FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters

Wenqi Jiang*
wenqi.jiang@inf.ethz.ch
Systems Group, ETH Zurich

Zhenhao He*
zhe@inf.ethz.ch
Systems Group, ETH Zurich

Shuai Zhang
cheungshuai@outlook.com
Systems Group, ETH Zurich

Kai Zeng
kaizeng.zk@gmail.com
Alibaba Group

Liang Feng
fl237079@alibaba-inc.com
Alibaba Group

Jiansong Zhang
muduan.zjs@alibaba-inc.com
Alibaba Group

Tongxuan Liu
tongxuan.ltx@alibaba-inc.com
Alibaba Group

Yong Li
jiufeng.ly@alibaba-inc.com
Alibaba Group

Jingren Zhou
jingren.zhou@alibaba-inc.com
Alibaba Group

Ce Zhang
ce.zhang@inf.ethz.ch
Systems Group, ETH Zurich

Gustavo Alonso
alonso@inf.ethz.ch
Systems Group, ETH Zurich

ABSTRACT

We present FleetRec, a high-performance and scalable recommendation inference system within tight latency constraints. FleetRec takes advantage of heterogeneous hardware including GPUs and the latest FPGAs equipped with high-bandwidth memory. By disaggregating computation and memory to different types of hardware and bridging their connections by high-speed network, FleetRec gains the best of both worlds, and can naturally scale out by adding nodes to the cluster. Experiments on three production models up to 114 GB show that FleetRec outperforms optimized CPU baseline by more than one order of magnitude in terms of throughput while achieving significantly lower latency.

CCS CONCEPTS

• **Information systems** → **Social recommendation**; • **Hardware** → **Reconfigurable logic and FPGAs**; • **Computer systems organization** → *Distributed architectures.*

KEYWORDS

Recommendation System, Hardware Acceleration, FPGA, GPU

ACM Reference Format:
Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467139

*Both authors contributed equally to this research.

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467139

1 INTRODUCTION

DNN-based recommendation inference comprises a huge portion of the workload in data centers. Thus, it is crucial to optimize its performance to serve these models efficiently. Such optimizations can lead to instant economic benefits through (a) higher recommendation quality since more candidate items can be scored in the same time frame; and (b) reduced energy consumption as a result of the improved inference efficiency. Figure 1 shows the architecture of a classical deep recommendation model for *Click-Through Rate* (CTR) prediction. The input feature vector consists of dense features (e.g., age) and sparse features (e.g., advertisement category). The model translates each sparse feature into a dense embedding vector by looking it up in an embedding table. These vectors are then combined with dense features and fed to several fully-connected (FC) layers before the model outputs the predicted CTR. Although there are alternative architecture designs [1, 5, 6, 17, 18], most recommendation systems are built around two major building blocks, i.e., the embedding tables and the DNN classifier, thus sharing the inference challenges we describe below.

Key Challenges. Due to the embedding table architecture and the need for real-time recommendations, three challenges are faced to build efficient inference systems for recommendations. *First*, the embedding table architecture becomes a performance bottleneck. Due to the tiny size of each embedding vector (usually 4 to 64 dimensions) and the large number of embedding tables (tens to hundreds), the embedding table lookup operations are costly because they induce massive random DRAM accesses, leading to low memory bandwidth utilization and significantly downgraded performance. Even worse, these lookup operations result in extra overhead if one resorts to state-of-the-art machine learning frameworks such as TensorFlow and PyTorch. For example, even in an

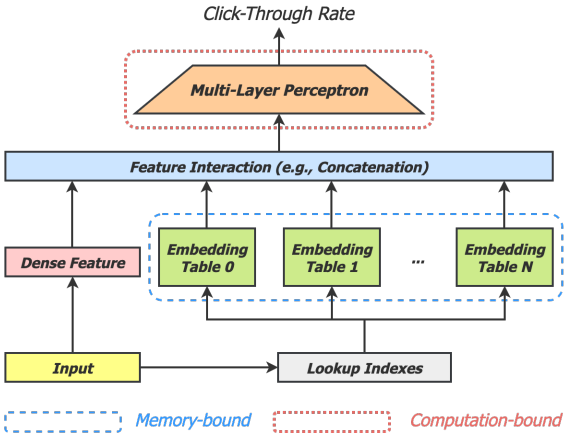

Figure 1: The representative deep recommendation model that we target to accelerate.

inference-oriented framework such as *TensorFlow Serving*, there are tens of types of operators involved in the embedding layer and each operator is invoked multiple times. The many operator invocations significantly degrade performance, especially as small batches are often required for real-time inference. *Second*, the scale of recommendation models can reach over 100 GB since some embedding tables are huge, e.g., account information encodings. Such sizes exclude the option of using hardware accelerators, e.g., FPGAs and GPUs, as the inference engine because of the lack of memory on the device. *Third*, the latency requirement is stringent (usually tens of milliseconds). Large batch sizes usually lead to better throughput for CPUs and GPUs because of the better utilization of the single instruction multiple data (SIMD) architecture and the amortization of function call overheads. In real-time recommendation systems, the *Performance Metric* is throughput under service-level agreement (SLA) constraints, limiting the batch sizes usable in practice. Although huge batch sizes are beneficial for CPUs and GPUs to improve throughput (inferences per second), recommendation systems require small batches due to the latency constraints.

Although much effort has been invested into accelerating deep recommendation models [4, 9, 10, 13, 14], they all fail to solve some of the challenges above, making them suitable only for a subset of use cases. For instance, Gupta et al. [4] suggests GPUs could be useful in recommendation for large batches compared to regular CPU-based engines, but the embedding performance bottleneck remains and GPUs cannot serve large models for the lack of memory capacity. Similarly, hybrid CPU-GPU and CPU-FPGA designs are evaluated but without solving the memory bottleneck Hwang et al. [9]. Jiang et al. [10] resort to the high-bandwidth memory (HBM) available on FPGAs for high-performance embedding lookups, but its applicability is heavily limited to small models because of the 8 GB of HBM available on the board. Kwon et al. [14] and Ke et al. [13] propose to redesign DRAM at the micro-architectural level; however, it takes years to put such new DRAM chips in production even if they are eventually adopted, making the solution interesting from a research perspective but not from a practical stand point.

