RM-SSD: In-Storage Computing for Large-Scale Recommendation Inference

Xuan Sun*[1], Hu Wan*[1], Qiao Li[1], Chia-Lin Yang[2], Tei-Wei Kuo[1,2,3], and Chun Jason Xue[1]

[1]Department of Computer Science, City University of Hong Kong
[2]Department of Computer Science and Information Engineering, National Taiwan University
[3]NTU High Performance and Scientific Computing Center, National Taiwan University

***Abstract*—To meet the strict service level agreement requirements of recommendation systems, the entire set of embeddings in recommendation systems needs to be loaded into the memory. However, as the model and dataset for production-scale recommendation systems scale up, the size of the embeddings is approaching the limit of memory capacity. Limited physical memory constrains the algorithms that can be trained and deployed, posing a severe challenge for deploying advanced recommendation systems. Recent studies offload the embedding lookups into SSDs, which targets the embedding-dominated recommendation models. This paper takes it one step further and proposes to offload the entire recommendation system into SSD with in-storage computing capability. The proposed SSD-side FPGA solution leverages a low-end FPGA to speed up both the embedding-dominated and MLP-dominated models with high resource efficiency. We evaluate the performance of the proposed solution with a prototype SSD. Results show that we can achieve 20-100× throughput improvement compared with the baseline SSD and 1.5-15× improvement compared with the state-of-art.**

I. Introduction

Recommendation systems are heavily utilized today in online services, e.g., e-commerce [1], video and music streaming [2]–[5], and social media platforms [6], [7], to provide personalized recommendations based on users' interests and preferences. Inspired by the high accuracy of deep learning, recommendation models based on deep neural networks (DNNs) have drawn a lot of attention [2], [3], [6], [8]–[14] in recent years. Besides the basic multi-layer perceptron (MLP) layer, deep learning-based recommendation models often introduce a data-intensive embedding layer, which is used to handle sparse categorical input features. For example, Facebook's post recommendation model [15] contains hundreds of or thousands of categorical features (e.g., users, posts, and pages), while each categorical feature could take on as many as tens of millions (e.g., words and figures of the post) and even tens of billions (e.g., social media pages) of possible categories. In the embedding layer, the aforementioned sparse features are mapped to embedding vectors (EVs) and stored in various embedding tables. When handling recommendation inference, the related embedding vectors are fetched from the embedding tables according to the lookup indices, which is defined as embedding lookup. After aggregating vectors from each embedding table, the results of embedding lookup will be fed to the MLP layers for further processing.

The rapidly growing embedding tables bring new challenges for recommendation systems. Currently, the entire set of embeddings in recommendation systems is stored in the DRAM to reduce the latency for serving user requests. The model size is around hundreds of gigabytes, which is approaching the memory limit of typical servers (16-256GB) [16]. As the model of production-scale recommendation systems scales up, the size of the embedding layer can easily reach terabytes and beyond in the near future [17]–[19]. Storing all embeddings in DRAM will become infeasible. On the other hand, DRAM is expensive in terms of cost per gigabyte. Unlimited DRAM capacity expansion would increase the total cost of ownership (TCO). To tackle the large-embedding issue in recommendation systems, one natural solution is to move the large-scale embedding tables to flash-based solid-state drives (SSDs).

However, a naïve implementation of recommendation systems inside SSD would bring long latency and throughput degradation for all recommendation models. We profile the recommendation inferences in detail from the perspectives of latency and input characteristics. Based on the execution time of the embedding layer and MLP layers, recommendation models can be identified as embedding-dominated and MLP dominated models. The performance degradation of the former mainly comes from the significant read amplification and irregular embedding access patterns [15], [20], [21]. Recent work RecSSD [20] proposes to utilize the in-storage computing (ISC) approach to reduce the inference latency by conducting the embedding lookup in SSD, which is further enhanced by an LRU cache. RecSSD optimizes embedding-dominated but not MLP-dominated models.

