DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-based Approximate Nearest Neighbor Search

Shulin Zeng*
Department of Electronic Engineering, Tsinghua University
Beijing, China
zengsl18@mails.tsinghua.edu.cn

Zhenhua Zhu*
Department of Electronic Engineering, Tsinghua University
Beijing, China
zhuzhenh18@mails.tsinghua.edu.cn

Jun Liu
Department of Electronic Engineering, Tsinghua University
Beijing, China
liu-j20@mails.tsinghua.edu.cn

Haoyu Zhang
Department of Electronic Engineering, Tsinghua University
Beijing, China
hy-zhang22@mails.tsinghua.edu.cn

Guohao Dai†
Qingyuan Research Institute, Shanghai Jiao Tong University
Shanghai, China
daiguohao@sjtu.edu.cn

Zixuan Zhou
Department of Electronic Engineering, Tsinghua University
Beijing, China
zhouzx21@mails.tsinghua.edu.cn

Shuangchen Li
DAMO Academy, Alibaba Group
Sunnyvale, USA
shuangchen.li@alibaba-inc.com

Xuefei Ning
Department of Electronic Engineering, Tsinghua University
Beijing, China
foxdoraame@gmail.com

Yuan Xie
ACCESS, HKUST
Hongkong, China
DAMO Academy, Alibaba Group
Sunnyvale, USA
y.xie@alibaba-inc.com

Huazhong Yang
Department of Electronic Engineering, Tsinghua University
Beijing, China
yanghz@mail.tsinghua.edu.cn

Yu Wang†
Department of Electronic Engineering, Tsinghua University
Beijing, China
yu-wang@tsinghua.edu.cn

ABSTRACT

Embedding retrieval is a crucial task for recommendation systems. Graph-based approximate nearest neighbor search (GANNS) is the most commonly used method for retrieval, and achieves the best performance on billion-scale datasets. Unfortunately, the existing CPU- and GPU-based GANNS systems are difficult to optimize the throughput under the latency constraints on billion-scale datasets, due to the underutilized local memory bandwidth (5-45%) and the expensive remote data access overhead (~85% of the total latency). In this paper, we first introduce a practically ideal GANNS architecture for billion-scale datasets, which facilitates a detailed analysis of the challenges and characteristics of distributed GANNS systems. Then, at the architecture level, we propose DF-GAS, a Distributed FPGA-as-a-Service (FPaaS) architecture for accelerating billion-scale Graph-based Approximate nearest neighbor Search. DF-GAS uses a feature-packing memory access engine and a data prefetching and delayed processing scheme to increase local memory bandwidth by 36-42% and reduce remote data access overhead by 76.2%, respectively. At the system level, we exploit the "*full*-graph + *sub*-graph" hybrid parallel search scheme on distributed FPaaS system. It achieves million-level query-per-second with sub-millisecond latency on billion-scale GANNS for the first time. Extensive evaluations on million-scale and billion-scale datasets show that DF-GAS achieves an average of 55.4×, 32.2×, 5.4×, and 4.4× better latency-bounded throughput than CPUs, GPUs, and two state-of-the-art ANNS architectures, *i.e.,* ANNA [23] and Vstore [27], respectively.

*Both authors contributed equally to this research.
†Corresponding authors.

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3614292

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Embedding Retrieval, FPGA, Distributed Architecture, Graph, Approximate Nearest Neighbor Search

ACM Reference Format:
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 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO*

'23), October 28–November 01, 2023, Toronto, ON, Canada. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3613424.3614292

1 INTRODUCTION

Recommendation systems have been widely deployed in the industry, including Google DCN [41], YouTube [9], Facebook DLRM [35], and Microsoft Bing [33]. Industrial recommendation systems are usually composed of two stages: First, ***retrieval*** roughly selects hundreds of candidates from billion-scale databases based on search algorithms, *e.g.,* approximate nearest neighbor search (ANNS). Second, ***ranking*** precisely chooses dozens from the selected candidates using learning algorithms, *e.g.,* fully-connected networks (FCN). ANNS is one of the most commonly used methods for retrieval, since it can effectively boost the search throughput with a slight accuracy loss. Industry giants have deployed different commercial ANNS solutions, such as Google ScaNN [15], Facebook Faiss [21], Microsoft SPTAG [34], etc. These solutions contain various ANNS algorithms, like tree-based methods [16, 46], hash-based methods [10, 38], product-quantization (PQ) based methods [15, 21], and graph-based methods [17, 32].

Among different ANNS algorithms, the graph-based ANNS algorithm (GANNS) achieves the best performance with high accuracy and short search latency [26]. GANNS first maps the features in datasets to the points of an embedding space and then constructs a graph using these points offline. During the online search, GANNS traverses the graph to find the nearest points to the input query that is also mapped to the embedding space. The search stage mainly contains three operations: accessing neighbor index lists and features, calculating feature distance between query and datasets, and sorting distance. The first one is a memory-intensive graph traversal operation, while the others are compute-intensive operations.

One important metric for GANNS is the throughput under the service-level agreement (SLA) constraint, namely latency-bounded throughput. On the one hand, improving the throughput, *i.e.,* query-per-second (QPS), can reduce the operating cost by serving more requests with fewer servers. On the other hand, tighter latency constraints can leave more time budget for the ranking stage to improve final results [8]. To achieve high latency-bounded throughput, we derive that a practically ideal GANNS hardware architecture should satisfy the following features: ***First***, it should contain a distributed memory system since billion-scale datasets require hundreds to thousands GBs of memory footprints; ***second***, the architecture should enable memory-efficient graph traversing on both local and remote nodes in distributed systems; ***third***, sufficient data parallelism is required for efficient processing of the batched queries.

Unfortunately, existing general-purpose hardware (CPU and GPU) cannot meet the above requirements with the following challenges unresolved:

(1) Low memory bandwidth utilization for single-node scaling up. Because of the irregular graph data access, the memory bandwidth of CPU and GPU are both severely underutilized, *i.e.,* 5-45%, which introduces extremely long latency. Besides, the lack of sufficient parallel strategy for the searching flow (especially for the sorting operations) causes the waste of parallel computing capability of multi-core hardware, further limiting the achievable latency-bounded throughput.

(2) Long remote data access latency with heavy communication overhead for multi-node scaling out. Existing distributed GANNS solutions [13, 14] partition the original *full*-graph database into multiple independent *sub*-graphs, eliminating all the inter-*sub*-graph communication overhead (~85% of the total latency). However, because the connectivity of the graph deteriorates, the graph partition needs more search steps in *sub*-graphs to find the nearest neighbors. As a result, existing graph partition methods introduce additional computation overhead (4-6×), reducing the throughput by 3-5×. Thus, there is still much room for optimizing GANNS on billion-scale datasets.

In this paper, we aim to design a customized architecture for billion-scale GANNS, with the goals of scaling up, scaling out, and high performance. To this end, we propose **DF-GAS**, a Distributed FPGA-as-a-Service (FPaaS) architecture for accelerating billion-scale Graph-based Approximate nearest neighbor Search. DF-GAS is a proof-of-concept implementation of the practically ideal GANNS hardware architecture with a customized feature-packing Memory Access Engine (MAE). The contributions of this paper include:

- We introduce a practically ideal architecture for billion-scale GANNS. The practically ideal architecture facilitates a detailed analysis of the challenges and characteristics of existing distributed GANNS systems.
- For scaling up, we propose a two-level outstanding request generation technique to boost the memory-level parallelism. We also design the intra-Processing Element (PE) pipeline and the inter-PE out-of-order (OoO) execution flow to improve the memory bandwidth utilization by 36-42%.
- For scaling out, we propose prefetching and delayed processing schemes for remote neighbor lists and features to hide the long remote access latency. Therefore, the communication overhead is reduced by 76.2%.
- We build up the first distributed FPaaS system with sub-millisecond ultra-low latency for billion-scale GANNS. We propose a *hybrid* parallel search scheme that exploits both throughput-optimized *full*-graph parallel and latency-optimized *sub*-graph parallel. Compared with CPUs and GPUs, DF-GAS achieves at least 92-218× and 135-294× lower latency, respectively.

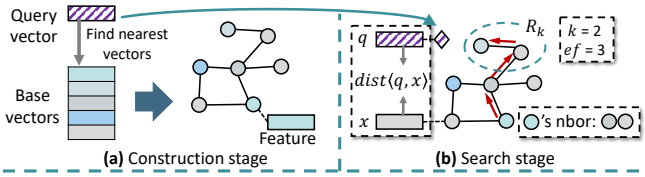
Comprehensive evaluations on million-scale and billion-scale datasets show that DF-GAS achieves 23.8-233.4×, 1.9-50.8×, 2.1-8.6×, and 1.3-7.5× latency-bounded throughput than CPUs, GPUs, ANNS accelerators, *i.e.,* ANNA [23] and Vstore [27], respectively.

2 BACKGROUND

2.1 Nearest Neighbor Search

2.1.1 Definition. Nearest Neighbor Search (NNS) aims to find the most similar result for a *query vector* $q \in \mathbb{R}^d$ in a given finite set of base feature vectors $X \in \mathbb{R}^d$. It can be expressed as:

$$R = \arg\min_{x \in X} dist\,\langle q, x\rangle \tag{1}$$



(c) Search Algorithm Pseudocode

```
1 # Graph ANN Search
2 for i in iterations:
3   Features = GetNborFeature(DataBase, NextP) # graph traverse
4   Features = Deduplication(VisitedList, Features) # filter
5   Scores = DistCal(Features, Input) # distance calculation
6   NextP = Sorting(Queue, Scores) # update topK queue
7 return Result = TopK(Queue) # return final topK candidates
```

Figure 1: Illustration of GANNS algorithms: (a) the construction stage, (b) the search stage, and (c) the pseudocode.

Here $dist\ \langle q, x\rangle$ is the distance metric (*e.g.*, L2 or inner product) to indicate the similarity between q and x. Similarly, we denote the nearest k results as R_k.

2.1.2 Graph-based ANNS. GANNS algorithms abstract the base feature vectors as points in a high-dimensional space. Then a construction stage is carried out to construct a graph structure using these points, whose edges represent relations among points. We call the i-th base feature vector and its neighbor indexes as the feature and neighbor list of the i-th point in the graph, respectively.

Construction Stage. The construction stage is to connect the points in the base dataset according to construction strategies, as shown in Figure 1(a). The main difference among different GANNS algorithms is the construction strategy. For example, hierarchical navigable small world (HNSW) [32] adopts an incremental and relative nearest neighbor [39] based construction strategy. NSG [13] adopts a pre-built kNN graph and approximately monotonic search networks [11].

Search Stage. Most GANNS algorithms adopt the best first search strategy [40], which requires multiple iterations to approach the nearest results, as shown in Figure 1(b). For each iteration, we first select the point closest to the query from the traversed points. Then, we visit its untraversed neighbors from the neighbor list, which are filtered by a visited list to avoid repeated visits. Next, we calculate the distances among the query feature and these neighbors' features. Finally, we insert the distances and the corresponding point indexes into one priority queue. This process is repeated for ef iterations. Here ef is a hyperparameter to control the accuracy-speed tradeoff.

2.1.3 Evaluation metrics. The search accuracy of GANNS is evaluated by the recall rate. For the query vector q, its nearest k (top K) results from ANNS is R'_k, Then the recall rate $Recall@k$ is defined as:

$$Recall@k = \frac{\left|R'_k \cap R_k\right|}{|R_k|} \tag{2}$$

The search performance is usually evaluated using the query-per-second (QPS) and latency.

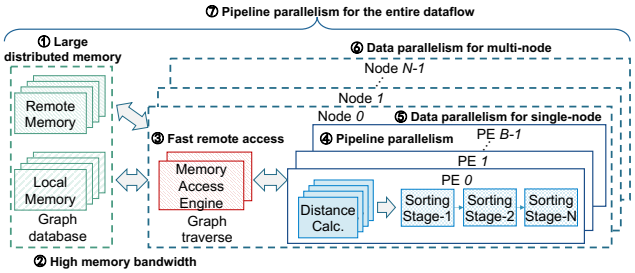


Figure 2: Practically ideal GANNS hardware architecture with appropriate data and pipeline parallelism for billion-scale datasets.

2.2 Cloud FPGAs

Distributed FPaaS [25] has the potential to fulfill all the features required by the ideal GANNS architecture. First, FPGAs with both double data rate memory (DDR) and high bandwidth memory (HBM) [44] can provide both high memory bandwidth (*e.g.*, 490GB/s) and large memory capacity (*e.g.*, 40GB). Besides, the sufficient and programmable on-chip resources of FPGAs enable easy implementation with appropriate data and pipeline parallelism [48]. Moreover, the FPaaS platform is naturally scalable in the cloud.

3 MOTIVATIONS AND CHALLENGES

3.1 Practically Ideal GANNS Hardware Architecture

To maximize the latency-bounded throughput while meeting realistic hardware constraints, we introduce a practically ideal GANNS hardware architecture for billion-scale datasets, as shown in Figure 2.

On the memory side, the graph database of GANNS usually contains billions of points, each with a high-dimensional feature (*e.g.*, 96-128B). Thus, the entire database requires a large distributed memory with hundreds to thousands of GBs capacity (Figure 2 ①).

On the computation side, the online search stage of GANNS mainly contains three operations:

- ***Neighbor lists and feature access*** (line 3, Figure 1(c)) is a memory-intensive operation. Traversing the large and distributed database introduces massive local and remote memory requests for a single query, demanding an efficient memory access engine to provide both high memory bandwidth (Figure 2 ②) and fast remote access (Figure 2 ③).
- ***Distance calculation*** (line 5, Figure 1(c)) calculates the distance between the selected points and the input query. Since there is no dependency among distance calculations for different points, an ideal architecture should implement distance calculation with enough data parallelism (Figure 2 ⑤) to match the throughput of graph traversal.
- ***Distance sorting*** (line 6, Figure 1(c)) usually runs in a deeply pipelined manner (Figure 2 ④) with multiple sub-stages to achieve a balance between performance and resources (*e.g.*, insertion sort). Besides, the distance calculation and sorting can also be overlapped with appropriate pipeline parallelism (Figure 2 ④) to serve one query inside a single PE.

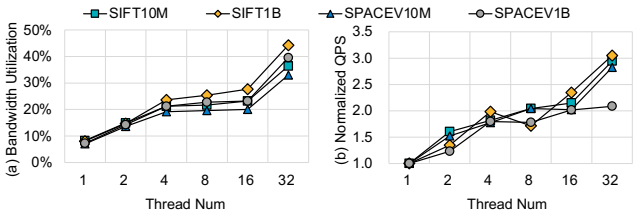


Figure 3: Characterizations on CPUs. (a) Bandwidth utilization and (b) scability with different number of threads.

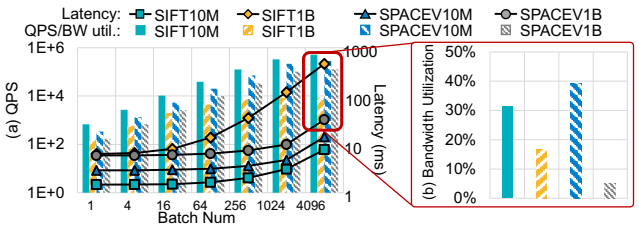


Figure 4: Characterizations on GPUs. (a) QPS and latency under different batch sizes, and (b) bandwidth utilization with a 10ms SLA constraint.

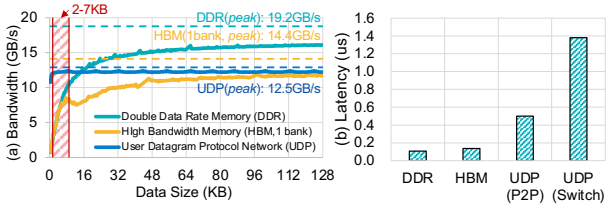


Figure 5: (a) Bandwidth with different data sizes and (b) memory access latency of DDR, HBM (1 bank), and UDP on FPGAs.

The single-node ideal architecture can support batch queries with data parallelism (Figure 2 ⑤) using multiple PEs. Also, it can be extended to the distributed system for larger batch sizes using multi-node data parallelism (Figure 2 ⑥). Moreover, to overlap the computation with local/remote communication, there should be system-level pipeline parallelism (Figure 2 ⑦) for the entire dataflow.

In order to implement distributed GANNS systems, we need to partition the graph-based search and deploy it to a multi-node/multi-PE system. There exist two partition strategies for running GANNS in parallel, *i.e., full-* and *sub*-graph parallel search (GPS). For the *full*-GPS strategy, the graph construction stage is the same as the original GANNS algorithms. In the search stage, multiple PEs or nodes share one logical *full* graph with batch queries, where the partitioned graphs still retain connectivity with each other. For the *sub*-GPS strategy, points of the base dataset are divided into multiple separate parts, which are used to construct *sub*-graphs independently. During the search stage, different PEs or nodes perform searching for the same query on different sub-graphs simultaneously. Then the partial results of these sub-graphs are merged to get the final results.

3.2 Limitation of Existing Work

In this part, we present motivation examples by running state-of-the-art (SOTA) CPU- [1, 13] and GPU-based design [14] and analyze their major limitations using the proposed practically ideal GANNS architecture. The hardware platforms we used are Intel Xeon Gold 5218 CPU and Nvidia RTX 3090 GPU. The evaluated datasets include SIFT [19] and SPACEV [2], with ten million (10M) and one billion (1B) points. We evaluate million-scale and billion-scale datasets on a single machine and an eight-node cluster, respectively.

3.2.1 Distributed multi-node design. In the *full*-GPS strategy, because of the large number of edges connecting different sub-graphs, there exist frequent remote data accesses during the search. Our evaluation shows that the remote communication overhead takes up 84.1-86.0% of the total latency in existing distributed GANNS systems. Therefore, the existing distributed GANNS solutions [13, 14] utilize the *sub*-GPS strategy. Since there is no connectivity among sub-graphs in *sub*-GPS, it eliminates almost all the remote communication overhead (except for the final merge operation). Considering a realistic eight-node system with 200Gbps networks, the *sub*-GPS outperforms the throughput of *full*-GPS by ~1.5× on the SIFT1B dataset. However, because the *sub*-GPS strategy loses plenty of global information with poor connectivity, more search steps are required to find the nearest neighbors. Such additional computation overhead can reach 4-6× and reduce the throughput by 3-5×, meaning that there is still a lot of room for optimization.

3.2.2 CPU-based design. Typical CPU-based systems (*e.g.,* Intel Xeon Gold CPU) have relatively large memory capacity (*e.g.,* 32-128GB) but low memory bandwidth (*e.g.,* 38.4-153.6GB/s). We find that memory access is the bottleneck of GANNS, which takes up about 80% runtime of the search stage. Moreover, the memory bandwidth is underutilized (violate Figure 2 ②). Figure 3(a) shows that the bandwidth utilization is only 8% under the single-thread execution. Even when running with 32 threads, the bandwidth utilization only increases by about 5× to 39%. On the one hand, the conflicts of shared cache resources among multiple cores and GANNS's poor spatial and temporal locality (*e.g.,* 15-26% last-level cache hit rate) lead to frequent DRAM access. On the other hand, the frequently fine-grained (100-128 Byte) yet random graph traversal (*i.e.,* the indirect pointer chasing memory access [25]) is unfriendly to DRAM since it causes the row buffer inside the DRAM to be repeatedly charged and discharged [31].

For the computation part, the CPU contains tens of cores (*e.g.,* 8-64) that can be flexibly configured for both data and pipeline parallelism. However, the CPU-based system cannot support both of them simultaneously under the batch mode due to the limited number of cores (violate Figure 2 ④-⑦). Moreover, we observe extremely poor performance scalability of the CPU-based system, as shown in Figure 3(b). Taking SIFT10M as an example, the QPS of eight threads is only 2.0× higher than that of the single-thread execution. Besides, the 32-thread QPS is about 2.8-3.4× higher than the single-thread QPS on average, which suggests great optimization potentials compared with the ideal linear-scaling case.

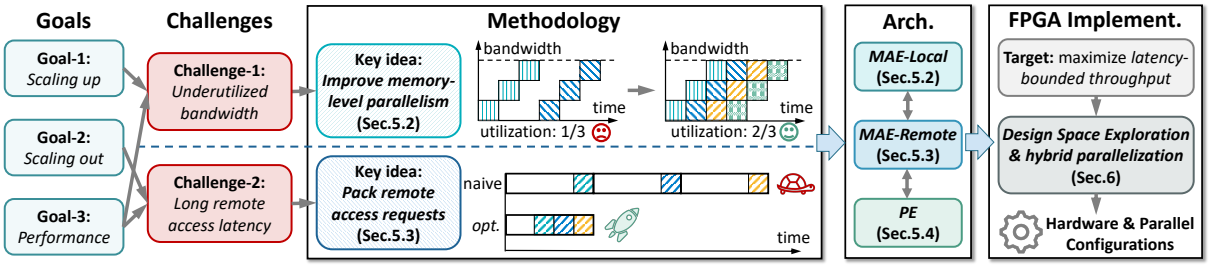


Figure 6: We aim at designing a customized architecture, DF-GAS, with the goals of scaling up, scaling out, and high performance for billion-scale GANNS. We propose a feature-packing memory access engine architecture to deal with the challenges of underutilized bandwidth and long remote access latency. To maximize latency-bounded throughput, DF-GAS is implemented on the distributed FPGA system by employing system-level DSE with hybrid parallelization.

3.2.3 GPU-based design. GPUs provide massive data parallelism with thousands of lightweight CUDA cores. Besides, GPUs provide high memory bandwidth (*e.g.,* Nvidia 3090 provides 936.2GB/s bandwidth with GDDR6X) with a relatively smaller memory capacity (*e.g.,* 24GB on Nvidia 3090). However, we observe that the GPU-based system suffers the same issue as the CPU-based system. Because GPUs cannot manage the fine-grained yet random memory access patterns efficiently, GPU-based GANNS is also bounded by the memory access, which takes ~70% of the total runtime, and the memory bandwidth is underutilized (*i.e.,* 5-40%), as shown in Figure 4(b).

Furthermore, GPU lacks sufficient pipeline parallelism for sorting operations and the entire dataflow, since streaming multiprocessors (SMs) of GPUs are not optimized for pipeline parallelism (violate Figure 2 ④⑦). The limited on-chip resources per SM make it difficult for GPUs to support pipelining, as it introduces heavy inter-SM communication and synchronization overheads. As a result, GPUs take 4.2-47.6× latency compared to CPU-based systems, and the problem will be further exacerbated when the batch size becomes larger. Figure 4(a) shows that when the batch size increases from 1 to 4096, the latency deteriorates by 5.5×, and the latency will exceed the 10ms latency constraints when the batch size is larger than 1024 on the SPACEV1B dataset.

3.2.4 FPGA-based design. Peng *et al.* [36] propose an FPGA-based GANNS accelerator for molecular similarity search. However, they cannot resolve the underutilized memory bandwidth and support billion-scale datasets (violate Figure 2 ①③⑥). Kim *et al.* [22] propose a GANNS accelerator based on the SmartSSD platform. In SmartSSD, the DRAM capacity is only 4GB, and the SSD bandwidth is only 4GB/s, resulting in frequent but costly I/O access between the SSD and DRAM. Thus, this work can only achieve 75.6 QPS with 0.94 recall on SIFT1B.

3.3 Challenges of FPaaS-based GANNS Systems

Thanks to the high memory bandwidth of HBM, large memory capacity of DDR, and high reconfigurability of FPGA, FPaaS becomes a potential solution for billion-scale GANNS in the cloud. However, two serious challenges hinder the adoption of FPaaS in GANNS systems.

3.3.1 Challenge-1: The underutilized local memory bandwidth. In addition to CPUs and GPUs, accessing fine-grained (*e.g.,* 128B) and random data can also result in severely underutilized memory bandwidth (*e.g.,* 4%/6% for DDR/HBM) on FPGAs. We evaluate the memory access distribution on SIFT10M/1B. Results show that the ***total*** request data size is about 2-7KB per iteration (*i.e.,* red zone in Figure 5(a)). If we can pack all the requests of each iteration into an outstanding request from 128B (one request) to 4KB (multiple requests), the memory bandwidth can be increased by about 13/8× for DDR/HBM. But there still exists a gap (*i.e.,* 55-65%) from the peak memory bandwidth.

3.3.2 Challenge-2: The long remote access latency and heavy communication overhead. To process billion-scale datasets, it is non-trivial to deal with the long remote access latency on the distributed FPaaS system. Figure 5(b) presents the latency of local and remote memory access. The latency of the remote access using the user datagram protocol (UDP) network stack [4] is about 3-8× than that of the local access. Besides, the round-trip remote access latency reaches around 1-2μs, which is almost comparable to the latency of a single GANNS iteration. As discussed earlier, the communication overhead can take up around 85% of the total end-to-end latency.

4 DESIGN METHODOLOGY OVERVIEW

The goals of FPaaS-based billion-scale GANNS systems are **scaling up, scaling out, and high performance**.

- ***Scaling up:*** A memory-efficient architecture is needed to improve memory bandwidth utilization and provide linear scaling-up ability.
- ***Scaling out:*** A communication-efficient parallel scheme is required to reduce inter-node communication overhead and provide scaling out ability while adhering to SLA constraints.
- ***High performance:*** The GANNS architecture should ensure high performance through data and pipeline parallelism, *i.e.,* maximizing latency-bounded throughout under given constraints.

Figure 6 shows the design methodology overview of DF-GAS to achieve these goals. To tackle *Challenge-1*, our first key idea is to improve the memory-level parallelism, increasing the bandwidth utilization from 35-45% to 77-81%. To tackle *Challenge-2*, our second key idea is to pack multiple discontinuous requests into one request,

reducing the communication overhead by 76.2%. To enable these ideas, we implement a customized GANNS architecture, and verify its effectiveness on the FPaaS system. Moreover, we maximize the latency-bounded throughput through system-level design space exploration considering a hybrid parallel search scheme.

5 DF-GAS ARCHITECTURE

5.1 Architecture Overview

We design a customized domain-specific architecture to accelerate GANNS on the distributed FPGA cluster. Figure 7(a) shows the overview of DF-GAS architecture. FPGAs are connected with each other using the 2x100Gbps network stack [4]. Inside each FPGA, there are multiple PEs and feature-packing MAEs supporting both local and remote access optimizations (Section 5.2 and 5.3).

Each PE has several distance compute units (DCUs), local top-k sorting units (LSUs), an optional global top-k sorting unit (GSU) (Section 5.4), and a controller with RISC-V compatible ISA (Section 5.5).

The customized feature-packing MAE is designed to optimize both local and remote access, as shown in Figure 7(c). MAE contains four key modules for managing instructions and data movements:

- The ***R/W-CMD*** module reads the commands (*e.g.*, `start` and `init`) from the host CPU (ⓓ) to the controller and writes the `sync` command to remote FPGAs (ⓐ) for the synchronization.
- The ***R/W-Neighbor.id*** module is to read the neighbor list N_{id} of the point to be traversed (V_{id}) returned by PEs (ⓙ) and then check the hash visited RAM (HVR) for filtering out visited N_{id} (ⓖ). Besides, this module also manages data prefetching for local and remote access and optionally stores remote N_{id} into the shared RAM (ⓕ) for delayed remote feature access.
- The ***R/W-Feature*** module fetches features of unvisited nodes from local (ⓑⓒ) or remote memory (ⓐ) and then sends them to PEs (ⓚ) for distance computation.
- The ***R/W-Output*** module is to read/write the partial results ($\{V_{id}, Dist.\}$) from/to the remote GSU (ⓛ) under the *sub*-GPS scheme and write the final result back to the host CPU (ⓓ).

All the four modules are connected with the corresponding direct memory access (DMAs) through a request dispatcher. The request dispatcher is connected to Network ⓐ, DDR ⓑ, HBM ⓒ, and PCIe ⓓ. A local scheduler controls the four modules (ⓘ) with RISC-V compatible ISA sent by the controller (ⓜ).

The search process begins with the host CPU uploading the bitstream to FPGAs, and placing the graph database (*i.e.,* features V and indexes V_{id}) into the main memory. Then, the host CPU sends an `init` command to configure DF-GAS (*e.g.,* iterations ef and batch size B). After that, a `start` command with a batch of queries (Q) is sent to DF-GAS for searching. During each iteration, MAEs are in charge of loading graph data (*i.e.,* neighbor lists N_{id}s and their features F) from both local and remote memory. When features F are loaded, they are passed to PEs for distance calculation and top-k sorting. After ef iterations, the final top-k results ($\{V_{id}, Dist.\}$) are written back to the host CPU.

5.2 Scaling Up: MAE for Local Access

5.2.1 Two-level outstanding request generation. The underutilized memory bandwidth is mainly caused by the fine-grained and irregular feature access (100-128B), as discussed in Section 3. Inspired by Figure 5(a), the key idea of improving bandwidth utilization is to enlarge the total feature volume of continuous outstanding requests. To this end, we propose an outstanding request generation technique that employs two-level feature packing.

Firstly, MAE employs an *intra-PE* data-level pipeline scheme. The ***intra-PE merger*** in ***R/W-Feature*** merges all the feature requests of one PE to increase the memory-level parallelism at a fine-grained level (Figure 8(a)). Besides, the sub-modules of MAE are connected using streaming-based FIFOs, achieving a pipeline initiation interval (II) of one. For each iteration, MAE increases the total data amount of a single outstanding request (*e.g.,* from 128B to 8KB), improving data access throughput (*i.e.*, memory-level parallelism) by 10-12× while introducing a little latency overhead (≤ 5%) for each request.

Secondly, MAE utilizes *inter-PE* packing with out-of-order outstanding request generation to increase the memory-level parallelism at a coarse-grained level. The *inter-PE* packing is executed by the ***inter-PE OoO generator*** in ***R/W-Feature***. However, maintaining the dependency of requests needs additional information as context, which introduces non-trivial control overhead. To deal with that, we design a 64-bit tag to store the context that includes request information, *e.g.,* iteration rounds, number of feature packages, etc. The tag is embedded into the *intra-PE* outstanding request package. Thus, the context-switching overhead is greatly reduced to less than 1%. Furthermore, we implement a score-board module in MAE to maintain the dependency and synchronization for the issued OoO outstanding requests. By applying the *inter-PE* packing with the OoO design, MAE increases the total data amount with massive memory-level parallelism (*e.g.,* 8KB to 128KB with 16 PEs). In all, the bandwidth utilization can be increased from 35-45% to 77-81%.

5.2.2 Pipeline optimization for latency hiding. To efficiently hide the local access latency, we propose a task-level pipeline scheme based on five sub-tasks in one search iteration (Figure 8(b)). The five sub-tasks include loading N_{id}, loading feature F, distance calculation, Top-k sorting, and queue update. The task-level pipeline can reduce the latency of one iteration by 24-32%.

Furthermore, we propose a local data prefetching technique to further hide the latency, where the idea is to pack the data of two adjacent iterations, as shown in Figure 8(c). To be specific, ***R/W-Neighbor.id*** in MAE fetches the neighbor lists of the iteration i and $(i+1)$ at the same time, where the $(i+1)$-th neighbor list is predicted from the PE's local top-k queue (ⓙ). The controller monitors the results of the $(i-2)$ and $(i-1)$ iterations to check whether the prediction is correct. Once there is a miss, the controller sends the `flush` command to the ***R/W-NeighborIndex***, ***R/W-Feature***, and PEs (ⓜ) for flushing FIFOs and computation units. Then MAE reloads the correct data and sends them to PEs to perform another $(i+1)$-th iteration. The local data prefetching technique further reduces the iteration latency to 44-48% of the original design. In practice, combining the two techniques improves the end-to-end performance by 25-30%.

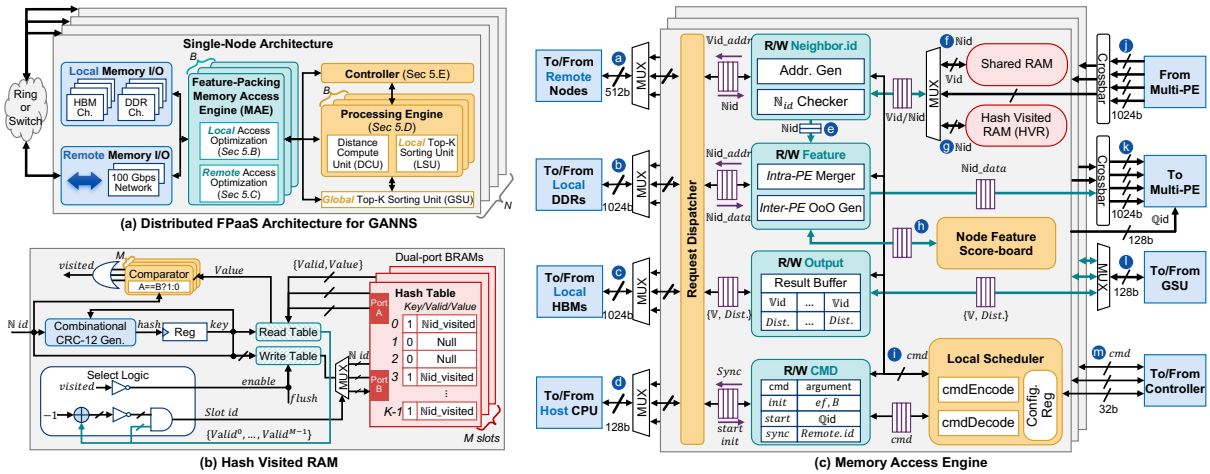


Figure 7: (a) Hardware architecture overview of DF-GAS. (b) Hash visited RAM implemented on CRC functions and fixed-length slot-based table. (c) Feature-packing MAE with local and remote memory access optimization.

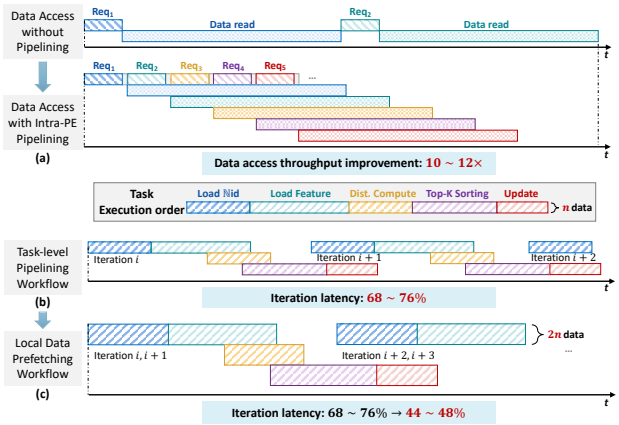


Figure 8: (a) Intra-PE data-level pipelining technique. (b) Task-level pipelining workflow. (c) Local data prefetching with iteration-level data packing.

5.2.3 Hardware optimization for visited lists. A visited list is used to check whether a point has been visited or not, thus avoiding repeated computations. Prior work [22] utilizes the single-bit tag-based optimization, *i.e.,* using one bit to denote whether a point has been visited or not. This optimization reduces the required on-chip memory by 32×. However, it is still impractical for billion-scale datasets, which require 250MB on-chip memory for single-bit tags.

To tackle this issue, we propose a hash visited RAM (HVR), as shown in Figure 7(b). HVR utilizes CRC-12 function [45] to generate the hash key, which is used to read out M values from an M-slot hash table. The input N_{id} is compared with M values simultaneously to generate the *visited* signal. If N_{id} has not been visited, the select logic generates the next insert location (*Slot id*) based on M valid signals and then inserts the N_{id} into the target slot. HVR utilizes eight dual-port BRAMs with $K = 1024$ buckets per slot and $M = 4$ slots, reducing the required on-chip memory to 32KB. HVR employs double buffering to hide the latency for initialization, which happens for every query. Besides, the latency of the search and insert operation in HVR is about 2-3 cycles per point. Evaluations show the average conflict rate is about 0.1%, resulting in negligible latency overhead (≤0.05%) with no accuracy loss since conflicts only cause redundant visits without changing the graph traversal path.

5.3 Scaling Out: MAE for Remote Access

5.3.1 Delayed remote feature access. To relieve the remote feature access overhead, an intuitive solution is to reduce the number of remote access. The original remote access occurs when the required data is stored in other nodes. As Figure 9(a) shows, a large number of consecutive and fine-grained feature accesses cause significant long remote access latency and low bandwidth utilization. Aiming at this problem, we propose a delayed remote access strategy to reduce the number of remote access. The delayed remote access strategy pends the former remote accesses and merges them with the later ones into coarse-grained access. A basic example is shown in Figure 9(a). In the skip phase, we pend the current remote access and temporarily store this access locally. In the sync phase, each node sends its locally stored remote accesses to different remote nodes through the network. We define the number of iterations between two adjacent sync phases as the communication interval. The communication interval length influences the nodes' access order during the search, causing slightly longer search steps. The communication interval is determined for each dataset during the construction stage. The end-to-end performance can be improved by 1.69-1.86× with the same recall.

5.3.2 Prefetching remote neighbor list. In order to reduce the remote neighbor list access latency, we propose a remote neighbor list prefetching technique. In GANNS, we use a priority queue to track the closest nodes to the query. During each search iteration, the head of the priority queue is popped, whose neighbors will be traversed. In the prefetching technique, if we need to access one

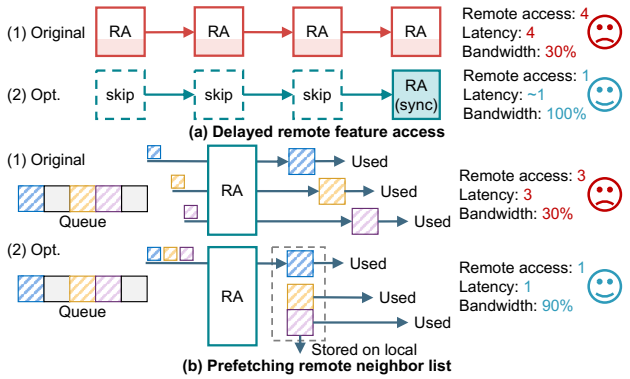


Figure 9: Optimizations for parallel search: (a) Delayed remote feature access. (b) Prefetching remote neighbor list. *RA* **denotes remote access.**

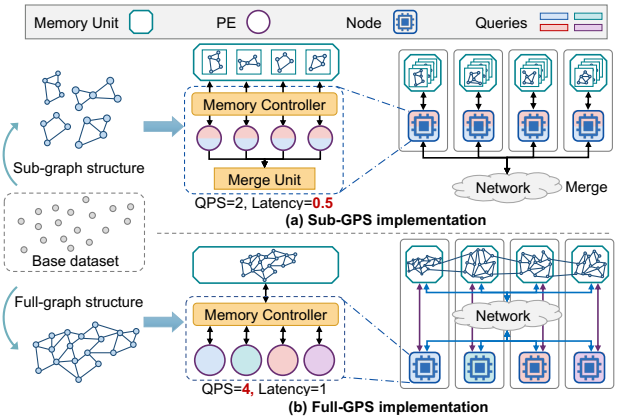


Figure 10: Two parallel schemes: (a) *sub*-GPS and (b) *full*-GPS of the *intra*-node and *inter*-node design.

neighbor list from a remote node X, we first traverse the priority queue and select the points whose neighbor lists are also stored in X. Then we send all of these neighbor list requests to X. As shown in Figure 9(b), the original method needs three remote accesses for three different neighbor lists. While in our prefetching techniques, we traverse the priority queue and find that these neighbor lists are stored in one remote node. Then it takes only one remote access to get these neighbor lists with a higher bandwidth utilization rate. The prefetching technique is also enabled by an interesting phenomenon: the GANNS algorithm can approach the query very quickly, then *search around the query for most of the remaining iterations.* In the later phase, which takes most iterations, points in the queue are seldom updated, guaranteeing a high hit rate of prefetching. The number of remote neighbor list access is decreased by 78.2-82.1%, with 1.59-1.68× end-to-end performance improvement.

5.4 High Performance: PE Design

5.4.1 Distance computing unit with data parallelism. The role of DCU is to perform the distance calculation $d(q, p)$ between the D-dimensional query q and point p. DCU employs a pipelined sum reduction structure with data parallelism, which consists of N_D compute units with registers and an adder tree with $N_D - 1$ adders. It takes DCU D/N_D cycles to process a single point with N_D data parallelism.

5.4.2 Top-k sorting unit with pipeline parallelism. This unit is in charge of maintaining the k nearest neighbors and updating candidate indexes for the next iteration. We implement the top-k priority queue using a serial sorting structure [18], which can process one input per cycle with minimal resource overhead. It utilizes $2k$ registers with pipeline parallelism for high throughput sorting. Besides, the current nearest point is in the first register. Thus, the next iteration can start as soon as all points are loaded without waiting for the top-k priority queue to be completely sorted.

5.5 Instruction Set Architecture Design

To provide flexibility and modularity for GANNS, we design a RISC-V compatible ISA [43]. We encode the functionality of DF-GAS commands using the customized opcode set [42], where the bits [6:0] are used. Then, we encode the arguments (*e.g.,* batch size, iteration) into the remaining space to support different commands. The number of DF-GAS commands is 15, leaving room for supporting new GANNS variants, as GANNS algorithms are still developing.

6 SYSTEM-LEVEL OPTIMIZATION

At the system level, we propose the hybrid parallel search and design space exploration based on DF-GAS to maximize the latency-bounded throughout of distributed systems.

6.1 Hybrid Parallel Search Scheme

We take Figure 10 as a simple example to show the implementation of *sub*-GPS and *full*-GPS on *intra*-node and *inter*-node, respectively.

6.1.1 Latency-optimized sub-GPS. In *sub*-GPS scheme, different PEs in one node or different nodes perform searching on different sub-graphs simultaneously, as shown in Figure 10(a). Then the search results of these sub-graphs are merged to get the final results. Because one query is processed in parallel, the *sub*-GPS has the virtue of low search latency. However, the *sub*-GPS introduces more computations[1] than the *full*-GPS to achieve the same accuracy, reducing the search throughput.

6.1.2 Throughput-optimized full-GPS. In *full*-GPS scheme, different PEs or nodes share one large-scale graph, as shown in Figure 10(b). Because each PE can process different queries simultaneously, the *full*-GPS can provide higher throughput. However, due to the intra-/inter-node bandwidth limitation, the query latency of one point may become longer when other points occupy the memory bandwidth.

6.1.3 Combine together. Full-GPS has a throughput advantage, while *sub*-GPS has a latency advantage. Therefore, to achieve the best latency-bounded throughout, we propose a hybrid parallel search scheme for billion-scale datasets on distributed systems, which can have more options on the latency-throughout tradeoff.

[1]For example, return multiple Top-k values of each sub-graph, rather than the Top-k of the full-graph

Table 1: Hardware configurations of CPU, GPU, ANNA [23], Vstore [27], and FPGA. (*BW* for bandwidth)

	CPU	GPU	ANNA	Vstore	FPGA
Platform	Intel Xeon Gold 5218	Nvidia RTX 3090	TSMC 40nm	Xilinx XC7Z045	Xilinx Alveo U280/U250
Compute Units	16 cores	10496 CUDAs	1024 MACs	512 MACs	1024 DSPs
Frequency	2.3 GHz	1.4 GHz	1 GHz	200MHz	300 MHz
Memory	64GB	24GB	32GB	32GB&1TB	40/64GB
BW (GB/s)	102.4	936.2	75	107.3&76.8	498.4/76.8

Table 2: Hardware resource utilization on U250 and U280.

FPGA	LUT	FF	BRAM	URAM	DSP
U280	79.7%	51.9%	56.1%	26.7%	11.3%
U250	61.1%	38.2%	47.5%	20.0%	8.3%

Specifically, we can configure three different parallel methods (from inter-node to intra-node): *full*-GPS to *full*-GPS (*full-full*), *sub-full*, and *sub-sub*. In the billion-scale dataset, the *full-full* parallelism has the potential to provide extremely high throughput up to 333,201 QPS; the *sub-sub* parallelism achieves 51us latency; the *sub-full* parallelism can provide high flexibility of achieving the highest throughput under different SLA constraints.

6.2 Design Space Exploration (DSE)

We propose a system-level DSE approach to automatically explore the optimal hardware configuration and parallel scheme for the distributed FPaaS system. Specifically, a DSE engine first receives the user inputs and constructs the design space that satisfies the resource constraints. Then, the engine searches exhaustively in the design space with an analytical performance model to acquire the QPS and latency of each candidate. At last, the engine selects the design that has the highest QPS and also satisfies the latency constraint.

The performance model contains two levels, *i.e.,* PE and node, to achieve high accuracy (modeling error rate≤4.5%) on the latency and QPS simulation.

- **PE-level:** Suppose we have S PEs in one FPGA node, the PE-level latency LAT_{PE} can be represented as:

$$LAT_{PE} = \max_{b=1}^{B} \left(LAT_{Query_b} \right) + PS_2 \times T_{im}, \tag{3}$$

  where LAT_{Query_b} is the latency of executing one query point b, PS_2 denotes the choice of parallel scheme in PE-level (*i.e.*, 0 for *full*-GPS and 1 for *sub*-GPS), and T_{im} is the time of merging the top-k distances of B PEs.
- **Node-level:** Similarly, we can represent the node-level latency on multiple FPGA nodes LAT_{Node} as:

$$LAT_{Node} = \max_{i=1}^{N} \left(LAT_{PEi} \right) + PS_1 \times T_c, \tag{4}$$

  where T_c denotes the total communication time, and PS_1 denotes the choice of the parallelization strategy in node-level (*i.e.*, 0 for *full*-GPS and 1 for *sub*-GPS).

7 EVALUATION

7.1 Evaluation Setup

GANNS datasets. We choose two representative ANNS datasets, SIFT [19] and SPACEV [2], with different number of points (*i.e.,* 1M, 10M, 100M, and 1B). We select a representative GANNS algorithm (*i.e.,* HNSW [1]) to construct the graph database. Unless otherwise specified, we evaluate the performance with the *Recall*@10 to be 0.95 for SIFT and 0.9 for SPACEV, as commonly used by previous work [7, 22].

FPGA clusters. We consider two kinds of cloud FPGAs, including Xilinx Alveo U250 (w/o HBM) and U280 (w/ HBM), as shown in Table 1. We set up two 8-node FPGA cluster configurations, including the bandwidth-optimized *all-U280* and capability-optimized *all-U250*. Both of them deliver enough off-chip memory capability for billion-scale datasets, while the million-scale datasets are evaluated on a single-node system. We utilize a switch-based P2P topology for inter-FPGA communication with the 2×100GbE interface.

DF-GAS configurations. By employing system-level DSE on both the U280 and U250 FPGA, DF-GAS is designed with $B = 16$ PEs per FPGA and $N_D = 128$ multipliers per PE. We implement DF-GAS using Verilog and Xilinx Vitis 2021.1 with a frequency of 300MHz. Table 2 shows the resource utilization. We measure the end-to-end performance with host applications, which are implemented using C++ with Xilinx runtime (*xrt*) libraries. We measure the power of DF-GAS (including DDRs and HBMs) through the on-chip power monitor, which can be accessed via the vendor-provided Xilinx Board Utility tool *xbutil* [3].

CPU and GPU baselines. Both the CPU- and GPU-based GANNS systems are evaluated on the 8-node cluster for fair comparisons, and also employ the same switch-based P2P topology with 2x100Gbps networking bandwidth on the same rack. Table 1 shows the configurations of the CPU and GPU server. For the CPU baseline, we choose the 16-core Xeon Gold CPU with 64GB DDR memory and 102.4GB/s bandwidth, and evaluate a SOTA *sub*-GPS multi-CPU design [13]. Since HBM-equipped CPUs are not publicly available yet [20], we simulate the performance of HBM-equipped CPUs with 500GB/s bandwidth (compared to U280) for fair comparisons. The GPU baseline is a SOTA *sub*-GPS multi-GPU design GGNN [14] on Nvidia RTX 3090 with 24GB HBM memory and 936.2GB/s bandwidth. We measure their performance and power by Intel RAPL and *nvprof*.

SOTA architecture baselines. We compare DF-GAS with SOTA FPGA-based designs (PQ-ANN [49] and HPQ-ANN [5]), heterogeneous memory designs (SSD-GANN [22] and HM-ANN [37]), PQ-based ANN accelerator (ANNA [23]), and in-storage GANNS accelerator (VStore [27]). Since all these prior works are evaluated on the platform without HBMs, we also evaluate DF-GAS under the *all-U250* configuration. We scale their performance based on the same memory bandwidth of U250 FPGA for a fair comparison.

7.2 Evaluation Results

7.2.1 Design space exploration. To show the benefits of the hybrid parallel search scheme, we perform system-level DSE on billion-scale datasets under the soft and hard SLA constraint (*i.e.*, 1ms and 0.1ms) with a 5% budget. Figure 12 illustrates that for the soft SLA constraint, both *all-U280* and *all-U250* with *full*-GPS provide

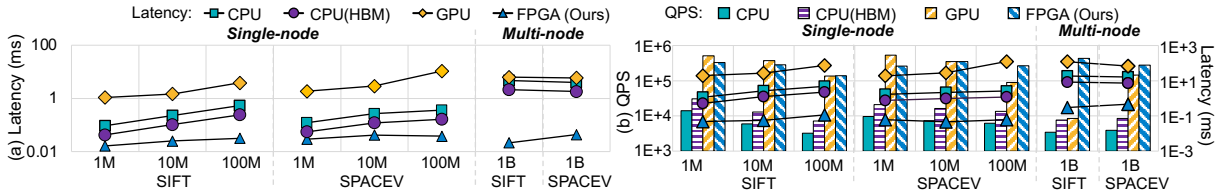


Figure 11: (a) Latency with a batch size of one and (b) maximum throughput without any SLA constraint of CPU, CPU with HBMs, GPU, and DF-GAS.

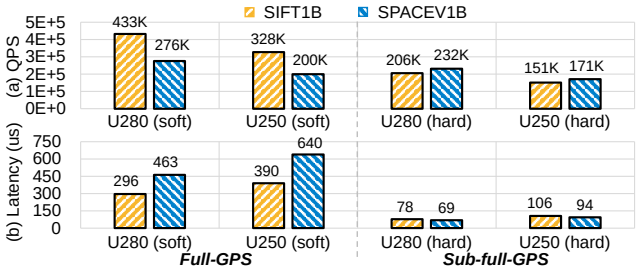


Figure 12: DSE results, (a) QPS and (b) Latency, on *all-U250/U280* clusters under the soft/hard SLA constraint on billion-scale datasets.

the best throughput. For the hard SLA constraint, only employing the hybrid parallel search with *sub-full*-GPS can meet the 0.1ms SLA constraint. Compared with *full*-GPS, *sub-full*-GPS achieves 3.7-6.8× lower latency while providing competitive throughput (with 17-117% reduction). For the comparison between U280 and U250, *all-U280* achieves ~1.4× higher throughput than *all-U250* due to a much higher bandwidth (498.4GB/s *vs.* 76.8GB/s). Thus, we choose *all-U280* with *full*-GPS as the basic configuration for DF-GAS, considering that the 1ms SLA constraint can satisfy most recommender system scenarios [7].

7.2.2 Latency comparison. We compare the latency of CPU, GPU, and DF-GAS with batch size one. Figure 11(a) shows that DF-GAS outperforms CPU and GPU in latency on both million- and billion-scale datasets. Compared with CPU, DF-GAS reduces the single- and multi-node query latency by 4.2-16.9× and 92.1-218.6×, respectively. It is because that DF-GAS makes full use of the high bandwidth (77-81%) of HBM-equipped FPGAs, and provides sufficient parallelism for different operators. Introducing HBMs in the CPU-based system brings 2.2× improvement on average, but DF-GAS still outperforms HBM-equipped CPU by 1.9-98.9×. Compared with GPU, DF-GAS achieves 59.1-293.6× lower latency. Considering that GPU has twice the bandwidth of the U280 FPGA, it further verifies that GPU has poor pipeline parallelism.

7.2.3 Throughput comparison. We compare the maximum throughput of different GANNS systems without any SLA constraint, and show their corresponding latency in Figure 11(b). The batch size of DF-GAS is 16 per node, while the batch size of CPU and GPU is 32 and 4096, respectively. Further increasing the batch size for CPU and GPU will bring marginal throughput improvement but severe latency deterioration. DF-GAS outperforms HBM-equipped CPU in throughput by 11.3-52.2× with 11.9-25.1× lower latency. GPU provides 1.3-2.0× higher throughput over DF-GAS on several million-scale datasets, but with a much higher latency (308.6-2064.3×, ranging from 18.8ms to 124.1ms). Instead, DF-GAS maintains sub-millisecond ultra-low latency consistent with batch size one, ranging from 0.046ms to 0.114ms. For billion-scale datasets, DF-GAS achieves 1.9-50.8× higher throughput with 148.0-406.1× lower latency than GPU.

7.2.4 Latency-bounded throughput. The SLA constraint is usually set within 10ms in recommender systems [13, 24], and a stricter bound (*e.g.*, 1ms) can help to optimize the entire pipeline [8]. Thus, we compare the latency-bounded throughput with 1ms and 10ms SLA constraints. Figure 13(a) shows the results under the 10ms SLA constraint. GPU provides 1.7-1.8× higher latency-bounded throughput than DF-GAS on SIFT1M and SPACEV1M, because GPU can employ a large batch size (2048) to improve the throughput with relatively low shared resource contention when the dataset is small. As the dataset size increases from 10M to 1B, DF-GAS outperforms GPU in latency-bounded throughput by 1.2-29.9×. Besides, GPU fails to meet the 10ms SLA constraint on the SIFT1B dataset. Compared with CPU with and without HBMs, DF-GAS achieves a geometric average of 24.1× and 55.4× higher latency-bounded throughput, respectively. Figure 13(b) shows that GPU cannot meet the 1ms SLA constraint in any dataset other than SIFT1M, and both CPU and GPU cannot meet the SLA constraint on billion-scale datasets. Moreover, Figure 14 shows that DF-GAS consistently outperforms CPU and GPU under different recall on billion-scale datasets.

7.2.5 Comparison with SOTA architecture baselines. Figure 15 illustrates the throughput comparison between SOTA architecture baselines and DF-GAS under different recall. For FPGA-based designs, HPQ-ANN [5] achieves 34.7M QPS on SIFT1M with 0.97 recall. This is because HPQ-ANN utilizes a hierarchical PQ approach to fully store the database in the on-chip BRAMs. However, the performance of HPQ-ANN will degrade rapidly when processing billion-scale datasets, which are too large to be stored on-chip. As for PQ-ANN [49], it achieves 56.5K QPS on SIFT1M with 0.94 recall, and 202.4K QPS with 0.7 recall on SIFT1B. DF-GAS consistently outperforms PQ-ANN with 1.61-1.76× higher QPS, while maintaining a high accuracy (*i.e.*, 0.90-0.96 *Recall*@10) on billion-scale datasets. PQ-ANN compresses datasets into small structured data, thus does not suffer from the inter-node communication overhead and fine-grained random memory access problem faced by GANNS, but at the cost of relatively low accuracy on billion-scale datasets. DF-GAS enables both high performance and high accuracy by utilizing

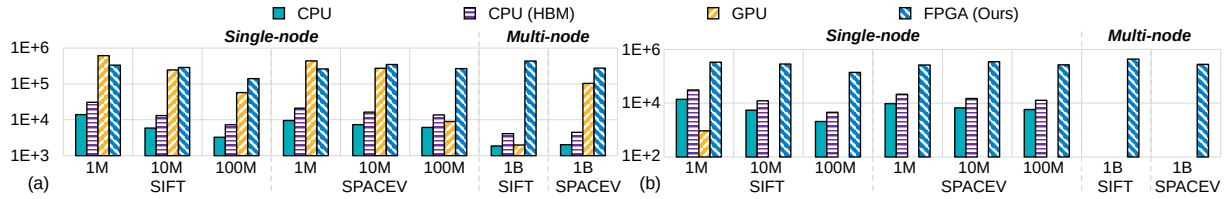


Figure 13: Throughput of CPU, CPU with HBMs, GPU, and DF-GAS with (a) 10ms and (b) 1ms SLA constraint.

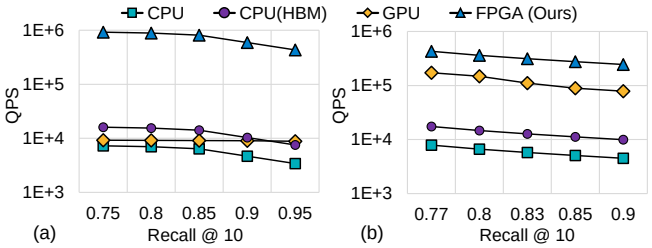


Figure 14: The throughput of CPU, CPU with HBMs, GPU, and DF-GAS under different recall on (a) SIFT1B and (b) SPACEV1B.

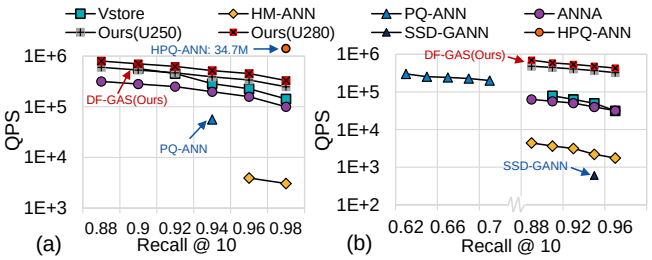


Figure 15: QPS of SOTA architectures and DF-GAS under different recall on (a) SIFT1M and (b) SIFT1B.

feature-packing MAE with remote data prefetching and delayed processing for billion-scale ANNS.

For heterogeneous memory designs, both SSD-GANN [22] and HM-ANN [37] achieve low throughput of 604.7 and 2200 QPS (609.6× and 167.55× lower than DF-GAS) with 0.94 recall on SIFT1B, respectively. The reason is that the limited SSD bandwidth becomes the performance bottleneck. Compared with ANNA [23], DF-GAS offers comparable throughput (2.1× on average) on SIFT1M, and provides superior performance (8.6× on average) on SIFT1B. Compared with Vstore [27], DF-GAS achieves an average of 1.3× and 7.5× higher throughput on SIFT1M and SIFT1B, respectively. The performance advantage of DF-GAS comes from several factors: (1) DF-GAS packs inter- and intra-PE feature accesses to improve bandwidth utilization. (2) DF-GAS utilizes local data prefetching to perform graph traverse efficiently. (3) DF-GAS employs delayed remote access and remote data prefetching to efficiently hide the long remote access latency.

7.2.6 Energy and cost efficiency. We consider energy efficiency and cost efficiency as fair metrics. The latency-bounded throughput with a 10ms SLA constraint are used as the performance reference.

Table 3: Power consumption of hardware platforms.

	CPU	CPU×8	GPU	GPU×8	FPGA	FPGA×8
Power	100 W	840 W	165 W	1376 W	57 W	488 W

Table 3 shows the power consumption of CPU-, GPU-, and FPGA-based GANNS systems. Figure 16(a) shows the results of energy efficiency (QPS/W). DF-GAS consistently outperforms CPU and GPU with 41.8-401.7× and 1.6-86.5× better energy efficiency, respectively. For cost efficiency (QPS/(cost unit)), Based on the same memory capacity, we normalize the costs of GPU, FPGA, and host server to 3, 6, and 10, respectively. Thus, the relative costs of CPU, CPU+GPU, and CPU+FPGA server are 10, 13(=10+3), and 16(=10+6), respectively. Figure 16(b) shows that GPU delivers a 2.1-2.3× higher cost efficiency than DF-GAS on SIFT1M and SPACEV1M. For larger datasets over 10M points, DF-GAS provides 14.9-145.8× and 2.0-24.3× better cost efficiency over CPU and GPU, respectively.

7.2.7 Scalability. We compare the multi-node scalability of CPU, GPU, and DF-GAS on billion-scale datasets. Figure 17 shows that DF-GAS achieves near-linear scalability, with only ~10% difference compared to the ideal linear case. Besides, DF-GAS delivers 3.0-3.5× and 2.0-5.1× better scalability over CPU and GPU, respectively. The poor scalability of CPU and GPU illustrates the influence of low bandwidth utilization and long remote access latency on billion-scale GANNS. Moreover, it verifies the effectiveness of the proposed *full*-GPS scheme, with data prefetching and delayed processing to reduce communication overhead by 76.2%.

7.2.8 Ablation studies. Figure 18 illustrates the improvement breakdown of the proposed techniques on the SIFT1B dataset. The throughput are normalized to DF-GAS with all optimization techniques. The original design with pipeline takes up 8.5%. Based on that, the feature packing with two-level outstanding request generation improve the original design by 20.5%. Then, prefetching local data provides another performance improvement of 23.2%. The delayed remote feature access and remote neighbor list prefetching further improve performance by 10.8% and 37.1%, respectively. Thus, all the four techniques provide a total of 11.8× performance improvement over the original design.

For networking sensitivity analysis, Figure 19(a) shows that the P99 tail latency of DF-GAS is 0.025ms and 0.051ms on SIFT1B and SPACEV1B, respectively. It verifies that DF-GAS can satisfy the SLA requirements for most queries.

To gain insight into the architecture optimizations, we try to apply the proposed local and remote prefetching techniques for CPU and GPU baselines. Figure 19(b) shows that adding local prefetching

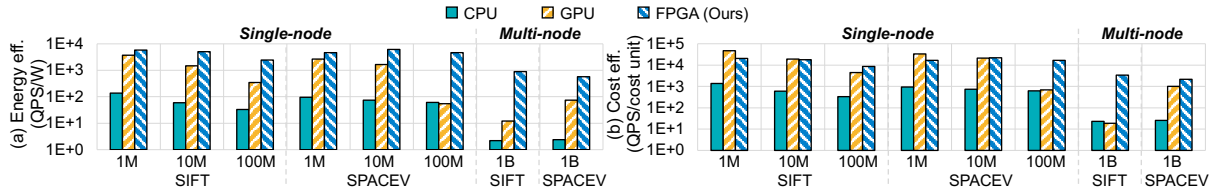


Figure 16: (a) Energy efficiency and (b) cost efficiency.

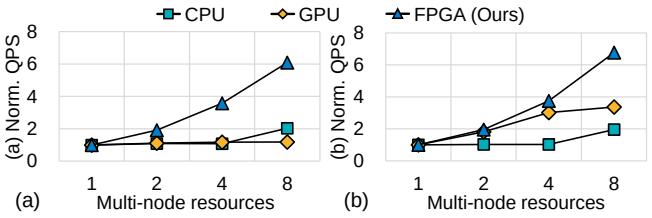


Figure 17: Multi-node scalability analysis on (a) SIFT1B and (b) SPACEV1B.

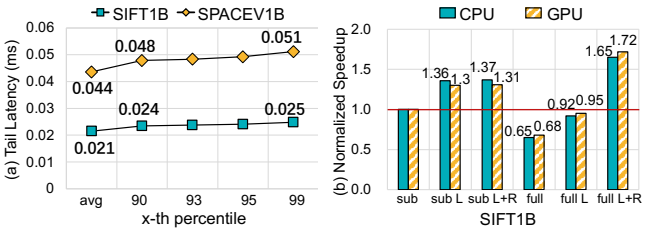


Figure 19: (a) Tail latency of DF-GAS. (b) The normalized speedup of applying the proposed local (L) and remote (R) prefetching techniques onto CPU and GPU baselines with *sub*-GPS and *full*-GPS designs.

to the original *sub*-GPS CPU/GPU design [13, 14] can improve performance by 1.36/1.3×. While adding remote prefetching gives no further improvements, since there is almost no inter-node communication in the *sub*-GPS design. However, if we switch the parallel search mode to *full*-GPS, employing the two techniques can improves the performance from 0.65/0.68× to 1.65/1.72×, showing that the proposed methods can optimize the distributed parallel computing. Furthermore, it demonstrates that the main performance improvement of DF-GAS stems from the joint optimization of the micro-architecture and parallel scheme.

8 RELATED WORK

Software-based ANNS Acceleration. Accelerating ANNS on software has been well studied in the community. At the beginning, researchers employ hash-based (*e.g.*, locality-sensitive-hashing [10, 38] and learning-based hash [29]) or tree-based (*e.g.*, R-tree [16], KD-tree [12], and VP-tree [46]) techniques to optimize ANNS. Later, many product quantization based ANNS algorithms [6, 15, 21] have been proposed to improve the quality of codebook. Recently, graph-based ANNS algorithms achieve the best performance and accuracy on large-scale datasets [26, 28], *e.g.*, HNSW [32] and NSG [13].

Hardware-based NNS Acceleration. FEARY [47] and CHIP-KNN [30] focus on accurate NNS instead of GANNS. Accurate NNS is hard to deal with billion-scale datasets because of low search efficiency. In addition to the GANNS accelerators [22, 36], Zhang *et al.* [49] design a PQ-based accelerator on the OpenCL-based FPGA with a specialized quantization method, which achieves 0.02ms latency with 0.94 *Recall*@10 on the SIFT1M dataset. However, it cannot achieve high accuracy (*e.g.*, 90%) on most billion-scale datasets. Abdelhadi *et al.* [5] propose a modified PQ-based ANNS algorithm to fit as much graph data as possible into the on-chip BRAMs of FPGAs, but it is limited to million-scale datasets. Recently, ANNA [23] explores a specialized dataflow pipeline for PQ-based ANNS to support million- and billion-scale datasets, but ignoring the performance potential of distributed multi-node parallel searching. HM-ANN [37] explores prefetching methods on the heterogeneous memory, *i.e.*, from SSD to DRAM. DF-GAS explores another dimension of prefetching strategies, *i.e.*, from DRAM to PE, and achieves 168× performance gains by incorporating micro-architectural optimization. Vstore [27] proposes an in-storage GANNS accelerator by exploiting data reuse between queries. Since our proposed techniques are independent of correlation between queries, we can combine the techniques of Vstore with DF-GAS to realize further performance improvement.

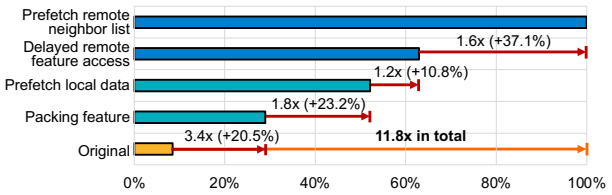


Figure 18: Improvement breakdown of DF-GAS.

9 CONCLUSION

Graph-based approximate nearest neighbor search is a key operation for recommender systems, demonstrating superior accuracy and performance over other ANN algorithms. We build up the first distributed FPaaS system for GANNS named DF-GAS. which enables billion-scale GANNS with million-QPS throughput and sub-millisecond ultra-low latency for the first time. By overcoming the challenges of underutilized bandwidth and long remote access latency, DF-GAS provides 55.4/32.2× latency-bounded throughput improvement over CPU and GPU on billion-scale GANNS, respectively.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No. U19B2019, 62325405, U21B2031, 61832007, 62104128, 62204164), and Beijing National Research Center for Information Science and Technology (BNRist). This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] 2022. hnswlib. [Online]. https://github.com/nmslib/hnswlib.
[2] 2022. SPACEV datasets. [Online]. https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B.
[3] 2022. Xilinx Board Utility Tool. [Online]. https://xilinx.github.io/XRT/2021.1/html/xbutil2.html.
[4] 2022. Xup vitis network example (vnx). [Online]. https://github.com/Xilinx/xup_vitis_network_example.
[5] Ameer MS 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 (ICFPT)*. IEEE, 90–98.
[6] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 931–938.
[7] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighborhood Search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.
[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.
[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.
[10] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th annual symposium on Computational geometry*. 253–262.
[11] DW Dearholt, N Gonzales, and G Kurup. 1988. Monotonic search networks for computer vision databases. In *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, Vol. 2. IEEE, 548–553.
[12] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)* 3, 3 (1977), 209–226.
[13] 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.
[14] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik Lensch. 2022. Ggnn: Graph-based gpu nearest neighbor search. *IEEE Transactions on Big Data* (2022).
[15] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.
[16] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.
[17] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.
[18] Flemming Hoeg, Niels Mellergaard, and Jørgen Staunstrup. 1994. The priority queue as an example of hardware/software codesign. In *Third International Workshop on Hardware/Software Codesign*. IEEE, 81–88.
[19] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.
[20] Hong Jiang. 2022. Intel's Ponte Vecchio GPU: Architecture, Systems & Software. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 1–29.
[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[22] Ji-Hoon Kim, Yeo-Reum Park, Jaeyoung Do, Soo-Young Ji, and Joo-Young Kim. 2022. Accelerating Large-Scale Graph-based Nearest Neighbor Search on a Computational Storage Platform. *IEEE Trans. Comput.* 01 (2022), 1–1.
[23] 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 *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 169–183.
[24] 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 *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3181–3189.
[25] Shuangchen Li, Dimin Niu, Yuhao Wang, Wei Han, Zhe Zhang, Tianchan Guan, Yijin Guan, Heng Liu, Linyong Huang, Zhaoyang Du, et al. 2022. Hyperscale FPGA-as-a-service architecture for large-scale distributed graph neural network. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 946–961.
[26] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.
[27] Shengwen Liang, Ying Wang, Ziming Yuan, Cheng Liu, Huawei Li, and Xiaowei Li. 2022. VStore: in-storage graph based vector search accelerator. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 997–1002.
[28] Jun Liu, Zhenhua Zhu, Jingbo Hu, Hanbo Sun, Li Liu, Lingzhi Liu, Guohao Dai, Huazhong Yang, and Yu Wang. 2022. Optimizing Graph-based Approximate Nearest Neighbor Search: Stronger and Smarter. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 179–184.
[29] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2074–2081.
[30] Alec Lu, Zhenman Fang, Nazanin Farahpour, and Lesley Shannon. 2020. CHIP-KNN: A configurable and high-performance k-nearest neighbors accelerator on cloud FPGAs. In *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 139–147.
[31] Alec Lu, Zhenman Fang, Weihua Liu, and Lesley Shannon. 2021. Demystifying the memory system of modern datacenter FPGAs for software programmers through microbenchmarking. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 105–115.
[32] 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.
[33] Microsoft. 2022. Bing Vector Search. [Online]. https://www.microsoft.com/en-us/ai/ailab-vector-search.
[34] Microsoft. 2022. Sptag: A library for fast approximate nearest neighbor search. [Online]. https://github.com/microsoft/SPTAG.
[35] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).
[36] Hongwu Peng, Shiyang Chen, Zhepeng Wang, Junhuan Yang, Scott A Weitze, Tong Geng, Ang Li, Jinbo Bi, Minghu Song, Weiwen Jiang, et al. 2021. Optimizing FPGA-based Accelerator Design for Large-Scale Molecular Similarity Search (Special Session Paper). In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.
[37] Jie Ren, Minjia Zhang, and Dong Li. 2020. Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. *Advances in Neural Information Processing Systems* 33 (2020), 10672–10684.
[38] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).
[39] Godfried T Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.
[40] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.
[41] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*. 1785–1797.
[42] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovi. 2014. *The risc-v instruction set manual. volume 1: User-level isa, version 2.0*. Technical Report. California Univ Berkeley Dept of Electrical Engineering and Computer Sciences.
[43] Andrew Shell Waterman. 2016. *Design of the RISC-V instruction set architecture*. University of California, Berkeley.
[44] Xilinx. 2022. Xilinx Alveo U280 FPGA Datacenter Acceleration Card. https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.
[45] Fumito Yamaguchi and Hiroaki Nishi. 2013. Hardware-based hash functions for network applications. In *2013 19th IEEE International Conference on Networks (ICON)*. IEEE, 1–6.

[46] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, Vol. 66. SIAM, 311.

[47] Chaoliang Zeng, Layong Luo, Qingsong Ning, Yaodong Han, Yuhang Jiang, Ding Tang, Zilong Wang, Kai Chen, and Chuanxiong Guo. 2022. {FAERY}: An {FPGA-accelerated} Embedding-based Retrieval System. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 841–856.

[48] Shulin Zeng, Guohao Dai, Niansong Zhang, Xinhao Yang, Haoyu Zhang, Zhenhua Zhu, Huazhong Yang, and Yu Wang. 2022. Serving Multi-DNN Workloads on FPGAs: A Coordinated Architecture, Scheduling, and Mapping Perspective. *IEEE Trans. Comput.* 72, 5 (2022), 1314–1328.

[49] Jialiang Zhang, Soroosh Khoram, and Jing Li. 2018. Efficient large-scale approximate nearest neighbor search on OpenCL FPGA. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4924–4932.