Our Goal. We target to build an end-to-end high-performance recommendation inference system that can (a) achieve high throughput (inferences per second) under SLA (latency) constraints of tens of milliseconds, and (b) adapt for various models with minimal usage of hardware devices (the model sizes can range from hundreds of MB to hundreds of GB, and the workload characteristic can be embedding-lookup-intensive or computation-intensive).

Our Approach. Based on the careful analysis of three production-scale models, we design and implement *FleetRec*, a high-performance and configurable heterogeneous computing cluster for recommendation inference. On the embedding table lookup side, we resort to (a) FPGAs equipped with high-bandwidth memory (HBM) to enable highly concurrent lookup operations and (b) CPU servers with sufficient DRAM capacity for a few large tables (e.g., tens of GB). On the computation side, we use GPUs exclusively for DNN computation to avoid the irregular memory lookup operations that degrade the SIMD performance. These hardware resources (GPUs, FPGAs, and CPU servers) are regarded as end devices connected through a high-speed network (100 Gbps per link), so that one can configure the node type and quantity to support various size scales (up to hundreds of Gigabytes), number of embedding tables, and computation density.

Key Results. We evaluate FleetRec on three production models from Alibaba covering size scales from 1 GB to over 100 GB. FleetRec achieves 15.5~49.0× speedup in terms of throughput over the CPU-baseline and 7.4~16.1× speedup over FPGA accelerators. Besides, FleetRec lowers the inference latency by 21.0%~92.5% percent compared to CPUs. As a result, FleetRec is an ideal candidate for real-time inference — it outperforms a CPU based system by 41.8~387.2× given a 10 ms latency bound. Besides the three industry models, one can also generalize FleetRec to any recommendation models: the performance interpretability of FleetRec enables to estimate the performance of any model without the need to implement the hardware. With this, the contributions of the paper include:

- We design and implement FleetRec, a high-performance and configurable recommendation engine supporting a wide range of model size scales and architectures. The design is based on our observation of the characteristics of three production models used by Alibaba.
- We implement an efficient dataflow architecture on an FPGA for high-throughput embedding table lookups. We also integrate a 100 Gbps TCP/IP stack into Xilinx's Vitis development platform, enabling FPGAs to serve as smart disaggregated memory for recommendations. We further develop an optimized software infrastructure on the GPU server, allowing a seamless and high-performance coordination between the memory and computation nodes.
- We test FleetRec on three production models. Compared to an optimized CPU baseline, FleetRec shows more than one order of magnitude speedup in terms of throughput while significantly lowering latency — a significant advantage in real-time recommendations.

2 BACKGROUND & MOTIVATION

We describe a typical deep recommendation model and point out the challenges to design a high-performance inference system for it, namely the embedding-vector-lookup bottleneck, the latency constraints, and the model size scale. Then existing solutions and their shortcomings are discussed.

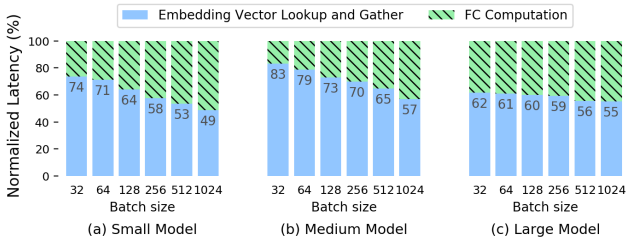

Figure 2: Latency breakdown of three production recommendation models ranging from 1 GB to over 100 GB.

2.1 Representative Recommendation Models

Figure 1 outlines a representative deep recommendation model deployed in Alibaba. It is responsible for predicting *click-through-rates* (CTR), i.e., how likely it is that the user will click on a given product. The model takes a set of sparse and dense features as input. For example, account IDs and region information are encoded as one-hot vector (sparse feature), while age is among the dense features. The prediction process is as follows. First, the sparse features are converted to a set of indexes to lookup vectors on a set of embedding tables. For each inference task, one or several vectors are retrieved from each table [5]. The embedding vectors are then concatenated with the dense features. Finally, the concatenated vectors are used as input to the top fully-connected (FC) layers for CTR prediction to determine the products with the highest CTRs.

This neural network architecture is representative of models used in industry. Although concrete designs vary, most of them are built around the embedding table lookup and the DNN computation, thus sharing the same performance challenges (Section 2.2) observed in the Alibaba use case. For example, Facebook introduces additional fully-connected layers to process dense input features: the dense features are fed into the bottom FC layers, and the output features are concatenated with embedding vectors [5]. Google adds an linear model alongside the deep model [1].

2.2 Inference Challenges

Challenge 1: embedding table lookup. The lookup of the embedding tables is a unique bottleneck in recommendation inference compared to regular DNN workloads. Figure 2 shows the cost of embedding layers during inference on three production models ranging from 1 GB to over 100 GB.

These lookup operations cause performance issues for two reasons. First, looking up embedding vectors on many tables causes random DRAM accesses, leading to significantly under-utilized memory bandwidth and low lookup performance. Figure 3 shows a toy example of looking up two embedding tables. The pattern of looking up vectors from different tables is unfriendly to the underlying hardware because jumping around virtual memory space requires the row buffer of the DRAM bank to be charged and discharges repetitively. Since each embedding vector is short (usually containing between 4 and 64 elements), the DRAM bandwidth utilization is very low, as shown in the lower half of Figure 3. Second, embedding lookups result in operator-call overhead if one resorts to machine learning frameworks such as TensorFlow and PyTorch.