In this paper, we propose RM-SSD, an SSD with FPGA-based in-storage computing engine, to offload the entire recommendation system. It is the first **complete** solution that can improve the performance of both embedding-dominated and MLP-dominated models while storing all embedding tables in SSDs. To reduce the long latency of the embedding layer, an Embedding Lookup Engine is proposed in RM-SSD to eliminate read amplification by applying a two-stage fine-

*Xuan Sun and Hu Wan contributed equally to this work.

grained read strategy. Embedding Lookup Engine accesses embedding vectors from flash chips in vector-granularity and conducts the pooling operation for each embedding table. To accelerate the MLP layers in recommendation systems, a MLP Acceleration Engine is proposed inside RM-SSD. MLP Acceleration Engine applies intra-layer decomposition, inter-layer composition, and kernel search to minimize FPGA resource consumption and maximize throughput. Finally, as part of software integration with the host, we propose recommendation system semantic-aware interfaces to narrow the I/O semantic gap with negligible overhead.

The main contributions of this work are as follows:

- We propose an in-storage computing scheme based on FPGA (RM-SSD) for accelerating SSD-based recommendation system inference.
- We design an Embedding Lookup Engine to accelerate the embedding layers and eliminate read amplification by taking advantage of a two-stage fine-grained read strategy.
- We design a MLP Acceleration Engine to accelerate the MLP layers by remapping the recommendation model topology to FPGA with pipelining. We exploit the characteristics of the recommendation model to achieve resource efficiency.
- We develop a recommendation system semantic-aware interface between host and SSD for latency and throughput optimization.
- We implement RM-SSD and conduct comprehensive experiments. Evaluations show that RM-SSD can achieve 20-100× throughput improvement compared with the baseline SSD and 1.5-15× improvement compared to the latest work [20].

The rest of the paper is organized as follows. Section II introduces the background on recommendation systems and in-storage computing. Section III illustrates the motivation for offloading the recommendation system. Section IV and V describe the design and implementation of the proposed RM-SSD. Section VI shows the experimental results for RM-SSD. Section VII provides the related work and Section VIII presents the conclusion.

II. Background

A. Recommendation Systems and Models

To accurately model user behaviors and preferences, state-of-the-art recommendation models take advantage of deep learning solutions [2], [3], [6], [8], [10]–[12]. Fig. 1 depicts a general architecture of deep learning-based recommendation models. The inputs to recommendation models are a set of dense and sparse features. Dense features are processed by the bottom MLP layers, while sparse features are transformed to dense representations by performing embedding table lookups. Each sparse index is used to look up a unique embedding vector in an embedding table. The embedding

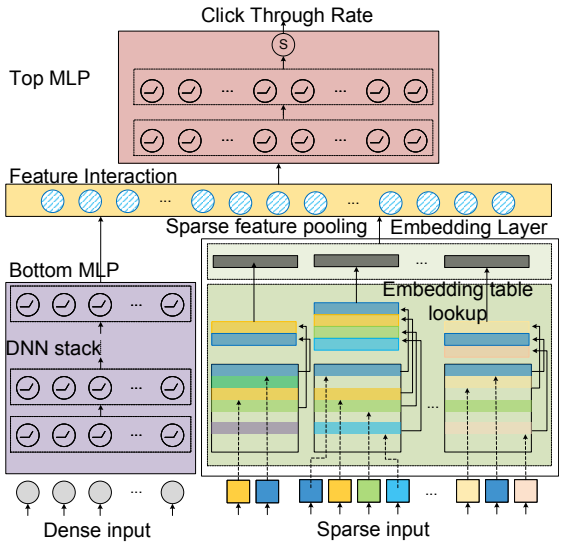


Figure 1. A typical architecture of the DNN-based recommendation model.

vectors returned from embedding tables are first aggregated into a single vector via element-wise pooling operations (e.g., addition, average), and then concatenated with dense features via feature interaction. Finally, the outputs of these transformations are processed by the top MLP layers. The output is a single value representing the probability of a certain event, such as the click-through-rate (CTR).

