MICROREC: EFFICIENT RECOMMENDATION INFERENCE BY HARDWARE AND DATA STRUCTURE SOLUTIONS

Wenqi Jiang [1 2] **Zhenhao He** [1] **Shuai Zhang** [1] **Thomas B. Preußer** [1] **Kai Zeng** [3] **Liang Feng** [3] **Jiansong Zhang** [3] **Tongxuan Liu** [3] **Yong Li** [3] **Jingren Zhou** [3] **Ce Zhang** [1] **Gustavo Alonso** [1]

ABSTRACT

Deep neural networks are widely used in personalized recommendation systems. Unlike regular DNN inference workloads, recommendation inference is memory-bound due to the many random memory accesses needed to lookup the embedding tables. The inference is also heavily constrained in terms of latency because producing a recommendation for a user must be done in about tens of milliseconds. In this paper, we propose MicroRec, a high-performance inference engine for recommendation systems. MicroRec accelerates recommendation inference by (1) redesigning the data structures involved in the embeddings to reduce the number of lookups needed and (2) taking advantage of the availability of High-Bandwidth Memory (HBM) in FPGA accelerators to tackle the latency by enabling parallel lookups. We have implemented the resulting design on an FPGA board including the embedding lookup step as well as the complete inference process. Compared to the optimized CPU baseline (16 vCPU, AVX2-enabled), MicroRec achieves 13.8∼14.7× speedup on embedding lookup alone and 2.5∼5.4× speedup for the entire recommendation inference in terms of throughput. As for latency, CPU-based engines needs milliseconds for inferring a recommendation while MicroRec only takes microseconds, a significant advantage in real-time recommendation systems.

1 INTRODUCTION

Personalized recommendations are widely used to improve user experience and increase sales. Nowadays, deep learning has become an essential building block in such systems. For example, Google deploys wide-and-deep models for video and application recommendations (Cheng et al., 2016; Zhao et al., 2019); Facebook uses different kinds of deep models for a range of social media scenarios (Gupta et al., 2020b); and Alibaba combines attention mechanism with DNNs and RNNs for online retail recommendations (Zhou et al., 2018; 2019). Due to the popularity of DNN-based recommendation models, they can comprise as much as 79% of the machine learning inference workloads running in data centers (Gupta et al., 2020b).

Deep Recommendation Models We first briefly introduce deep recommendation models to provide the necessary context to discuss the challenges, our methods, and contributions. Figure 1 illustrates a classical deep recommendation model for *Click-Through Rate* (CTR) prediction (Gupta et al., 2020b; Cheng et al., 2016) and summarize its workload characteristics. An input feature vector consists of dense features (e.g., age and gender) and sparse features (e.g., location and advertisement category). Over the dense feature vector, some systems apply a neural feature extractor that consists of multiple fully connected (FC) layers (Gupta et al., 2020b; Kwon et al., 2019), while some design (Cheng et al., 2016) does not contain the bottom FC layers. Over the sparse feature vector, the system translates each feature into a dense feature embedding by looking up it in an embedding table. These features are then combined (e.g., concatenated) and fed to a neural classification model consisting of multiple fully connected layers.

Challenges in a CPU-based System When deploying recommendation systems on typical CPU servers (left half of Figure 2), embedding tables are stored in DDR DRAM, and the cores are responsible for the computation. There are two system bottlenecks in such deployments.

First, embedding table lookups are costly because they induce massive random DRAM accesses on CPU servers. Production recommendation models usually consist of at least tens of embedding tables, thus each inference requires the corresponding lookup operations. Due to the tiny size of each embedding vector, the resulting DRAM accesses are nearly random rather than sequential. Since CPU servers have only a few memory channels, these random DRAM accesses are expensive.

Second, both embedding lookups and computation can be expensive if one resorts to ML frameworks such as TensorFlow

[1]ETH Zurich [2]Columbia University [3]Alibaba Group. Correspondence to: Wenqi Jiang <wenqi.jiang@inf.ethz.ch>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021.

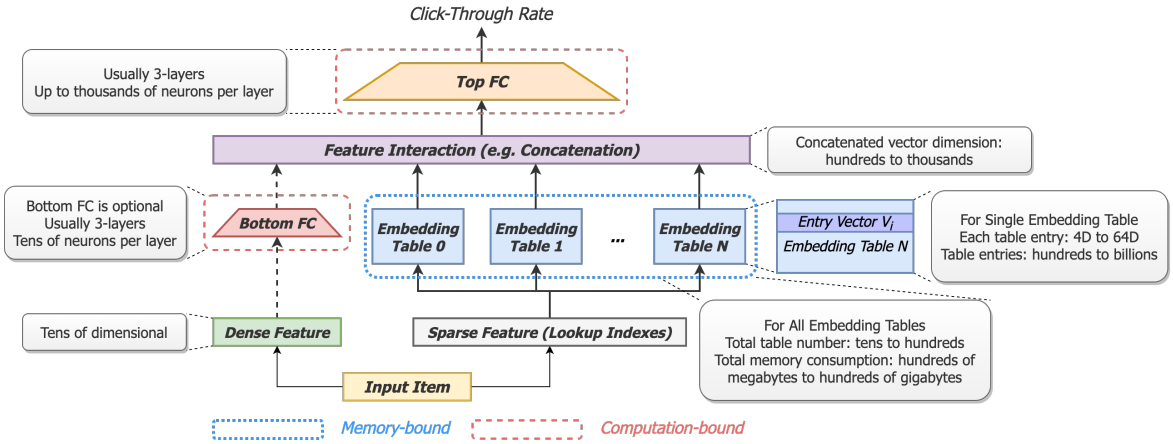

Figure 1. A typical deep recommendation model and it's workload specification.

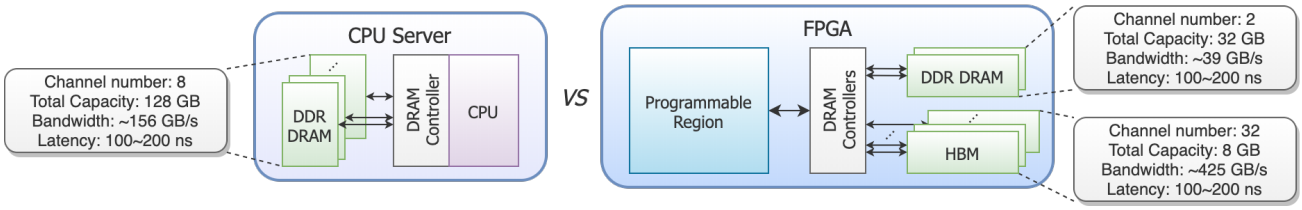

Figure 2. Two hardware choices for recommendation inference. Left: a typical CPU server on which models are stored in DDR DRAM (memory channel number varies from server to server) and computation is done in CPU. Right: our FPGA accelerator where embedding tables are distributed over many memory channels and fast inference is supported by reprogrammable circuit.

and PyTorch. 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 significant time consumption especially in small batches. Similarly, the throughput of neural network computation can also be restricted when using small batches. Unfortunately, small batch sizes are usually required in CPU-based recommendation engines to meet the latency requirements of tens of milliseconds, thus the framework overhead is non-negligible.

Not surprisingly, there has been a range of work trying to accelerate deep recommendation models. Kwon et al. (2019) and Gupta et al. (2020b) observed the main system bottleneck of substantial random memory accesses. Kwon et al. (2019) and Ke et al. (2020) thus proposed to redesign DRAM in micro-architectural level; however, it would take years to put such new DRAM chips in production even if they are adopted. Gupta et al. (2020a) suggested GPUs could be useful in recommendation for large batches, but the memory bottleneck still remains and GPUs suffer from high latency. Similarly, Hwang et al. (2020) implemented an FPGA accelerator for recommendation but without removing the memory bottleneck. In this paper, we ask: *Can we accelerate deep recommendation models, at industrial scale, with practical yet efficient hardware acceleration?*

Our Approach Based on careful analysis of two production-scale models from Alibaba, we design and implement MicroRec, a low-latency and high-throughput recommendation inference engine. Our speed-ups are rooted in two sources. First, we employ more suitable hardware architecture for recommendation with (a) hybrid memory system containing High Bandwidth Memory (HBM), an emerging DRAM technology, for highly concurrent embedding lookups; and (b) deeply pipelined dataflow on FPGA for low-latency neural network inference. Second, we revisit the data structures used for embedding tables to reduce the number of memory accesses. By applying Cartesian products to combine some of the tables, the number of DRAM accesses required to finish the lookups are significantly reduced .

Our contributions in this paper include:

1. We show how to use high-bandwidth memory to scale up the concurrency of embedding lookups. This introduces 8.2∼11.1× speedup over the CPU baseline.

2. To the best of our knowledge, this is the first paper that proposes to reduce the number of random memory accesses in deep recommendation systems by data structure design. We show that applying Cartesian Products between embedding tables further improves the lookup performance by 1.39∼1.69× with marginal storage overhead (1.9∼3.2%).

3. To optimize performance with low storage overhead, we propose a heuristic algorithm to combine and allocate tables to the hybrid memory system on the FPGA.

4. We implement MicroRec on FPGA and test it on two production models from Alibaba (47 tables, 1.3 GB; 98 tables, 15.1 GB). The end-to-end latency for a single inference only consumes 16.3∼31.0 microseconds, 3 to 4 orders of magnitude lower than common latency requirements for recommender systems. In terms of throughput, MicroRec achieves 13.8∼14.7× speedup on the embedding layer, and 2.5∼5.4× speedup on the complete inference process compared to the baseline (16 vCPU; 128 GB DRAM with 8 channels; AVX2-enabled).

2 Deep Recommendation Systems

Personalized recommendation systems are widely deployed by YouTube (Covington et al., 2016; Zhao et al., 2019), Netflix (Gomez-Uribe & Hunt, 2015), Facebook (Park et al., 2018), Alibaba (Zhou et al., 2018; 2019), and a number of other companies (Underwood, 2019; Xie et al., 2018; Chui et al., 2018). In this section, we review their basic properties and analyze their performance to identify the main bottlenecks.

2.1 Deep Model for Ranking

Figure 1 abstracts the deep model for recommendation ranking that we target: it is responsible for predicting *click-through-rates* (CTR), i.e., how likely it is that the user will click on the 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 serves as part of the dense feature since the number is consecutive. The prediction process is as follows. First, dense and sparse input features are processed separately. Depending on the model design, dense features can be processed by a few fully-connected layers (Naumov et al., 2019) or served *as-is* without any pre-processing (Cheng et al., 2016; Zhou et al., 2018). The sparse features, on the other hand, are converted to a set of indexes to lookup vectors from embedding tables. For each inference task, one or several vectors are retrieved from each table (Gupta et al., 2020b). The embedding vectors so retrieved are then concatenated with raw or processed dense features. Finally, the concatenated vectors are fed to the top fully-connected layers for CTR prediction. Product candidates with the highest CTRs are recommended to users.

The specific model design varies from scenario to scenario. Some adjustable parameters include: number of fully-connected layers, number of hidden neurons in each layer, numbers and sizes of embedding tables, feature interaction operations (e.g., concatenation, weighted sum, and element-wise multiplication), whether to include bottom fully-connected (FC) layers.[1]

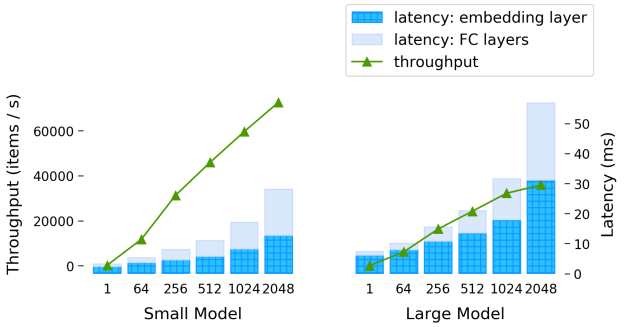

Figure 3. The embedding layer is expensive during inference.

2.2 Embedding Table Lookups

Embedding table lookup is the key difference between deep recommendation models and regular DNN workloads, and it shows the following traits. First, the embedding tables contribute to the majority of storage consumption in deep recommendation models. Large embedding tables at industry scale can contain up to hundreds of millions of entries, consuming tens or even hundreds of gigabytes of storage. Second, the size of the tables varies wildly between a few hundred (e.g., countries or "province ID") to hundreds of millions of entries (e.g., "user account ID").

Embedding table lookup is problematic from a performance perspective. Due to the traits mentioned above, most tables are held in main memory, inducing many random memory accesses during inference. Ke et al. (2020) proves this point by showing that high cache miss rates are common in deep recommendation inference.

2.3 Performance Analysis

We chose CPUs as the hardware platform for baseline experiments. Although GPUs are popular for neural network training, they have not shown clear advantages over CPUs for deep recommendation inference. As reported by Gupta et al. (2020a), GPUs can only outperform CPUs when (a) the model is computation-intensive (less embedding lookups), and (b) very large batch sizes are used.

Figure 3 shows the cost of the embedding layer during inference on two models from Alibaba (models specified in Table 1) . As a side effect of the massive number of memory accesses, the many related operators also lead to significant overhead. According to our observation on *TensorFlow Serving*, an optimized ML framework for inference, 37 types of operators are involved in the embedding layer (e.g., slice

[1]The models we target do not contain bottom FCs, and each table is looked up only once.

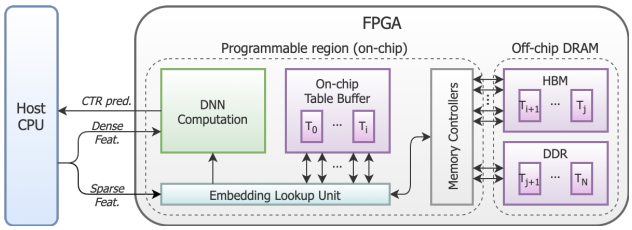

Figure 4. System overview of MicroRec.

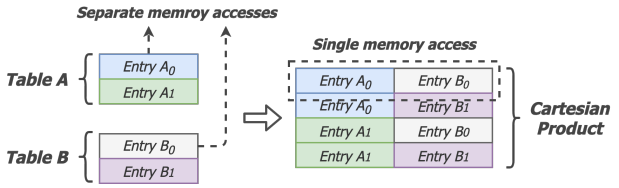

Figure 5. Cartesian product of two embedding tables. Each entry of the product concatenates an entry from table A and another from B: one memory access retrieves two embedding vectors.

and concatenation), and these operators are invoked many times during inference. The close latency to infer small batches (size of 1 and 64) illustrates the expense of operator-calls. Larger batch sizes can lead to better throughput, yet SLA (latency requirement) of tens of milliseconds must be met, thus extremely large batches are not allowed for recommendations.

3 MICROREC

We present MicroRec, an FPGA-enabled high-performance recommendation inference engine which involves both hardware and data structure solutions to reduce the memory bottleneck caused by embedding lookups. On the hardware side, our FPGA accelerator features highly concurrent embedding lookups on a hybrid memory system (HBM, DDR DRAM, and on-chip memory). On the data structure side, we apply Cartesian products to combine tables so as to reduce random memory accesses. Putting them together, we show how to find an efficient strategy to combine tables and allocate them across hybrid memory resources.

3.1 System Overview

Figure 4 overviews the hardware design of MicroRec. Embedding tables are distributed over both on-chip memory (BRAM and URAM) and off-chip memory (HBM and DDR). Neural network inference is taken cared by the DNN computation units which contain both on-chip buffers storing weights of the model and computation resources for fast inference. To conduct inference, the host server first streams dense and sparse features to the FPGA[2]. Then, the embedding lookup unit translates the sparse features to dense vectors by looking up embedding tables from both on-chip and off-chip memory. Finally, the computation unit takes the concatenated dense vector as input and finishes inference before returning the predicted CTR to the host.

3.2 Boost Emebdding Lookup Concurrency by Increased Memory Channels

The tens of embedding table lookup operations during inference can be parallelized when multiple memory channels are available. MicroRec resorts to high-bandwidth memory as the main force supporting highly concurrent embedding lookups. Besides that, we also take advantage of other memory resources on FPGA, i.e., DDR4 DRAM and on-chip memory, to further improve lookup performance.

3.2.1 *High-Bandwidth Memory*

We resort to HBM to parallelize embedding lookups. As an attractive solution for high-performance systems, HBM offers improved concurrency and bandwidth compared to conventional DRAMs (Jun et al., 2017; O'Connor, 2014). In this paper, we use a Xilinx Alveo U280 FPGA card (xil) equipped with 8 GBs of HBM which provides a bandwidth of up to 425 GB/s (Wang et al., 2020). More specifically, the HBM system on U280 consists of 32 memory banks, which can be accessed concurrently by independent pseudo-channels. Thus, embedding tables can be distributed to these banks so that each bank only contains one or a few tables, and up to 32 tables can be looked up concurrently.

3.2.2 *Hybrid Memory System on FPGA*

The Xilinx Alveo U280 FPGA involves multiple types of memory resources, including on-chip memory (BRAM and URAM) and off-chip memory (DDR4 DRAM and HBM), which exhibit different traits. HBM and DDR show close access latency of a couple of hundreds of nanoseconds given the memory controller generated by Vitis (Kathail, 2020), but have different concurrency-capacity trade-off (HBM: 32 channels, 8GB; DRAM: 2 channels, 32 GB). Besides HBM and DDR, FPGAs also equip a few megabytes of on-chip memory that plays a similar role as CPU cache (small yet fast memory to cache frequently-accessed data or intermediate results).Without read initiation overhead as in DRAM, the latency to access on-chip memory only consists of control logic and sequential read. According to our experiments, finish retrieving an embedding vector from an on-chip memory bank only consumes up to around 1/3 time of DDR4 or HBM.

3.3 Reduce Memory Accesses by Cartesian Products

We reduce the number of memory accesses by combining tables so that each memory access can retrieve multiple

[2]The Vitis hardware development platform does not yet support streaming from the host server to a Xilinx U280 FPGA, thus we have prototyped the design by caching the input features on FPGA.

embedding vectors. As shown in Figure 5, two embedding tables can be joined into a single larger one through a relation Cartesian Product. Since tables A and B in Figure 5 have two entries, the product table ends up with four entries: each of them is a longer vector obtained by concatenating an entry from table A and another from table B. Using such a representation, the number of memory accesses is reduced by half: instead of two separate lookup operations now only one access is needed to retrieve the two vectors.

By applying a Cartesian product, the latency to lookup two tables is reduced by almost half. Embedding tables in deep recommendation models usually contain short entry vectors (with between 4 to 64 elements in most cases). Although the entry vectors of the product are longer, i.e., the sum of two individual entries, they are still not long enough to fully take advantage of the spatial locality within DRAM. To retrieve a vector up to a few hundreds of bytes, a DRAM spends most of the time initiating the row buffer, while the following short sequential scan is less significant in terms of time consumption. As a result, reducing the memory accesses by half can lead to a speedup of almost $2\times$.

Though Cartesian products lead to higher storage consumption, this overhead is comparatively small. This may sound counter-intuitive, however, most deep recommendation models contain tables of different size scales, so applying Cartesian products on small tables is almost for free compared to some of the largest tables in the model. According to our observations of real-world deployments, while some tables only consist of 100 4-dimensional embedding vectors, large tables can contain up to hundreds of millions of entries with a vector length of 64 due to the reasons discussed in section 2.2. In this case, a Cartesian product of two small tables requires only tens of kilobytes (assume 32-bit floating-point storage): almost negligible compared to a single large table of tens or hundreds of gigabytes.

Cartesian products can help balancing the workload on off-chip DRAM (DDR and HBM). For example, suppose there are 34 off-chip memory channels (32 for HBM and 2 for DDR), and 40 tables should be allocated on them. In this case, some banks have to store two embedding tables while others only hold one. When retrieving one vector from each table, the lookup performance is bound by the channels holding two tables, as the lookup latency on them is potentially $2\times$ that of those containing only one table. Using Cartesian products, the total number of tables can be reduced from 40 to 34. This allows us to balance the workload on each memory channel resulting in potentially $2\times$ speedup compared to an unbalanced workload situation.

3.4 Putting Everything Together: A Rule-based Algorithm for Table Combination and Allocation

Our objective is to minimize embedding lookup latency given the memory constraints discussed in section 3.2.2, i.e., available capacity and channels of each type of memory. To achieve this, an algorithm is required to explore solutions of combining tables through Cartesian products and deploying the result on memory banks.

3.4.1 Brute-force Search

A straightforward way to achieve this objective is to explore all possibilities in a brute-force manner and choose the best solution. First, one would list all possibilities of using tables as Cartesian product candidates. Then, for each one of these options, all possible combinations of Cartesian products would be calculated (including joining more than two tables). Based on the combinations of tables available, the single and combined tables are allocated to memory banks (solutions exceeding the memory capacity of a bank can be dropped) minimizing the latency. For ties in latency, the solution with the least storage overhead is chosen.

However, applying brute-force search is unrealistic because of the large exploration space. For example, selecting n of out N total tables as Cartesian candidates is a combinatorial problem with a time complexity of $O(\frac{N!}{n!(N-n)!})$. Then, it costs $O(n!)$ to explore any Cartesian products combinations of the candidates. Each outcome, including Cartesian products and original tables, are then allocated to memory banks at the cost of $O(N)$. Using a parameter to control how many tables are selected for Cartesian products, the overall time complexity of the brute-force search is $O(\sum_{n=1}^{N} N\frac{N!}{(N-n)!})$, making brute-force searching infeasible as the number of tables grows up.

3.4.2 Heuristic-rule-based Search

To optimize embedding lookup latency, we propose a heuristic search algorithm that can efficiently search for near-optima solutions with a low time complexity of $\mathcal{O}(N^2)$. Besides, this algorithm can be generalized to any FPGAs, no matter whether they are equipped with HBM, and no matter how many memory channels they have. Due to the memory traits introduced in section 3.2.2, the algorithm simply regards HBM as additional memory channels: designers can adjust the memory channel number and bank capacities in the algorithm according to the available hardware.

Four heuristics are applied in the algorithm to reduce the search space where the optimal solution is unlikely to appear[3]. Consequently, the algorithm can return near-optimas with low time complexity. The first three rules are designed

[3]The rules can be expanded, modified, or removed to adpat different models since these rules are table-size-dependent.

to explore Cartesian combinations efficiently, while the fourth rule is for memory allocation.

Heuristic rule 1: large tables are not considered for Cartesian products. Tables are sorted by size and only the n smallest tables should be selected for Cartesian products, otherwise products of large tables can lead to heavier storage overhead.

Heuristic rule 2: Cartesian products for table pairs of two. Although Cartesian products of the three smallest tables may only consume tens of megabytes storage (still small compared to a single large table of several or tens of gigabytes), the overall solution could be sub-optimal because this method consumes too many small tables at once while they are appropriate candidates to pair with larger tables.

Heuristic rule 3: within the product candidates, the smallest tables are paired with the largest tables for Cartesian products. This rule avoids terrible solutions where a Cartesian product is applied between two large tables.

Heuristic rule 4: cache smallest tables on chip. After applying Cartesian products, we sort all tables by sizes and decide the number of small tables to store on chip. Two constraints must be considered during this process. First, the size of selected tables should not exceed assigned on-chip storage. Second, if multiple tables are co-located in the same on-chip bank, the total lookup latency should not exceed off-chip (DDR or HBM) lookups, otherwise caching tables on-chip is meaningless.

Algorithm 1 sketches the heuristic-rule-based search for table combination and allocation. It starts by iterating over the number of tables selected as Cartesian product candidates. Within each iteration, the candidates are quickly combined by applying the first three heuristic rules ($\mathcal{O}(N)$). All tables are then allocated to memory banks efficiently by rule 4 ($\mathcal{O}(N)$). The algorithm ends by returning the searched solution that achieves the lowest embedding lookup latency. Considering the outer loop iterating over Cartesian candidate numbers, the total time complexity of the heuristic algorithm is as low as $\mathcal{O}(N^2)$.

Algorithm 1 Heuristic Search

Input: N: total number of embedding tables; n : number of tables that are selected for Cartesian products; c: candidate tables for Cartesian products; p : all tables after applying Cartesian products
Output: $current\_best$: the best solution found by the algorithm, including the resulting table number and sizes as well as which banks they are allocated to.

for $n \in \{1...N\}$ **do**
  $c \leftarrow$ `select_tables`(n, N) // Heuristic Rule 1
  $p \leftarrow$ `Cartesian_product`(c) // Heuristic Rule 2 & 3
  `solution` $\leftarrow$ `allocate_to_banks`(p) // Heuristic Rule 4
  if `solution` is better than `current_best` **then**
    `current_best` $\leftarrow$ `solution`
  end if
end for
return `current_best`

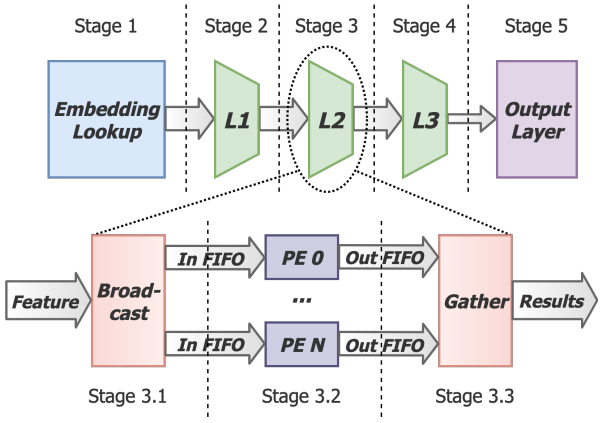

Figure 6. Highly pipelined and parallelized hardware design.

4 FPGA Implementation

In this section, we describe the implementation of MicroRec on an FPGA with an emphasis on its low inference latency.

4.1 Reduce Latency by Deeply Pipelined Dataflow

As shown in Figure 6, we apply a highly pipelined accelerator architecture where multiple items are processed by the accelerator concurrently in different stages. In this design, the embedding lookup stage and three computation stages are pipelined. Each DNN computation module is further divided into three pipeline stages: feature broadcasting, computation, and result gathering. BRAMs or registers are applied to build pipes (FIFOs) as inter-module connections.

Latency concerns (SLA requirements) are eliminated by this highly pipelined design for two reasons. First, input items are processed item by item instead of batch by batch, thus the time to wait and aggregate a batch of recommendation queries is removed. Second, the end-to-end inference latency of a single item is much less than a large batch.

4.2 Embedding Lookup Module

The embedding lookup module gathers and prepares concatenated dense features for fully-connected layers. After receiving lookup indexes, the module concurrently retrieves embedding vectors from HBM, DDR, and on-chip memory banks. The concatenated embeddings are then fed to DNN computation modules through FIFOs.

4.3 DNN Computation Module

The lower half of Figure 6 presents the computation flow for a single FC layer, which consists of three pipeline stages: input feature broadcasting, general matrix-matrix multiplication (GEMM) computation, and result gathering. Partial GEMM is allocated to each processing unit (PE) for better routing design and potentially higher performance (de Fine Licht et al., 2020). Each PE conducts partial GEMM through parallelized multiplications followed by an add tree (Chen et al., 2014).

5 Evaluation

We evaluate the performance of MicroRec for both end-to-end recommendation inference and embedding lookups alone. Given real-world models from Alibaba and the recent recommendation inference benchmark (Gupta et al., 2020b), MicroRec outperforms the optimized CPU baseline significantly under all experiment settings.

5.1 Experiment Environment

We employ Xilinx Alveo U280 FPGA (xil), a high-end card equipped with 8GB of HBM2 (32 channels) and 32 GB of DDR4 (2 channels). We program the FPGA by Vivado HLS (viv), which can translate C++ programs to hardware description language (HDL). The code is then deployed on Vitis (Kathail, 2020) to generate FPGA bitstream.

The software baseline performance is tested on an AWS server with Intel Xeon E5-2686 v4 CPU @2.30GHz (16 vCPU, SIMD operations, i.e., AVX2 FMA, supported) and 128 GB DRAM (8 channels). We apply an open-source solution on deep recommendation systems (Lapis-Hong, 2018), where *TensorFlow Serving* (Olston et al., 2017; Abadi et al., 2016) supports highly optimized model inference.

5.2 Model Specification

We experiment the performance of MicroRec on two classes of models from different sources. The first class contains production models deployed in Alibaba, while the second class comes from the recent recommendation inference benchmark (Gupta et al., 2020b).

5.2.1 *Production Models*

We experiment two deep recommendation models from Alibaba in our experiments. Both of them are memory-access intensive: they contain 47 and 98 embedding tables respectively, much more than current benchmark models (Gupta et al., 2020b), among which the largest model consists of only 12 tables. Table 1 shows the parameters of our models. For example, the smaller recommendation model retrieves one vector from each of the 47 tables and gathers them into a 352-dimensional dense vector to be fed to fully-connected layers. The models we experiment do not contain bottom fully-connected layers, which are adopted in some systems to process dense input features (Gupta et al., 2020b; Ke et al., 2020).

Table 1. Specification of the production models.

Model	Table Num	Feat Len	Hidden-Layer	Size
Small	47	352	(1024,512,256)	1.3 GB
Large	98	876	(1024,512,256)	15.1 GB

5.2.2 *Facebook Recommendation Benchmark*

We also experiment MicroRec on the recent recommendation inference benchmark by Facebook (Gupta et al., 2020b). The benchmark published three classes of recommendation models and their performance breakdown. Although we target to experiment these models for real-world-deployment, the benchmark only published a range of parameters for each type of model. For example, the model class DLRM-RMC2 can contain from 8 to 12 tables, yet no numbers about table sizes and embedding vector lengths are provided. Without such information, it is difficult to compare the inference performance, because some of the parameters are decisive to the inference workload. For instance, embedding vector lengths decide the number of operations to be performed in fully-connected layers.

Therefore, we compare the performance of the embedding layer: given the narrow range of table numbers Gupta et al. (2020b) published, we can conduct multiple experiments and identify a speedup range of MicroRec.

5.3 End-to-End Inference

Table 2 compares the performance of end-to-end recommendation inference on production models between the CPU baseline and MicroRec (both Cartesian and HBM are applied). On the CPU side, performance increases as batch size grows, so we select a large batch size of 2048 as the baseline (larger batch sizes can break inference latency constraints). On the FPGA side, MicroRec infers items without batching as discussed in Section 4.1. Besides, we evaluate the FPGA performance of different precision levels, i.e., 16-bit and 32-bit fixed-point numbers.

MicroRec achieves significant speedup under all experimented settings. In terms of throughput, it is 2.5∼5.4× better than the baseline under two precision levels and two model scale. Moreover, the end-to-end latency to infer a single input item is 16.3∼31.0 microseconds, 3∼4 orders of magnitude lower than common latency requirements (tens of milliseconds). Note that the throughput of MicroRec is not the reciprocal of latency, since multiple items are

Table 2. MicroRec performance on end-to-end recommendation inference. MicroRec achieves 2.5∼5.4× speedup compared to the optimized CPU baseline (the speedup is compared to batch latency of FPGA, which consists of both the stable stages in the middle of the pipeline as well as the time overhead of starting and ending stages). Besides, the end-to-end latency to infer a single input item is as low as a couple of tens of microseconds: the latency concern of online model serving is eliminated.

	CPU B=1	CPU B=64	CPU B=256	CPU B=512	CPU B=1024	CPU B=2048	FPGA fp16	FPGA fp32
Smaller Recommendation Model								
Latency (ms)	3.34	5.41	8.15	11.15	17.17	28.18	**1.63E-2**	**2.26E-2**
Throughput (GOP/s)	0.61	24.04	63.81	93.32	121.16	147.65	**619.50**	**367.72**
Throughput (items/s)	299.71	1.18E+4	3.14E+4	4.59E+4	5.96E+4	7.27E+4	3.05E+5	1.81E+5
Speedup: FPGA fp16	204.72×	24.27×	9.56×	6.59×	5.09×	**4.19**×	-	-
Speedup: FPGA fp32	147.54×	14.58×	5.69×	3.91×	3.02×	**2.48**×	-	-
Larger Recommendation Model								
Latency (ms)	7.48	10.23	15.62	21.06	31.72	56.98	**2.26E-2**	**3.10E-2**
Throughput (GOP/s)	0.42	19.48	51.03	75.66	100.49	111.89	**606.41**	**379.45**
Throughput (items/s)	133.68	6.26E+3	1.64E+3	2.43E+4	3.23E+4	3.59E+4	1.95E+5	1.22E+5
Speedup: FPGA fp16	331.51×	29.56×	11.73×	7.96×	6.02×	**5.41**×	-	-
Speedup: FPGA fp32	241.54×	18.67×	7.36×	4.99×	3.77×	**3.39**×	-	-

processed by the deep pipeline at the same time.

5.4 Embedding Lookup Performance

We highlight the performance boost of embedding lookups brought by Cartesian products and HBM in this section on both the production models and the benchmark models.

5.4.1 Lookups on Production Models

MicroRec outperforms CPU baseline significantly on production models as shown in Table 4. Same as Section 5.3, a large batch size of 2048 is selected for the CPU baseline to achieve high throughput, while the accelerator always processes inputs item by item (no concept of batch sizes). This latency excludes streaming input features from CPU side memory as mentioned in footnote 2. The result shows that MicroRec outperforms the baseline by 13.8∼14.7× on the embedding layer (in addition to DRAM accesses, the many embedding-related operator calls in TensorFlow also leads to large consumption in the CPU baseline). Some detailed result interpretation includes:

Though HBM can achieve satisfying performance on its own, Cartesian products further speed up the process. For the smaller model, as shown in Table 3, except those tiny tables stored on-chip, there are still 39 tables left to be allocated to DRAM. Considering there are 34 DRAM channels in total (32 for HBM, 2 for DDR), it takes two DRAM access rounds to lookup 39 tables. Cartesian products can reduce the table number to 34, so that only one round of DRAM access is required. The experiment shows that, with Cartesian products, the latency of embedding lookup is only 59.17% of the HBM-only solution (458 ns vs 774 ns). Similarly, for the larger model, Cartesian products reduce the memory access rounds from 3 to 2, consumed only 72.12% of the time (1.63 us vs 2.26 us).

The storage overhead of Cartesian products is fairly low. As shown in table 3, the products only lead to 3.2% and 1.9% storage overhead on the two models respectively. This is because only small tables are selected for Cartesian products as introduced in section 3.4, so that the products are still considerably small compared to a single large table.

By Cartesian products and HBM, the memory bottleneck caused by embedding lookup is eliminated. Since the embedding lookups only cost less than 1 microsecond in MicroRec (as in Table 4), the bottleneck shifts back to computation, in which the most expensive stage takes several microseconds.

The accelerator performance is robust even as multiple rounds of lookups are required. Although the production models only involves one lookup operations per table, alternative DNN architectures may require multiple rounds of lookups (Gupta et al., 2020b). Figure 7 proves the performance robustness of MicroRec in such scenarios by assuming more rounds of embedding retrievals on the two production models — the smaller and larger models can tolerate 6 and 4 rounds of lookups without downgrading the end-to-end inference throughput at all using 16-bit fixed-points, because the DNN computation and embedding lookup stages are overlapped. Once more rounds of lookups are assumed, the performance starts to depend on the total memory access latency which is proportional to the rounds of DRAM accesses.

Table 3. Benefit and overhead of Cartesian products. It only costs marginal extra storage to achieve significant speedup.

	Table Num	Tables in DRAM	DRAM Access Rounds	Storage	Lookup Latency
Smaller Recommendation Model					
Without Cartesian	47	39	2	100%	100%
With Cartesian	42	34	1	**103.2%**	**59.2%**
Larger Recommendation Model					
Without Cartesian	98	82	3	100%	100%
With Cartesian	84	68	2	**101.9%**	**72.1%**

Table 4. MicroRec performance on the embedding layer. Given the same element data width of 32-bits, it outperformed the optimized CPU baseline by over one order of magnitude. Besides, it only took no more than one microsecond to finish lookups and concatenations even in embedding-intensive models (47 and 98 tables).

	CPU B=1	CPU B=64	CPU B=256	CPU B=512	CPU B=1024	CPU B=2048	FPGA: HBM	FPGA: HBM + Cartesian
Smaller Recommendation Model								
Latency (ms)	2.59	3.86	4.71	5.96	8.39	12.96	7.74E-4	**4.58E-4**
Speedup: HBM	3349.97×	77.91×	23.75×	15.04×	10.59×	**8.17**×	-	-
Speedup: HBM + Cartesian	5665.07×	131.76×	40.16×	25.44×	17.91×	**13.82**×	-	-
Larger Recommendation Model								
Latency (ms)	6.25	8.05	10.92	13.67	18.11	31.25	1.38E-3	**1.03E-3**
Speedup: HBM	4531.23×	91.29×	30.94×	19.36×	12.83×	**11.07**×	-	-
Speedup: HBM + Cartesian	6019.37×	121.28×	41.10×	25.72×	17.04×	**14.70**×	-	-

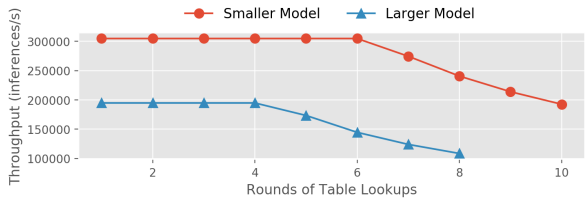

Figure 7. End-to-end inference throughput of MicroRec. It allows multi-rounds lookup without sacrificing performance.

Table 5. MicroRec achieves 18.7∼72.4× embedding lookup speedup compared to the Facebook's recommendation baseline.

Performance	Embedding Vector Length				
	4	8	16	32	64
8 Tables (Speedup Upper Bound)					
Lookup (ns)	334.5	353.7	411.6	486.3	648.4
Speedup	72.4×	68.4×	58.8×	49.7×	37.3×
12 Tables (Speedup Lower Bound)					
Lookup (ns)	648.5	707.4	817.4	972.7	1296.9
Speedup	37.3×	34.2×	29.6×	24.8×	18.7×

5.4.2 *Performance on Benchmark Models*

We compare the embedding lookup performance of MicroRec to the recent recommendation inference benchmark (Gupta et al., 2020b). Although the paper does not expose all model parameters, we can still identify the embedding lookup performance range on MicroRec by experimenting a range of table settings. To be more specific, we experiment the embedding-dominated model class DLRM-RMC2, which contains 8∼12 small tables and each table is looked up 4 times (thus 32 ∼ 48 lookups in total). Several assumptions are made for the missed information. First, by "small tables", we assume each table is within the capacity of an HBM bank (256MB). Second, we assume common embedding vector lengths from 4 to 64. Third, no Cartesian products are applied in our experiments, since the table sizes are assumed by us.

Table 5 shows the embedding lookup performance on MicroRec: it achieves 18.7∼72.4× speedup compared to the published baseline performance (2 sockets of Broadwell CPU @2.4GHz; 14 cores per socket; AVX-2 supported; 256 GB 2400MHz DDR4 DRAM; batch size=256). This performance range is identified by experimenting table numbers from 8 to 12 and vector lengths from 4 to 64. The highest speedup occurred when there are only 8 embedding tables (32 lookups) with a short vector size of 4, for which only

one round on HBM lookup is required. The lowest speedup happens when there are 12 tables with a long vector size of 64, where 2 rounds of HBM accesses are necessary.

6 Related Work

Deep learning for personalized recommendations. While regular DNNs take dense features as input, He et al. (2017) proposed to encode user information by embedding tables in recommendation models. The dense features gathered from embedding vectors are then processed by a series of matrix-factorization and FC layers. Covington et al. (2016) applied similar neural network structures for Youtube video recommendation. Cheng et al. (2016) discussed the pros-and-cons of linear models and deep models, and proposed to serve Google Play recommendations with joint wide-and-deep models. Zhao et al. (2019) improved the wide-and-deep model by taking multiple objectives into account beyond click-through rates (CTR), e.g., comments, likes, and ratings. Facebook introduced additional fully-connected layers to process dense input features: the dense features are fed into bottom FC layers, and the output features are concatenated with embedding vectors (Gupta et al., 2020b). Alibaba removed dense input features in their deep models and applied attention mechanism on the top of embedding tables (Zhou et al., 2018). Zhou et al. (2019) further extended this work by introducing sequential neural network.

Hardware solutions for recommendations. According to Facebook, recommendation workloads can consume up to 79% of total AI inference cycles in data centers (Gupta et al., 2020b). However, little research has been focused on serving personalized recommendations efficiently. In order to provide enough background knowledge to the research community and tackle this important problem, Gupta et al. (2020b) analyzed the recommendation workload comprehensively, open-sourced several models used by Facebook, and set up a performance benchmark. Kwon et al. (2019) is the first hardware solution for high performance recommendation inference. They reduced the memory bottleneck by introducing DIMM-level parallelism in DRAM and supporting tensor operations, e.g., gather and reduction, within the DRAM. Ke et al. (2020) extended the idea of near-memory-processing and added memory-side-caching for frequently-accessed entries. Gupta et al. (2020a) took into account the characteristics of query sizes and arrival patterns, and developed an efficient scheduling algorithm to maximize throughput under latency constraints by using both CPUs and GPUs. Hwang et al. (2020) implemented an FPGA accelerator (without HBM) for deep recommendation inference, and the speedup was significant for models with few embedding tables. Compared to previous work, MicroRec is the first system that introduces data structure solution, i.e., Cartesian products, to reduce the number of DRAM accesses. It is also the first work resorting to HBM so as to parallelize embedding lookups.

Efficient Model Serving. Aside from recommendations, a range of research has been focused on neural network serving. Due to the heavy workload of DNN inference, many works resort to specialized hardware (Jouppi et al., 2017; Mei et al., 2019; Chung et al., 2018; Hsieh et al., 2018; Zhang & Li, 2017; Chen et al., 2016; Mao et al., 2019; Sharify et al., 2019; Shao et al., 2019; Feng et al., 2019; Owaida et al., 2017; Gao et al., 2017; Chen et al., 2014; Hua et al., 2019; Farcas et al., 2020). Besides, designing hardware-efficient neural networks is essential for inference performance (Han et al., 2017; Stamoulis et al., 2019; Teja Mullapudi et al., 2018; Zhang et al., 2019; Howard et al., 2017; Elthakeb et al., 2018; Ghasemzadeh et al., 2018; Maschi et al., 2020). Furthermore, one can optimize serving performance on general-purpose hardware (CPU and GPU) by system-level optimization (Olston et al., 2017; Narayanan et al., 2018; Chen et al., 2018b; Choi & Rhu, 2020; Wu et al., 2019; Crankshaw et al., 2018).

Efficient Model Training. Due to the increasing numbers and sizes of neural networks, high-performance model training becomes essential (Mattson et al., 2019). Training usually resorts to accelerators such as GPUs (Shoeybi et al., 2019; Cho et al., 2019b; Dong et al., 2020; Cui et al., 2016) and FPGAs (Zhang et al., 2017; Cho et al., 2019a; Kara et al., 2017; He et al., 2020; 2018; Zhao et al., 2016; Gürel et al., 2020). Besides, many works accelerate training by better system and algorithm designs (Jayarajan et al., 2019; Das et al., 2018; Peng et al., 2019; Narayanan et al., 2019; Jia et al., 2018; Wang et al., 2019; Moritz et al., 2018; Kurth et al., 2017; Rajbhandari et al., 2017; Li et al., 2020; 2014; Chen et al., 2018a; Abuzaid et al., 2016).

7 Conclusion

We design and implement MicroRec, a high-performance deep recommendation inference engine. On the data structure side, MicroRec applies Cartesian products to reduce sparse memory accesses. On the hardware side, HBM is adopted to scale up embedding lookup concurrency, and the deeply pipelined architecture design on FPGA enables low inference latency. By the three strategies we propose, the memory bottleneck caused by embedding lookups is almost eliminated, and the latency requirements of recommendation inference are easily met.

Acknowledgements

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

Vivado high-level synthesis. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

Alveo u280 data center accelerator card. https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

Abuzaid, F., Bradley, J. K., Liang, F. T., Feng, A., Yang, L., Zaharia, M., and Talwalkar, A. S. Yggdrasil: An optimized system for training deep decision trees at scale. In *Advances in Neural Information Processing Systems*, pp. 3817–3825, 2016.

Chen, L., Wang, H., Zhao, J., Papailiopoulos, D., and Koutris, P. The effect of network width on the performance of large-batch training. In *Advances in Neural Information Processing Systems*, pp. 9302–9309, 2018a.

Chen, T., Moreau, T., Jiang, Z., Shen, H., Yan, E. Q., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. Tvm: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, 11:20, 2018b.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622. IEEE, 2014.

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

Cho, H., Oh, P., Park, J., Jung, W., and Lee, J. Fa3c: Fpga-accelerated deep reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 499–513, 2019a.

Cho, M., Finkler, U., and Kung, D. Blueconnect: Novel hierarchical all-reduce on multi-tired network for deep learning. In *Proceedings of the Conference on Systems and Machine Learning (SysML)*, 2019b.

Choi, Y. and Rhu, M. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 220–233. IEEE, 2020.

Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., and Malhotra, S. Notes from the ai frontier: Insights from hundreds of use cases. *McKinsey Global Institute*, 2018.

Chung, E., Fowers, J., Ovtcharov, K., Papamichael, M., Caulfield, A., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., et al. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38 (2):8–20, 2018.

Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

Crankshaw, D., Sela, G.-E., Zumar, C., Mo, X., Gonzalez, J. E., Stoica, I., and Tumanov, A. Inferline: Ml inference pipeline composition framework. *arXiv preprint arXiv:1812.01776*, 2018.

Cui, H., Zhang, H., Ganger, G. R., Gibbons, P. B., and Xing, E. P. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, pp. 1–16, 2016.

Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., et al. Mixed precision training of convolutional neural networks using integer operations. *arXiv preprint arXiv:1802.00930*, 2018.

de Fine Licht, J., Kwasniewski, G., and Hoefler, T. Flexible communication avoiding matrix multiplication on fpga with high-level synthesis. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 244–254, 2020.

Dong, J., Cao, Z., Zhang, T., Ye, J., Wang, S., Feng, F., Zhao, L., Liu, X., Song, L., Peng, L., et al. Eflops: Algorithm and system co-design for a high performance distributed training platform. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 610–622. IEEE, 2020.

Elthakeb, A. T., Pilligundla, P., Mireshghallah, F., Yazdanbakhsh, A., Gao, S., and Esmaeilzadeh, H. Releq: an automatic reinforcement learning approach for deep quantization of neural networks. *arXiv preprint arXiv:1811.01704*, 2018.

Farcas, A.-J., Li, G., Bhardwaj, K., and Marculescu, R. A hardware prototype targeting distributed deep learning for on-device inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 398–399, 2020.

Feng, Y., Whatmough, P., and Zhu, Y. Asv: accelerated stereo vision system. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 643–656, 2019.

Gao, M., Pu, J., Yang, X., Horowitz, M., and Kozyrakis, C. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 751–764, 2017.

Ghasemzadeh, M., Samragh, M., and Koushanfar, F. Rebnet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 57–64. IEEE, 2018.

Gomez-Uribe, C. A. and Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

Gupta, U., Hsia, S., Saraph, V., Wang, X., Reagen, B., Wei, G.-Y., Lee, H.-H. S., Brooks, D., and Wu, C.-J. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. pp. 790–803, 2020a.

Gupta, U., Wu, C.-J., Wang, X., Naumov, M., Reagen, B., Brooks, D., Cottel, B., Hazelwood, K., Hempstead, M., Jia, B., et al. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 488–501. IEEE, 2020b.

Gürel, N. M., Kara, K., Stojanov, A., Smith, T., Lemmin, T., Alistarh, D., Püschel, M., and Zhang, C. Compressive sensing using iterative hard thresholding with low precision data representation: Theory and applications. *IEEE Transactions on Signal Processing*, 68:4268–4282, 2020.

Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., Yan, H., and Dally, W. J. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84, 2017.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

He, Z., Sidler, D., István, Z., and Alonso, G. A flexible k-means operator for hybrid databases. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 368–3683. IEEE, 2018.

He, Z., Wang, Z., and Alonso, G. Bis-km: Enabling any-precision k-means on fpgas. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 233–243, 2020.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hsieh, K., Ananthanarayanan, G., Bodik, P., Venkataraman, S., Bahl, P., Philipose, M., Gibbons, P. B., and Mutlu, O. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 269–286, 2018.

Hua, W., Zhou, Y., De Sa, C., Zhang, Z., and Suh, G. E. Boosting the performance of cnn accelerators with dynamic fine-grained channel gating. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 139–150, 2019.

Hwang, R., Kim, T., Kwon, Y., and Rhu, M. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. pp. 790–803, 2020.

Jayarajan, A., Wei, J., Gibson, G., Fedorova, A., and Pekhimenko, G. Priority-based parameter propagation for distributed dnn training. *arXiv preprint arXiv:1905.03960*, 2019.

Jia, Z., Zaharia, M., and Aiken, A. Beyond data and model parallelism for deep neural networks. *arXiv preprint arXiv:1807.05358*, 2018.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

Jun, H., Cho, J., Lee, K., Son, H.-Y., Kim, K., Jin, H., and Kim, K. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4. IEEE, 2017.

Kara, K., Alistarh, D., Alonso, G., Mutlu, O., and Zhang, C. Fpga-accelerated dense linear machine learning: A precision-convergence trade-off. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 160–167. IEEE, 2017.

Kathail, V. Xilinx vitis unified software platform. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 173–174, 2020.

Ke, L., Gupta, U., Cho, B. Y., Brooks, D., Chandra, V., Diril, U., Firoozshahian, A., Hazelwood, K., Jia, B., Lee, H.-H. S., et al. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 790–803. IEEE, 2020.

Kurth, T., Zhang, J., Satish, N., Racah, E., Mitliagkas, I., Patwary, M. M. A., Malas, T., Sundaram, N., Bhimji, W., Smorkalov, M., et al. Deep learning at 15pf: supervised and semi-supervised classification for scientific data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2017.

Kwon, Y., Lee, Y., and Rhu, M. 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*, pp. 740–753, 2019.

Lapis-Hong. Lapis-hong/wide_deep. https://github.com/Lapis-Hong/wide_deep, Oct 2018.

Li, C., Chen, T., You, H., Wang, Z., and Lin, Y. Halo: Hardware-aware learning to optimize. In *The 16th European Conference on Computer Vision (ECCV 2020)*, 2020.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 583–598, 2014.

Mao, J., Yang, Q., Li, A., Li, H., and Chen, Y. Mobieye: An efficient cloud-based video detection system for real-time mobile applications. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Maschi, F., Owaida, M., Alonso, G., Casalino, M., and Hock-Koon, A. Making search engines faster by lowering the cost of querying business rules through fpgas. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 2255–2270, 2020.

Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., Tang, H., Wei, G.-Y., Bailis, P., Bittorf, V., et al. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*, 2019.

Mei, L., Dandekar, M., Rodopoulos, D., Constantin, J., Debacker, P., Lauwereins, R., and Verhelst, M. Sub-word parallel precision-scalable mac engines for efficient embedded dnn inference. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 6–10. IEEE, 2019.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 561–577, 2018.

Narayanan, D., Santhanam, K., Phanishayee, A., and Zaharia, M. Accelerating deep learning workloads through efficient multi-model execution. In *NeurIPS Workshop on Systems for Machine Learning*, pp. 20, 2018.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., and Soyke, J. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

Owaida, M., Zhang, H., Zhang, C., and Alonso, G. Scalable inference of decision tree ensembles: Flexible design for cpu-fpga platforms. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8. IEEE, 2017.

O'Connor, M. Highlights of the high-bandwidth memory (hbm) standard. In *Memory Forum Workshop*, 2014.

Park, J., Naumov, M., Basu, P., Deng, S., Kalaiah, A., Khudia, D., Law, J., Malani, P., Malevich, A., Nadathur, S., et al. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. *arXiv preprint arXiv:1811.09886*, 2018.

Peng, Y., Zhu, Y., Chen, Y., Bao, Y., Yi, B., Lan, C., Wu, C., and Guo, C. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 16–29, 2019.

Rajbhandari, S., He, Y., Ruwase, O., Carbin, M., and Chilimbi, T. Optimizing cnns on multicores for scalability, performance and goodput. *ACM SIGARCH Computer Architecture News*, 45(1):267–280, 2017.

Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 14–27, 2019.

Sharify, S., Lascorz, A. D., Mahmoud, M., Nikolic, M., Siu, K., Stuart, D. M., Poulos, Z., and Moshovos, A. Laconic deep learning inference acceleration. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 304–317. IEEE, 2019.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497. Springer, 2019.

Teja Mullapudi, R., Mark, W. R., Shazeer, N., and Fatahalian, K. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8080–8089, 2018.

Underwood, C. Use cases of recommendation systems in business-current applications and methods, 2019.

Wang, Z., Kara, K., Zhang, H., Alonso, G., Mutlu, O., and Zhang, C. Accelerating generalized linear models with mlweaving: A one-size-fits-all system for any-precision learning. *Proceedings of the VLDB Endowment*, 12(7): 807–821, 2019.

Wang, Z., Huang, H., Zhang, J., and Alonso, G. Benchmarking high bandwidth memory on fpgas. 2020.

Wu, C.-J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia, Y., Jia, B., et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 331–344. IEEE, 2019.

Xie, X., Lian, J., Liu, Z., Wang, X., Wu, F., Wang, H., and Chen, Z. Personalized recommendation systems: Five hot research topics you must know. *Microsoft Research Lab-Asia*, 2018.

Zhang, H., Li, J., Kara, K., Alistarh, D., Liu, J., and Zhang, C. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*, pp. 4035–4043, 2017.

Zhang, J. and Li, J. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 25–34, 2017.

Zhang, X., Lu, H., Hao, C., Li, J., Cheng, B., Li, Y., Rupnow, K., Xiong, J., Huang, T., Shi, H., et al. Skynet: a hardware-efficient method for object detection and tracking on embedded systems. *arXiv preprint arXiv:1909.09709*, 2019.

Zhao, W., Fu, H., Luk, W., Yu, T., Wang, S., Feng, B., Ma, Y., and Yang, G. F-cnn: An fpga-based framework for training convolutional neural networks. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 107–114. IEEE, 2016.

Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., Kumthekar, A., Sathiamoorthy, M., Yi, X., and Chi, E. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 43–51, 2019.

Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., and Gai, K. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1059–1068, 2018.

Zhou, G., Mou, N., Fan, Y., Pi, Q., Bian, W., Zhou, C., Zhu, X., and Gai, K. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5941–5948, 2019.

APPENDIX: MEMORY CONTROLLER AND AXI INTERFACE

To set up the communication between FPGA and DRAM (including HBM and DDR), we choose a narrow AXI interface data width of 32-bit. Although the full data width (512-bit) can reduce the number of clock cycles required for vector reading, it has two disadvantages. First, it consumes too much hardware resources. To support efficient communication to DRAM without much stalls, we apply BRAMs as long FIFOs. Since there are 34 DRAM channels in total (32 for HBM and 2 for DDR), these FIFOs will consume over half of total BRAMs slices on Alveo U280 FPGA given 512-bit data width. Such BRAM consumption is too expensive to afford because DNN computation modules also require substantial BRAM resources. Second, higher resource utilization can lead to downgraded clock frequency, resulting in lower inference performance. According to the experiments in section 5.3 and 5.4, the embedding lookup process in our design is fast enough to be covered by DNN computation (remember we applied a pipelined design). As a result, lower clock frequency will lead to decreased computation performance thus higher inference latency.

APPENDIX: FPGA RESOURCE UTILIZATION

Table 6 lists the resource utilization and clock frequency of our deep recommendation inference accelerator. We implement the design on Xilinx Alveo U280, a high-end FPGA card consisting of three die areas. The resource consumptions are composed of all GEMM PEs, their interconnection, and the embedding lookup module. According to the estimation of Vivado HLS (the consumption can be further optimized by the Vivado backend), each PE for 32-bits fixed-point GEMM consumes 7 BRAM slices and 18 DSPs while the 16-bit one consumes 4 BRAM slices and 14 DSPs. The number of PEs for three layers are 128, 128, and 32 for both models and precision-levels. Because of the high resource utilization rate (more than 80% for some resources), cross-die routing is necessary, and the long-distance communication must be tolerated by low clock frequency (120∼140MHz).

Table 6. FPGA frequency & resource utilization (Xilinx Alveo U280)

	Small Model		Large Model	
Precision	fixed-point 16	fixed-point 32	fixed-point 16	fixed-point 32
Freq (MHz)	120	140	120	135
	Utilization (Slices)			
BRAM 18Kbit	1,566	1,657	1,566	1721
DSP48E	4,625	5,193	4,625	5,193
Flip-Flop	683,641	764,067	691,042	777,527
LUT	485,323	568,864	514,517	584,220
URAM 288Kbit	642	770	642	770
	Utilization (%)			
BRAM 18Kbit	78	82	78	85
DSP48E	51	57	51	57
Flip-Flop	26	29	27	30
LUT	37	44	40	45
URAM 288Kbit	66	66	80	80

APPENDIX: COST ESTIMATION

We compare the price between CPU-based and FPGA-based inference engine on AWS. The CPU server we rent costs $1.82 per hour while renting an FPGA server only costs $1.65 (AWS provides U250, a similar model to what we use). Considering the 4∼5x speedup using 32-bit fixed-points, deploying FPGAs will be beneficial in the long-term.