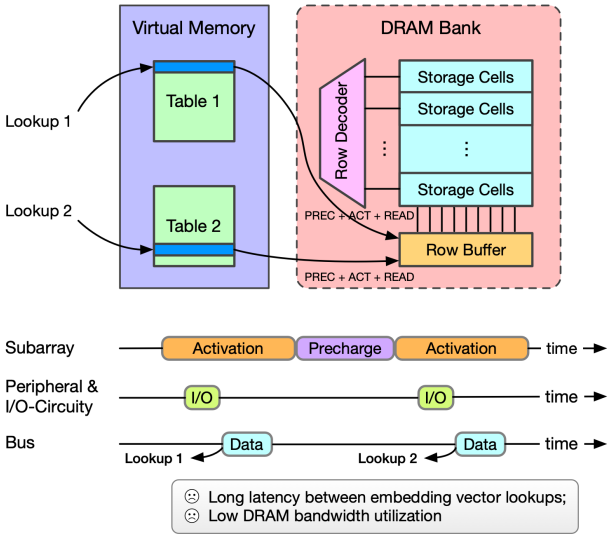

Figure 3: Embedding table lookup operations lead to under-utilized DRAM bandwidth, causing the memory bottleneck.

According to our observations on *TensorFlow Serving* which is optimized for inference, the embedding layer involves 37 types of operators (e.g., concatenation and slice) and these operators are invoked multiple times during inference, resulting in a large overhead especially for small batches. Unfortunately, small batch sizes are usually required in CPU-based recommendation engines to meet the latency requirements of tens of milliseconds.

Challenge 2: serving models over 100 GBs. Embedding tables usually contribute to the majority the memory requirements in recommendation systems. At industrial scale, they can contain up to hundreds of millions of entries, consuming tens or even hundreds of gigabytes of memory. For example, the largest model in our experiments contains 377 embedding tables and requires 114 GB of memory. The single largest table contains 2 million entries of 64-dimensional encoded vectors: over 50 GB for a single table. Such sizes pose a challenge for specialized hardware. For example, the DRAM capacity of GPUs and FPGAs is around a few tens of GB, thus unable to serve large recommendation models.

Challenge 3: optimizing throughput under SLA constraints. Large batch sizes usually lead to better throughput for CPUs and GPUs because of the better utilization of the SIMD architecture and the amortization of function call overheads. In real-time recommendation systems, the *Performance Metric* is throughput under SLA constraints, limiting the batch sizes usable in practice.

2.3 Existing Approaches & Limitations

Although several solutions have been proposed to serve recommendation models, they all fail to meet some of the challenges just described. This section introduces existing solutions, summarises their pros and cons, and points to the need for a novel system capable of meeting all outstanding challenges.

CPU-based. CPU-based recommendation inference is the go-to choice in industry [4, 5], because (a) deployment on CPU servers requires no extra investment on novel hardware; (b) the DRAM

Table 1: FleetRec compared with existing solutions.

Solution	Embedding Lookups	DNN Computation	Supported Model Size	Throughput under SLA	Inference Latency
CPU [5, 8, 14]	Slow	Slow	Large	Low	Medium
GPU [4, 8, 9]	Slow	Fast	Medium~Large	Low~Medium	Medium~High
CPU-FPGA [9]	Slow	Medium	Large	Low~Medium	Very Low
FPGA with HBM [10]	Fast	Medium	Medium	Medium	Very Low
FleetRec (Ours)	Fast	Fast	Large and Scalable	High	Low

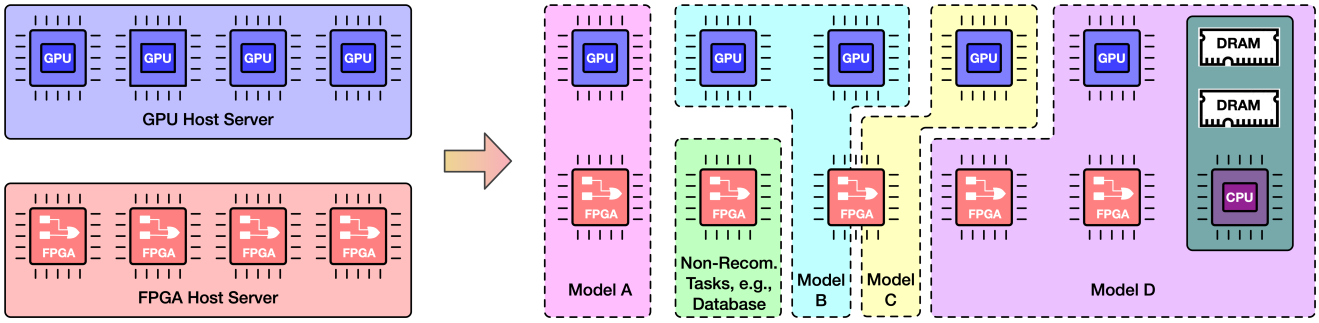

Figure 4: System overview of FleetRec. It is built upon heterogeneous computing clusters consists of CPUs, GPUs, and FPGAs without modifying the server setups. FPGAs and CPU servers are embedding table lookups engines while GPUs provide DNN computation firepower. These hardware resources can be bridged flexibly by a high-speed network to adapt various recommendation workloads (model A~D) while leaving the remaining hardware resources for non-recommendation tasks.

capacity installed on CPU servers is typically enough to serve large recommendation models; and (c) inference latency on small batches is generally lower than in GPUs. Though widely deployed, CPUs are not known for their DNN inference performance compared to those of GPUs, and the memory bottleneck caused by the embedding lookups worsens the situation.

GPU-based. Two deployments have been explored on GPU. Leaving both the embedding lookup and the DNN computation to the GPU is one option [4, 8], but it cannot serve large models because of the limited GPU DRAM capacity. Besides, although GPUs use high memory bandwidth (HBM), their SIMD architecture is not suitable for irregular operations such as individual table lookups, losing the bandwidth advantage of HBM. Another option is to do the embedding lookups on the CPU, and then transfer the concatenated vector to the GPU through the PCIe bus [9]. In general, the speedup of GPUs improves the DNN computation but the memory bottleneck remains. In addition, GPUs can deliver high throughput for large batches but do less than optimally for the small batches needed to meet the latency constraints.