Modern production recommendation models often possess an extremely large feature set with user behaviors and preferences. The embeddings are obtained by using very complex models to encode rich semantic relations in vectors. Thus, production recommendation systems require large-capacity memory, and their performance is often bounded by memory-intensive sparse embedding operations.

B. Opportunities for In-Storage Computing

Rather than treating SSD as pure storage device, enriching SSD with computing capability is a popular trend in recent years because of the exponential growth of data. The main reasons for offloading partial computing workloads into storage (or called in-storage computing, ISC) are as follows.

Mismatch bandwidth. Thanks to the multi-level parallelism in flash arrays, the internal bandwidth of a commercial SSD can be more than 10,000 MB/s, while the external bandwidth cannot catch up with the fast speed. For a typical high-end SSD, the communication protocol between the host and SSD is NVM Express (NVMe) through the PCIe lane. The latest Samsung SSD [22] applies PCIe Gen 4x4 NVMe interface, and the sequential read speed can be up to 7,000 MB/s, which is actually the maximum PCIe transfer bandwidth. We can take full advantage of the parallelism from the perspective of SSD-level, channel-level, and chip-level through offloading computations into the SSD directly.

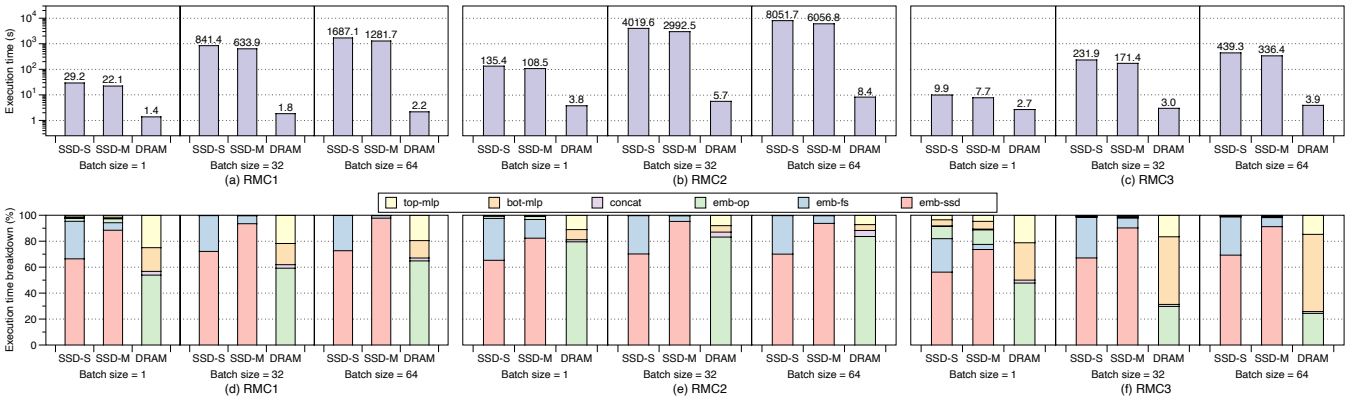


Figure 2. Performance of naïve SSD deployment for recommendation inference. (a)-(c) show the logarithmic execution time of 1K inferences. (d)-(f) indicate the detailed time breakdown of inference. SSD-S, SSD-M, and DRAM indicate recommendation models running with small size (1/4 size of embedding tables), medium size (1/2 size of embedding tables), and full size of DRAM.

Redundant data movement. In many cases of data processing, the bottleneck is the redundant data movement, such as filtering in database [23], data analysis in statistics [24]. These applications usually possess the characteristic that the output size can be ignored compared with the huge input size. The system performance is often degraded by the redundant data movement from storage to the host. Hu et al. [25] demonstrate that the time spent on data transfer can be several times more than kernel execution, even though a high-end NVMe SSD is employed. The recommendation systems studied in this paper also exhibit this pattern.

III. Motivation

A. Challenge of Large-Scale Recommendation Systems

Similar to other DNN-based systems, current recommendation systems adopt the in-memory processing approach: the entire models are loaded into the main memory. As described in Section II, the size of a recommendation model is almost equal to the size of embedding tables. The size of embedding tables scales proportionally to the number of embedding tables, the number of embedding vectors in each table, and the dimension of the embedding vector. As a result, embedding tables can easily exceed the DRAM capacity. Currently, the model size is around hundreds of gigabytes and is growing to terabytes in the near future [17]–[19]. Unlimited DRAM is neither realistic nor economical.

High-density SSD is a natural solution for storing large-scale data with 4-8× lower prices than DRAM. Compared with unlimited DRAM expansion, SSD deployment is a much more economical solution. However, simply employing SSD would cause severe performance degradation due to course-grained block access, I/O semantic gap, and the intrinsic long latency of flash. This brings new challenges for adopting SSD in the recommendation system with acceptable latency and throughput. To investigate the effects of deploying SSD in recommendation systems, we first look into the detailed performance of naïve SSD deployment.

B. Performance of Naïve SSD Deployment

We utilize Facebook's open-sourced DLRM framework [6] based on PyTorch without memory limitation as the ideal case. For the SSD version, we use a commercial NVMe SSD to store all the embedding tables and replace the original *SparseLengthSum* (SLS) operator with a customized C++ function. The C++ SLS operator will first read all the required embedding vectors stored on SSD by `lseek`, and then perform sum operation by utilizing OpenMP. The cache policy of SSD is handled by the file system. To emulate the real situation of insufficient memory, we limit the DRAM size to 1/4 (SSD-S) and 1/2 (SSD-M) size of embedding tables to measure the performance with various constrained memory sizes. We also synthesize the input traces based on the locality of the public Kaggle Criteo Ad Competition dataset by applying the method in [20]. We measure the inference performance with various batch sizes of three Facebook's recommendation models, DLRM-RMC1, DLRM-RMC2, DLRM-RMC3 on both the ideal DRAM-only version, and the naïve SSD version.

*1) **Performance profiling**:* Fig. 2 shows the execution time of 1K inferences. Each run is preceded by a warm-up period of sufficient length to ensure that we measure steady-state execution only. Time consummation of MLP layers comes from `top-mlp`, `bot-mlp` and `concat`, while embedding layers sum up `emb-ssd`, `emb-fs`, `emb-op`. `emb-ssd` represents the time consumption in the SSD only (end with device driver); `emb-fs` indicates the time cost of I/O stack, e.g. page cache; and `emb-op` is the SLS operation in the userspace.

It is not surprising that for all recommendation models, the performance of the SSD version drops significantly compared to the DRAM-only version, and becomes worse when the memory size is further limited. It is also expected

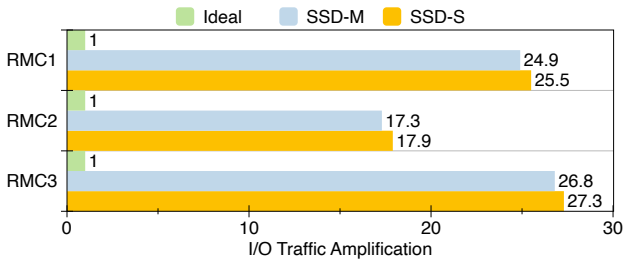
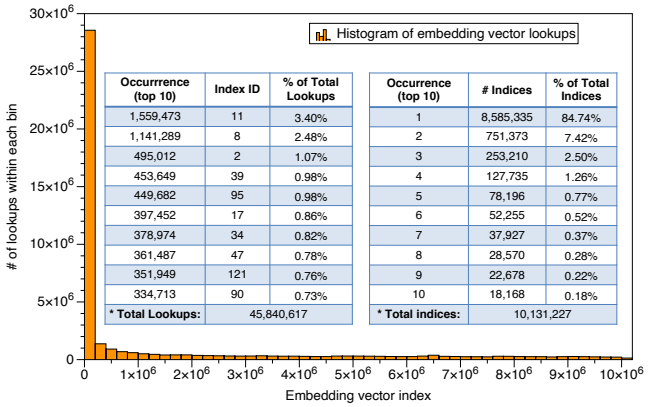


Figure 3. Read amplification of the recommendation system under different settings. The *Ideal* traffic assumes a byte-addressable storage device.



Occurrrence (top 10)	Index ID	% of Total Lookups
1,559,473	11	3.40%
1,141,289	8	2.48%
495,012	2	1.07%
453,649	39	0.98%
449,682	95	0.98%
397,452	17	0.86%
378,974	34	0.82%
361,487	47	0.78%
351,949	121	0.76%
334,713	90	0.73%
* Total Lookups:	45,840,617	

Occurrence (top 10)	# Indices	% of Total Indices
1	8,585,335	84.74%
2	751,373	7.42%
3	253,210	2.50%
4	127,735	1.26%
5	78,196	0.77%
6	52,255	0.52%
7	37,927	0.37%
8	28,570	0.28%
9	22,678	0.22%
10	18,168	0.18%
* Total indices:	10,131,227	

Figure 4. Embedding vector access pattern.

that the performance degradation of the MLP-dominated RMC3 is smaller since the MLP operation takes more than half of the execution time. To narrow the gap between the SSD version and the DRAM-only version, we need to consider both embedding-dominated models and MLP-dominated models.

*2) **Opportunities for offloading embedding lookup**:* The naïve SSD versions spend the majority of time on SSD accesses and file system I/O stacks. This paper proposes to adopt the in-storage computing approach to cut down the SSD latency. In-storage computing has been shown as an effective approach for I/O traffic reduction. It has been applied to accelerate database application.

(a) Significant read amplification. Fig. 3 shows the read amplification of the SSD-based recommendation system. As the size of available DRAM decreases, the read amplification can be up to dozens of times. The reason is two-fold. First, as the DRAM size reduces, the capacity for caching embedding vectors in the page cache is also limited. The system has to fetch data from SSD with higher frequency. Second, the recommendation system mainly accesses vector granularity data, while conventional SSDs only provide page granularity accesses. The typical size of an embedding vector ranges from 64B to 256B [15], [21], while the flash page size is usually 4KB-32KB. I/O requests smaller than the page size must be aligned to page boundaries, which results in reading an entire page with redundant data from the SSD. Previous work [26] also shows that embedding lookup operations present little spatial locality. The low effective bandwidth is a signal for possible in-storage computing adoption.

(b) Irregular embedding access pattern. The intrinsically irregular access pattern of embedding vectors undermines the effectiveness of page cache. As shown in Fig. 4, only a tiny portion of the data are accessed with high frequency, while others follow near-random access patterns. For this trace, the unique accesses account for 84.74%, while the top 10000 frequently accessed indices account for 59.2% of total accesses. As a result, simply increasing the cache capacity can only marginally improve the performance of recommendation systems.

*3) **Opportunities for offloading MLP operation**:* Compared to the latest work, which offloads only embedding lookup into SSD for addressing embedding-dominated models, accelerating MLP is critical for MLP-dominated models. We propose to also offload the MLP layers into the SSD with FPGA processing for the following reasons. First, FPGA is an order faster than the CPU when running the same MLP layer. This reduces the latency of MLP-dominated models significantly. Second, the improvement of recommendation systems' throughput can be explored with pipelining. The architecture of the recommendation system is suitable to be remapped to pipeline stages for throughput improvement.

While the host and SSD co-design pipeline mechanism is also a possibility, pipelining all stages in storage is more efficient. First, the whole system synchronization usually accompanies high pipe overhead, which is much more severe for server environments. