT-1M and (b) GIST-1M.

TABLE II: Throughput at target Recall@10.

Dataset	Recall	FlowIVF	IVF-Flat	IVF-PQ
SIFT-1M	0.95	221.6k	237.1k	601.6k
	0.97	193.8k	185.4k	568.9k
	0.99	84.4k	99.8k	439.8k
GIST-1M	0.95	36.7k	7.2k	80.3k
	0.97	30.6k	5.5k	69.5k
	0.99	20.7k	3.6k	48.0k

Cost normalization alone does not capture absolute search performance. Table II therefore reports raw throughput at matched Recall@10 targets. On SIFT-1M, FlowIVF remains close to IVF-Flat, achieving between $0.85\times$ and $1.05\times$ its throughput across the reported targets, while IVF-PQ achieves $2.7\times$ to $5.2\times$ higher throughput. On GIST-1M, FlowIVF outperforms IVF-Flat by $5.1\times$ to $5.8\times$ in absolute throughput, while IVF-PQ achieves 2.2 to $2.3\times$ higher throughput than FlowIVF. GIST vectors are $7.5\times$ wider than SIFT vectors ($D=960$ versus $D=128$), increasing the amount of database data moved for each candidate scan. By reorganizing query-major work into page-major execution, FlowIVF groups requests to the same page across queries and reuses each fetched page from local SRAM for multiple query evaluations. This reduces redundant off-chip transfers of the wider GIST vectors, helping FlowIVF sustain $5.1\times$ to $5.8\times$ higher throughput

than IVF-Flat on GIST-1M.

FlowIVF therefore achieves high cost efficiency through page reuse while searching uncompressed vectors, whereas IVF-PQ attains higher raw throughput by reducing scan traffic through vector compression. Supporting PQ in FlowIVF would require a different fine-search datapath for codebook lookups rather than the dense vector-distance computation that the current design maps to the matrix engine. Even against this compressed baseline, FlowIVF delivers substantially higher QPS/\$, and on GIST-1M, it also exceeds the absolute throughput of the GPU IVF-Flat baseline despite the GPU's $3.5\times$ higher memory bandwidth. These results demonstrate the effectiveness of page-centric execution on distributed SRAM for cost-efficient IVF search.

C. Batch Scaling

Fig. 4 shows how cost efficiency changes with query batch size at a fixed Recall@10 of approximately 0.90. FlowIVF benefits consistently from larger batches on both datasets and continues to scale through the largest batch shown. On SIFT-1M, its cost efficiency increases from 9.8 QPS/\$ at batch 8 to 87.2 QPS/\$ at batch 512, remaining above both cuVS baselines throughout the range. On GIST-1M, the gain grows from 1.7 to 23.3 QPS/\$, while IVF-Flat saturates at much smaller batch sizes. At batch 512, FlowIVF achieves $2.9\times$ and $3.4\times$ higher

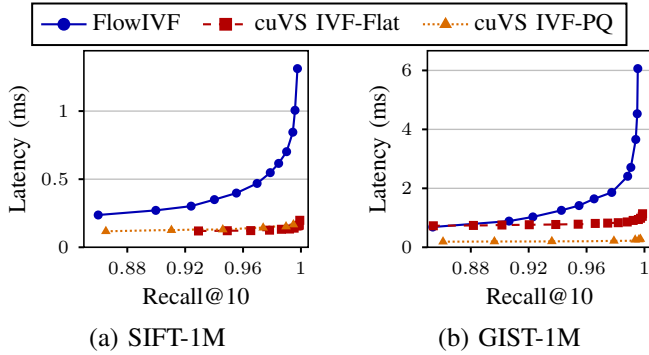


Fig. 5: Per-query latency under various recall rates.

QPS/\$ than the best GPU baseline on SIFT-1M and GIST-1M, respectively.

The scaling behavior follows from the same mechanism. A larger batch exposes more requests to the same pages, so each fetch is shared by more queries, and the effect is strongest on GIST-1M where pages carry more bytes. FlowIVF thus converts batch size into data reuse rather than only amortizing fixed overhead, whereas the GPU baselines retain their per-query list scans, which is why their curves flatten while FlowIVF’s continues to rise.

D. Online Latency

Fig. 5 compares mean per-query latency at a batch size of one. This setting removes cross-query page reuse, since every fetched page serves only a single query. Accordingly, both cuVS baselines achieve lower single-query latency across the evaluated recall range, with IVF-PQ lowest as it scans compressed codes. Latency increases with recall for all methods as larger n_{probe} values increase the search work per query.

FlowIVF nevertheless stays below 1 ms on SIFT-1M over most recall levels and within single-digit milliseconds on GIST-1M at the highest recall targets. Search state is kept in local SRAM, while intermediate tasks and compact partial results move directly between stages over the NoC. This avoids repeated off-chip transfers of intermediate state and keeps coordination within the accelerator throughout the search pipeline. Overall, FlowIVF maintains low single-query latency through its use of local SRAM and NoC communication, while batched execution provides the additional throughput benefit of cross-query page reuse.

V. RELATED WORK

A. ANN search algorithms and systems

A large body of work has developed ANN search algorithms and systems for large-scale retrieval. Graph-based methods such as HNSW [1] are extended to billion-scale, disk-resident search by DiskANN, which keeps a small in-memory graph over SSD-resident data [33]. Clustering-based methods such as IVF are similarly extended by SPANN, which stores inverted lists on SSD while keeping only cluster centroids resident in memory [34]. These systems are primarily CPU- or memory-centric in design. Other systems are built upon

custom, purpose-specific hardware for ANN search. DF-GAS builds a distributed FPGA-as-a-service architecture for billion-scale graph-based search [35], NDSearch accelerates graph-traversal-based search through near-data processing [9], and REIS targets energy-efficient retrieval via in-storage processing [10].

B. ANN on ML accelerators

Separately, other works extend ANN search using ML accelerators. On GPUs, FAISS exploits SIMT parallelism and shared-memory hierarchy for exact and quantized distance computation [5], while CAGRA instead targets GPU-native graph construction and traversal for approximate search [6]. Other GPU-based systems address cost and memory limitations. CXL-ANNS disaggregates memory over an external interconnect [8], RUMMY reorders query pipelining to process datasets beyond GPU memory capacity [36], and TridentANN combines low-end GPUs with SSD arrays for higher throughput per dollar [7]. On non-GPU accelerators, TPU-KNN targets TPUs by removing IVF-style pruning for a dense, accelerator-friendly scan [16]. Production vector databases such as GaussDB-Vector instead treat accelerators as an offload target, but use them only for batched distance computation rather than for the full search pipeline [37]. FlowIVF keeps IVF’s fractional search and maps it onto a spatial dataflow accelerator built for general ML serving.

VI. CONCLUSION AND FUTURE WORK

In this paper, we demonstrate the feasibility of leveraging spatial dataflow accelerators for ANN search and identify the challenges of executing IVF search on those platforms. We present FlowIVF, which restructures the IVF search pipeline across distributed on-chip SRAMs. As this design is extensible to other spatial dataflow accelerators sharing comparable architectural features, we envision that this work provides a foundation for future innovations in cost-efficient ANN search.

Several directions remain open: scaling beyond a single chip’s memory capacity would require extending page-to-core ownership to shard-to-device ownership, with cross-chip fabric communication and shard balancing left for future optimization; and graph-based ANN search, whose neighbor traversal and communication are determined dynamically rather than statically after coarse selection, remains an open problem for mapping onto distributed SRAM and NoC.

REFERENCES

- [1] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.
- [2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.
- [3] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, “Dense passage retrieval for open-domain question answering,” in *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, 2020, pp. 6769–6781.

- [4] J.-T. Huang, A. Sharma, S. Sun, L. Xia, D. Zhang, P. Pronin, J. Padmanabhan, G. Ottaviano, and L. Yang, "Embedding-based retrieval in facebook search," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2553–2561.
- [5] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE transactions on big data*, vol. 7, no. 3, pp. 535–547, 2019.
- [6] H. Ootomo, A. Naruse, C. Nolet, R. Wang, T. Feher, and Y. Wang, "Cagra: Highly parallel graph construction and approximate nearest neighbor search for GPUs," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4236–4247.
- [7] Y. Huang, B. Ma, E. Xu, and C. Weng, "Don't Surrender to Low QPS/\$: Fast and Cost-Efficient ANNS with TridentANN," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026, pp. 258–273.
- [8] J. Jang, H. Choi, H. Bae, S. Lee, M. Kwon, and M. Jung, "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)*, 2023, pp. 585–600.
- [9] Y. Wang, S. Li, Q. Zheng, L. Song, Z. Li, A. Chang, Y. Chen *et al.*, "Nd-search: 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, 2024, pp. 368–381.
- [10] K. Chen, R. Nadig, M. Frouzakis, N. M. Ghiasi, Y. Liang, H. Mao, J. Park, M. Sadrosadati, and O. Mutlu, "Reis: A high-performance and energy-efficient retrieval system with in-storage processing," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1171–1192.
- [11] J. Vasiljevic, L. Bajic, D. Capalija, S. Sokorac, D. Ignjatovic, L. Bajic, M. Trajkovic, I. Hamer, I. Matosevic, A. Cejkov *et al.*, "Compute substrate for software 2.0," *IEEE micro*, vol. 41, no. 2, pp. 50–55, 2021.
- [12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.
- [13] A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, G. Grewal, H. Aepala, B. Jakka, B. Dreyer *et al.*, "Mtia: First generation silicon targeting meta's recommendation systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.
- [14] Sivic and Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings ninth IEEE international conference on computer vision*. IEEE, 2003, pp. 1470–1477.
- [15] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 861–864.
- [16] F. Chern, B. Hechtman, A. Davis, R. Guo, D. Majnemer, and S. Kumar, "Tpu-knn: K nearest neighbor search at peak flop/s," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 489–15 501, 2022.
- [17] J. Vasiljevic and D. Capalija, "Blackhole & tt-metalium: The standalone ai computer and its programming model," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society Los Alamitos, CA, USA, 2024, pp. 1–30.
- [18] AWS. (2025) Neuroncore-v3 architecture. Referenced September 2025. [Online]. Available: <https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-hardware/neuron-core-v3.html#neuroncores-v3-arch>
- [19] S. Lie, "Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning," *IEEE Micro*, vol. 43, no. 3, pp. 18–30, 2023.
- [20] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla's exa-scale computer," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–28.
- [21] S. Knowles, "Graphcore," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–25.
- [22] D. Abts, G. Kimmell, A. Ling, J. Kim, M. Boyd, A. Bitar, S. Parmar, I. Ahmed, R. DiCecco, D. Han *et al.*, "A software-defined tensor streaming multiprocessor for large-scale machine learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 567–580.
- [23] R. Prabhakar and S. Jairath, "Sambanova sn10 rdu: Accelerating software 2.0 with dataflow," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–37.
- [24] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 14–27.
- [25] Y. Xie, C. Mao, C. Wu, C. Lu, C. Suo, C. Qian, C. Yang, D. Zhu, H. Xiong, H. Lu *et al.*, "M100: An orchestrated dataflow architecture powering general ai computing," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026, pp. 1044–1055.
- [26] W. Li, Z. Bai, H. Wang, P. Dangi, Z. Zhang, C. Tan, H. Lan, W.-F. Wong, and T. Mitra, "{TileLoom}: Automatic dataflow planning for {Tile-Based} languages on spatial dataflow accelerators," in *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, 2026, pp. 1891–1908.
- [27] Y. Liu, Y. Xue, N. Crawford, J. Xue, and J. Huang, "Elk: Exploring the efficiency of inter-core connected ai chips with deep learning compiler techniques," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 1284–1299.
- [28] D. Feng, E. Feng, D. Du, P. Xu, Y. Xia, H. Chen, and R. Zhao, "Topology-aware virtualization over inter-core connected neural processing units," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1210–1224.
- [29] Tenstorrent Inc., "Blackhole AI Accelerator Cards," <https://tenstorrent.com/hardware/cards>, 2026, accessed: Aug. 2026.
- [30] NVIDIA Corporation, "NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition," <https://marketplace.nvidia.com/en-us/enterprise/laptops-workstations/nvidia-rtx-pro-6000-blackwell-max-q-workstation-edition/>, 2026, accessed: Aug. 2026.
- [31] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2010.
- [32] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *arXiv preprint arXiv:2401.08281*, 2024.
- [33] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *Advances in neural information processing Systems*, vol. 32, 2019.
- [34] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang, and J. Wang, "Spann: Highly-efficient billion-scale approximate nearest neighborhood search," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5199–5212, 2021.
- [35] S. Zeng, Z. Zhu, J. Liu, H. Zhang, G. Dai, Z. Zhou, S. Li, X. Ning, Y. Xie, H. Yang *et al.*, "Df-gas: A distributed fpga-as-a-service architecture towards billion-scale graph-based approximate nearest neighbor search," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 283–296.
- [36] Z. Zhang, F. Liu, G. Huang, X. Liu, and X. Jin, "Fast vector query processing for large datasets beyond {GPU} memory with reordered pipelining," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 23–40.
- [37] J. Sun, G. Li, J. Pan, J. Wang, Y. Xie, R. Liu, and W. Nie, "Gaussdb-vector: A large-scale persistent real-time vector database for llm applications," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 4951–4963, 2025.