FPGA-based. FPGAs can be viewed as application-specific integrated circuits once they are programmed, thus are suitable for latency-sensitive applications such as recommendation. Hwang et al. [9] is an FPGA solution in which the FPGA accesses the CPU-side DRAM for embedding lookups and performs the DNN inference on the FPGA. This solution provides enough memory capacity but still suffers from the embedding bottleneck and the speedup is mainly obtained from the fast DNN computation. Jiang et al. [10] provides an alternative solution taking advantage of the High-Bandwidth Memory (HBM) available on the latest FPGA models, thus providing the highest embedding lookup performance among existing solutions. However, it is restricted to small model sizes since the available memory capacity is only 8 GB of HBM plus 32 GB of DDR4 DRAM.

Our Goal. As summarized in Table 1, all existing solutions have their limitations, thus are only suitable for a subset of inference scenarios. In this paper, we aim to obtain a single solution that can (1) minimizes the memory bottleneck caused by embedding table lookups; (2) achieves high end-to-end inference throughput under SLA constraints; and (3) supports recommendation models larger than 100 GB.

3 FLEETREC

Key Advantages. We introduce FleetRec, a high-performance and configurable heterogeneous computing cluster for recommendation inference. FleetRec provides several advantages over current solutions. First, it combines the strengths of heterogeneous hardware (FPGAs and GPUs) while avoiding the weaknesses of each platform. Second, it scales out to large models by simply plugging in more memory nodes (FPGAs and CPU servers). Third, by configuring the ratio of the two types of nodes (memory and computation), the cluster is adaptable to various workloads, regardles of whether the models are computation-intensive or table-lookup-intensive. In the following, we present the key insights of FleetRec's design abstracting away the low level details of the hardware implementation.

3.1 System Overview

System Components. Figure 4 shows the architecture of FleetRec, built on a cluster of heterogeneous hardware. The embedding vector lookups are performed on FPGAs equipped with high-bandwidth memory and CPU servers with sufficient DRAM, while the DNN

computation happens in the host servers equipping GPUs. FleetRec regards each accelerator as an individual end device connected by 100 Gbps TCP/IP network. FleetRec can be adapted to a wide range of workloads by using different configurations that vary the number and interconnection topology between CPUs, GPUs, and FPGAs (Figure 4).

Recommendation Query Processing Flow. The inference starts by retrieving the embedding vectors (the sparse features) through lookup indexes on the embedding tables residing in the memory nodes (FPGAs and, when needed, CPU servers). Each memory node completes the table lookup and concatenates the retrieved vectors before sending them to the GPU server. The GPU server concatenates all the received embedding vectors with dense features and runs them by the DNN in a batch.

The key design philosophy of FleetRec is two-fold:

First, FleetRec takes advantage of the strengths of different types of hardware. The latest FPGA equipped with HBM enables highly concurrent embedding table lookups (a few tens of lookups in parallel), yet it has limited memory capacity (8 GB HBM + 32 GB DDR) and insufficient DNN computation performance. According to our experiments implementing FPGA accelerators for recommendation inference, the DNN computation module is one order of magnitude slower than the HBM-fueled embedding lookup module [10]. GPUs are great for pure computation but suffer from irregular memory access patterns such as the embedding table lookups. FleetRec combines the best of both worlds: FPGAs implement the embedding table lookups while GPUs run the DNN computation. FleetRec can also include CPU servers as memory nodes to provide sufficient DRAM capacity for a few large tables: the largest embedding table in our models is more than 50 GB. Keeping such tables in a CPU server is a more efficient choice than FPGAs.

Second, FleetRec disaggregates computation and memory, leading to high scalability and flexibility. Instead of plugging a certain number of FPGAs and GPUs to the same host server, FleetRec treats these accelerators as individual end devices connected by a network, enabling flexible combinations between computation and memory resources. To scale out and support large recommendation models, more FPGAs or CPU nodes can be added to the cluster. To adapt to different models, the resources allocated to computation or embedding lookups can be independently adjusted to balance performance between the two components. For DNN-computation-intensive model architectures, installing more GPUs on the computation node matters more than having multiple memory nodes. For models with hundreds of embedding tables, pairing several FPGA nodes with one GPU can be a more reasonable choice.

3.2 The FPGA as Smart Disaggregated Memory

We use the latest FPGAs equipped with hybrid memory system as smart disaggregated memory to enable highly parallel embedding lookups. The Xilinx Alveo U280 FPGA cards used in the experiments contain three types of memory: high-bandwidth memory (HBM), DDR4 DRAM, and on-chip memory (BRAM and URAM). The HBM system on the U280 offers improved concurrency (32 independent memory banks) and bandwidth (up to 425 GB/s) compared to conventional DRAMs [11, 15, 16]. Thus, embedding tables can be distributed across these banks so that each bank only contains one or a few tables, and up to 32 tables can be looked up in parallel[1]. The 2 DDR4 DRAM channels on the board provide higher memory volume (32 GB DDR vs 8 GB HBM), although it only supports 2 parallel lookup operations at a time. Besides DRAM (HBM and DDR) for which the memory access latency is around a couple of hundreds of nanoseconds, the U280 card also contains tens of MB of on-chip SRAM, with low-latency access comparable to that of a CPU cache.

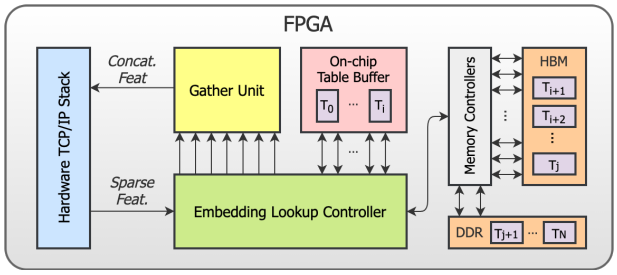

Figure 5: The hardware design of a FleetRec FPGA.

To maximize the embedding lookup performance, a FleetRec FPGA allocates the tables to its hybrid memory system in the following manner. There are no embedding data exchange between the memory hierarchy because the model is known at the system development stage. First, it stores as much small tables as possible in SRAM since that allows low-latency and high-concurrency vector retrieval. Second, it distributes the rest tables in HBM and DDR banks. The largest tables are stored in DDR banks because of its higher capacity (16 GB per DDR bank compared to 256 MB of an HBM bank). Because the random access latency to HBM and DDR are close (200~300 ns), FleetRec FPGA ties the number of embedding tables stored in each bank to balance the workload. During the embedding lookup process, vectors stored in different banks can be read in parallel and the performance is decided by the rounds of DRAM access. For example, a model contains 90 tables: 30 of them are stored on-chip while the rest 60 are evenly distributed to HBM and DDR banks (30 banks available in total). Then the FPGA will concurrently gather all on-chip vectors and issue 2 rounds of parallel DRAM accesses to retrieve all embedding vectors.

Figure 5 illustrates the hardware design of an FPGA node in FleetRec: it takes sparse feature (lookup indexes) as input and outputs the concatenated embedding vectors through the network to the computation node. FleetRec uses an open-source 100 Gbps network stack [7] integrated into the Vitis FPGA development platform [12], so that our integrated network stack supports the Xilinx U280 FPGA cards as well as High-Level Synthesis, an FPGA development flow allowing programming hardware in C/C++. We then implement the components enabling the FPGA to serve as smart disaggregated memory for recommendation, including a table lookup controller to handle memory accesses and a gather unit to concatenate all the retrieved vectors.

[1]We use most (28 of 32) HBM banks to hold embedding table in the experiments. Another 2 HBM channels serve as network cache, while the rest 2 channels are not used because they overlap with the PCIe region of the card and using them can lead to routing issues and degraded performance [2].

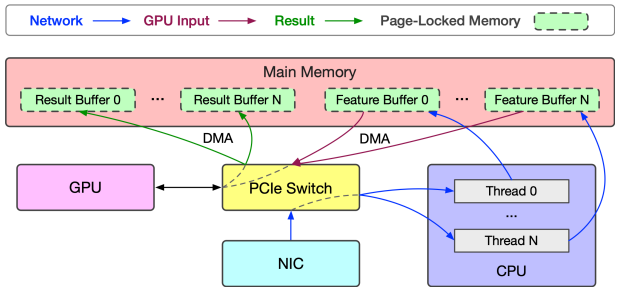

Figure 6: The GPU server receives concatenated feature from memory nodes over network and finishes inference.

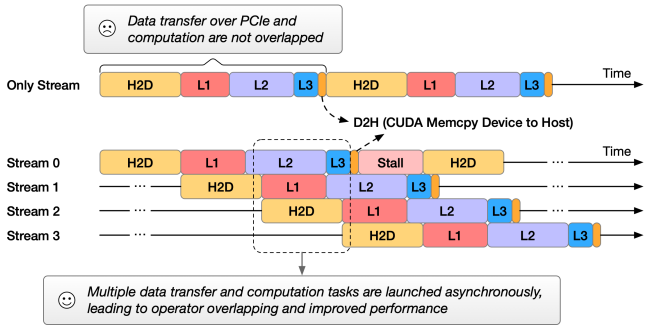

Figure 7: Maximize the GPU performance by overlapping data transfer and computation using multiple streams.

3.3 The GPU as DNN Engine

Figure 6 shows the working flow on the FleetRec GPU server. The inference starts by receiving the concatenated input features sent by the memory nodes. The batched input features are stored into page-locked memory in the CPU side DRAM (GPUs do not have direct network access). The GPU reads a batch of input features, runs the DNN computation, and returns the predicted CTR. We optimize performance to maximize throughput as follows:

GPU operator scheduling. To maximize GPU utilization, we use multiple *CUDA streams* for inference. As shown in the lower half of Figure 7, employing multiple streams (a) enables operator-level concurrency and (b) overlaps computation (3 layers of DNN in the form of general matrix-matrix multiplication) and communication between host and device (H2D and D2H). In this case, the GPU throughput is maximized with a marginal latency overhead.

From network to main memory. We use multiple threads on the CPU side for network packet processing, memory management, and issuing tasks to the GPU. As discussed above, multiple CUDA streams are launched to maximize inference throughput and we utilize individual threads on the host CPU to handle each task stream. The jobs of a thread include: (a) establishing TCP/IP connections to memory nodes, (b) receiving the network packets and storing the input feature into main memory, and (c) issuing the GPU commands including data transfer and computation. Page-locked memory serves as input feature buffer in main memory, so that direct memory access (DMA) between the CPU main memory and the GPU is possible (with DMA, a GPU can access the CPU side memory without involving CPU, thereby increasing throughput and reducing latency). After each batch of DNN computations, the GPU writes the predicted CTRs to the host memory.

Table 2: Specification of the three production models.

Scale	Table Num	Feature Len	Hidden-Layer	Size
Small	47	352	(1024, 512, 256)	1.3 GB
Medium	98	876	(1024, 512, 256)	15.1 GB
Large	377	3968	(2048, 512, 256)	114.4 GB

4 EVALUATION

Results Overview. We first evaluate FleetRec on three production models from Alibaba. FleetRec shows significant throughput improvement over both the CPU and FPGA baselines while also significantly reducing latency. Due to the improved throughput and reduced latency, FleetRec is especially good at real-time inference under strict SLA constraints — it achieves two orders of magnitude speedup over the CPU based system given a 10 ms SLA. We then show how to generalize and configure FleetRec for other recommendation models by estimating the speedup of FleetRec and balancing computation and lookup performance to maximize performance while minimizing hardware usage.

4.1 Model Specification

We experiment with three deep recommendation models of different sizes from Alibaba. All three models involve heavy embedding vector lookups, thus showing different workload characteristics compared with non-recommendation DNN architectures. Table 2 shows the parameters of the models. These models contain 47, 98, and 377 embedding tables respectively, and each table is looked up once during inference. For example, during each inference, the largest model gathers the embedding vectors retrieved from 377 tables into a 3968-dimensional dense vector, and feeds it to three fully-connected layers. Though not shown in the table, the single largest embedding table contains 200M entries consuming 51.2 GB memory footprint.

4.2 Experimental Setup

CPU baseline. We run the models on two types of CPU servers on Amazon's AWS. The small and medium models are tested on a server with Intel Xeon E5-2686 v4 CPU @2.30GHz (16 vCPU, Broadwell, SIMD operations, i.e., AVX2 FMA, supported) and 128 GB DRAM (8 channels). A more powerful server with Intel Xeon Platinum 8259CL CPU @ 2.50GHz (32 vCPU, Cascade Lake, SIMD supported) and 256 GB DRAM is used for the larger model. For the ML framework, we use *TensorFlow Serving* which is optimized for model inference.

FPGA baseline. Besides the CPU baseline, we also compare FleetRec with a set of FPGA recommendation accelerators (single FPGA implementation) optimized for each individual model. The FPGA accelerator is responsible for not only the embedding lookups but also the DNN computation. The computation module is implemented by constructing several general matrix-matrix multiplication (GEMM) blocks (each standing for one layer) connected by FIFOs. Each GEMM block is further composed by a set of processing elements

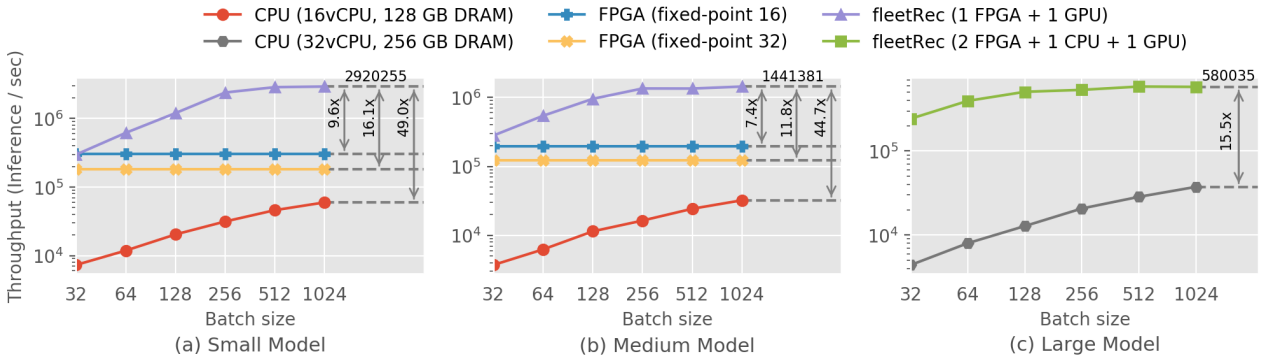

Figure 8: FleetRec significantly outperforms the CPU and FPGA baselines in terms of throughput.

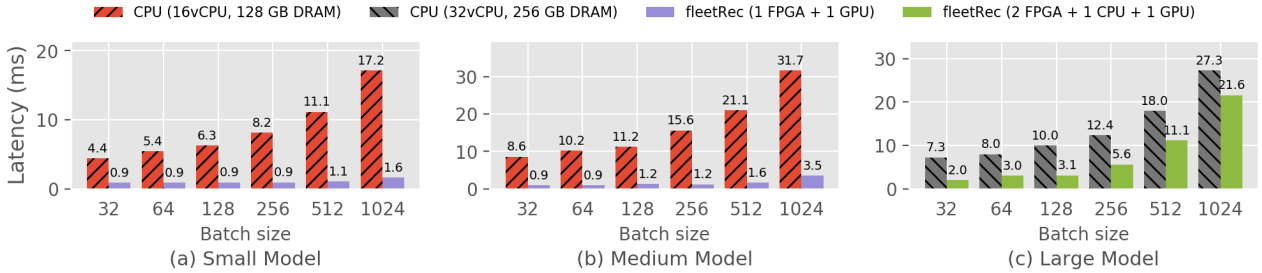

Figure 9: FleetRec achieves much lower latency compared to CPU engines given the same batch sizes.

(PEs) which is the basic unit to perform parallelized vector multiplications. Such modularized design maximizes hardware resource usage and avoids the performance degradation caused by placement and routing issues [3]. Since FPGAs are naturally good at fixed-point computation rather than floating point, we perform quantization on the models and test the performance under two level of precision, i.e., 16-bit and 32-bit fixed-point numbers. Both the CPU baseline and FleetRec are tested with 32-bit floating point. Due to the memory capacity limitations in the FPGA, the FPGA accelerator experiments only include the small and medium models.

FleetRec setups. Similar to the CPU baseline, we use two FleetRec configurations. One FPGA plus a GPU are enough to serve the small and medium models. The larger models contains hundreds of tables consuming 114 GB memory, thus we upgrade the disaggregated memory to 2 FPGAs and 1 CPU server. We use Xilinx Alveo U280 FPGAs equipped with 8GB of HBM2 DRAM (32 channels) and 32 GB of DDR4 DRAM (2 channels). We implement the FPGA hardware logic using Vitis HLS and set the clock frequency to 180 MHz. For the computation node, we use a Titan RTX GPU containing 4608 CUDA cores. The DNN computation flow is constructed by the cuBLAS library. We test 1~16 CUDA streams for all three models to maximize the GPU performance and presents the results with the highest throughput. The GPU server uses a Mellanox ConnectX-5 NIC with a 100 Gbps Ethernet connection. The FPGAs use an open-source 100 Gbps TCP/IP network kernel [7].

4.3 End-to-End Inference Performance

FleetRec shows more than one order of magnitude speedup in terms of inference throughput over the CPU baseline. In Figure 8, we compare the throughput of CPUs, FPGAs, and FleetRec using batch sizes ranging from 32 to 1024 (larger batch sizes can violate the SLA of 10 ms). The throughput of the CPU-baseline and FleetRec increases with the batch size, while the throughput of the FPGA accelerators remains constant because the dataflow architecture used in the FPGA processes inference item by item instead of in batches. According to the peak throughput on the three models, FleetRec achieves 15.5~49.0× speedup over the CPU baseline, and 7.4~16.1× over the FPGAs. The significant speedup over FPGAs justifies introducing heterogeneous hardware in the form of a GPU in addition to the FPGAs. FleetRec exhibits even higher speedups over CPUs due to both the fast DNN computation enabled by the GPU and the highly concurrent embedding lookups provided by FPGAs equipped with HBM. The speedups over CPUs on small and medium models are more significant than in the large one, because we use a more powerful CPU server (with twice as many cores and memory channels as well as the latest micro-architecture design) for large model inference, while only one GPU is deployed in FleetRec across the three experiments.

FleetRec exhibits much lower latency compared to CPUs. As shown in Figure 9, the latency reduction achieved by Fleet given the same batch sizes ranges from 21.0 % to 92.5% with an average of 76.6%. Note that FleetRec can perform single-millisecond-level inference on small and medium models using medium batch sizes (e.g., 256), while the CPU needs around 10 ms. Using a medium batch size of 256 already allows FleetRec to achieve close-to-peak throughput (around 85% of the peak performance), while for the CPU only achieves around half of the throughput reached with batch sizes of 1024. Though still better than the CPU baseline, the latency of FleetRec on the large model is higher compared with the medium and small ones, because (a) the network traffic to transfer a batch of inputs of the large model is far heavier due to the feature length as shown in Table 2; and (b) each thread on the GPU server

Table 3: Throughput under strict SLA requirements. FleetRec shows huge advantage over CPU for real-time inference.

	Small Model			Medium Model			Large Model		
SLA (ms)	5	10	20	5	10	20	5	10	20
Throughput (inferences / sec)									
CPU	7.30E+3	3.14E+4	5.96E+4	N/A	3.72E+3	1.64E+4	N/A	1.28E+4	2.85E+4
FPGA	3.05E+5	3.05E+5	3.05E+5	1.95E+5	1.95E+5	1.95E+5	N/A	N/A	N/A
FleetRec	2.92E+6	2.92E+6	2.92E+6	1.44E+6	1.44E+6	1.44E+6	5.07E+5	5.35E+5	5.80E+5
Speedup of FleetRec over									
FPGA	9.57×	9.57×	9.57×	7.39×	7.39×	7.39×	+∞×	+∞×	+∞×
CPU	400.07×	92.97×	48.96×	+∞×	387.24×	87.92×	+∞×	41.76×	20.34×

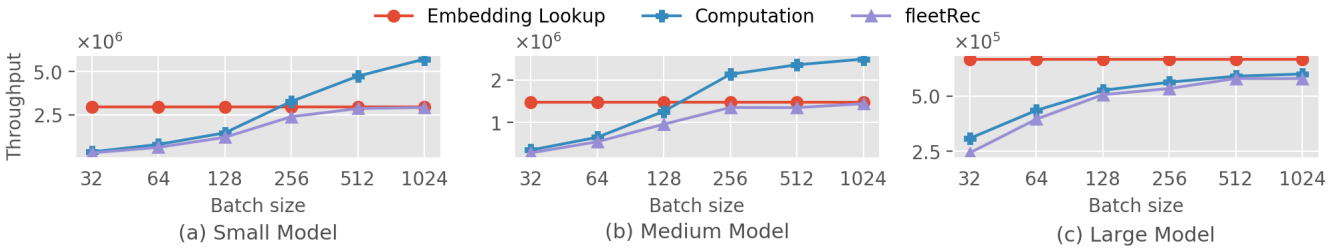

Figure 10: The performance of FleetRec can be estimated by taking the minimum of the lookup and computation module.

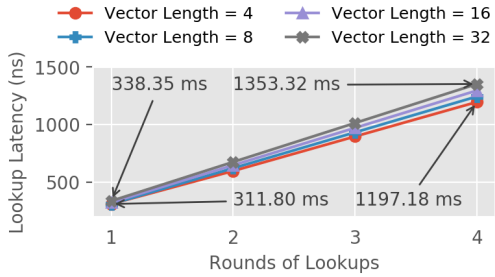

Figure 11: The embedding lookup performance can be estimated given the DRAM access rounds.

needs to maintain individual connections with several memory nodes and receive data from them in a round-robin manner.

For real-time recommendations under strict SLA (latency) constraints, the speedup of FleetRec is even more significant. Achieving high throughput and guaranteeing low latency can be contradictory on CPU-based inference engines, since throughput is increased by employing large batches. Table 3 presents the throughput of several systems under different SLAs by translating the data presented in Figure 8 and 9. Given a latency constraint of 10 ms, FleetRec can achieve 41.8~387.2× speedup in throughput, even more significant than the 15.5~49.0× achieved when not considering the latency constraint.

4.4 Generalizing and Configuring the System

FleetRec can be generalized and configured beyond the three industry models: given any recommendation models, one can (a) decide how to configure the system (number of GPUs, FPGAs, and CPU memory nodes); and (b) estimate FleetRec's performance, without the need to implement the hardware at all. To prove this, we first show that the performance of FleetRec can be estimated once the performance of the computation and embedding lookup modules is known. We then present how to estimate the performance of the two components without actually implementing them.

The performance of FleetRec can be estimated as shown in Figure 10. Once the performance of the computation and embedding lookup modules is known, we can estimate the throughput of FleetRec under different batch settings by taking the minimum throughput of the two components. There is a tolerable performance gap between FleetRec and the lower performance bound of the two components, because FleetRec involves network while the performance of the computation and embedding lookup modules is tested without network.

The performance of both computation and embedding lookup modules can be estimated easily without the need to implement hardware. On the GPU side, one can resort to existing ML programming frameworks, e.g., TensorFlow, PyTorch, or MXNex, and remove the embedding layer to test only the DNN computation performance on a GPU. On the FPGA side, the lookup performance is decided by DRAM (HBM and DDR) access rounds and influenced by embedding vector lengths. As shown in Figure 11, given the same embedding vector length, the lookup latency is proportional to the rounds of DRAM access. The lookup latency of different vector lengths are very close (within 40 ns per round): the embedding vectors are short and cannot fully take advantage of the spatial locality within the DRAM, thus these accesses are almost random, and the FPGA only needs to pay a few more clock cycles to read a longer vector. Note that the embedding lookup is issued item by item no matter what the batch size is, thus the throughput is simply the reciprocal of the latency shown in Figure 11. This predictability allows us to estimate the embedding lookup performance given a recommendation model without actually implementing it. For example, given a model with 90 tables and 30 of them small enough to be stored on-chip, we can allocate the rest 60 tables to DRAM (28

available HBM channels plus 2 DDR channels), and the FPGA can finish the lookup process with 2 rounds of DRAM access as long as the model size is within the capacity of the DRAM. This estimation also works for multi-FPGA lookup modules. We first estimate the performance of each individual nodes, and the lookup performance is bound by the one with the lowest throughput.

Once the performance of computation and embedding lookups is known, one can configure FleetRec in a way that maximizes performance while minimizing resource usage by balancing the performance of the two components. For example, one can couple an FPGA node with several GPUs for computation-intensive models. On the contrary, the design employing multiple FPGAs and a single GPU could fit models with many embedding tables and a set of light-weight DNN layers.

5 RELATED WORK

Hardware accelerators for recommendation inference. According to Facebook, recommendation workloads can consume up to 79% of total AI inference cycles in data centers [5], thus the research community has started to explore hardware-accelerated solutions. Kwon et al. [14] proposed to reduce the memory bottleneck by redeisgning the DRAM micro-architecture to support higher lookup concurrency. Ke et al. [13] extended this idea of near-memory-processing by adding memory-side-caching for frequently-accessed entries. Gupta et al. [4] developed a recommendation query scheduling policy to dispatch the workload to CPUs or GPUs given the real-time query arrival patterns. Hwang et al. [9] implemented an FPGA accelerator for deep recommendation inference. Jiang et al. [10] designed another FPGA acceletor by taking advantage of the HBM system recently available on FPGA, and introducing data structure solution, i.e., Cartesian products, to reduce the number of DRAM accesses. Zhu et al. [19] extended this idea by using an FPGA cluster to conquer the computation bottleneck, but it requires more accelerators to achieve the same performance compared with our work. FleetRec shows clear advantages over these solutions as summarized in Table 1.

6 FUTURE PERSPECTIVES

To popularize FleetRec in a wide range of deployments, we expect an end-to-end development flow from ML frameworks to hardware to be very useful. Though showing attractive speedups, FleetRec involves manually optimized hardware design for each individual recommendation models. This requires seamless collaboration between an ML team and a hardware team. Fortunately, the hardware logic of embedding table lookups follows a rather fixed pattern, thus it is possible to prepare a set of hardware code templates, and compile the look up logic to hardware using an automated code generator. Once this can be achieved, we can integrate FleetRec to existing ML frameworks such as TensorFlow and PyTorch, allowing an end-to-end development experience for ML engineers, who can then focus on the DNN architecture design and can deploy this high-performance system with a single button.

ACKNOWLEDGMENTS

Part of the work of Wenqi Jiang and Zhenhao He has been funded by the Alibaba Group. We would like to thank Xilinx for their generous donation of the XACC FPGA cluster at ETH Zurich on which the experiments were conducted.

REFERENCES

[1] 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*.

[2] Young-kyu Choi, Yuze Chi, Jie Wang, Licheng Guo, and Jason Cong. 2020. When HLS Meets FPGA HBM: Benchmarking and Bandwidth Optimization. *arXiv preprint arXiv:2010.06075* (2020).

[3] Johannes de Fine Licht, Grzegorz Kwasniewski, and Torsten Hoefler. 2020. Flexible Communication Avoiding Matrix Multiplication on FPGA with High-Level Synthesis. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 244–254.

[4] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-scale Neural Recommendation Inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*.

[5] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of Facebook's DNN-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[6] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*.

[7] Zhenhao He, Dario Korolija, and Gustavo Alonso. 2021. EasyNet: 100 Gbps Network for HLS. In *2021 31th International Conference on Field Programmable Logic and Applications (FPL)*.

[8] Samuel Hsia, Udit Gupta, Mark Wilkening, Carole-Jean Wu, Gu-Yeon Wei, and David Brooks. 2020. Cross-Stack Workload Characterization of Deep Recommendation Systems. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*.

[9] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: A Chiplet-based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*.

[10] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, et al. 2021. MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions. In *2021 4th Conference on Machine Learning and Systems (MLSys)*.

[11] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.

[12] Vinod Kathail. 2020. Xilinx Vitis Unified Software Platform. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[13] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*.

[14] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[15] Mike O'Connor. 2014. Highlights of the high-bandwidth memory (hbm) standard. In *Memory Forum Workshop*.

[16] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. 2020. Benchmarking High Bandwidth Memory on FPGAs. In *The 29th IEEE International Symposium On Field-Programmable Custom Computing Machines*.

[17] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*.

[18] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[19] Yu Zhu, Zhenhao He, Wenqi Jiang, Kai Zeng, Jingren Zhou, and Gustavo Alonso. 2021. Distributed Recommendation Inference on FPGA Clusters. In *2021 31th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE.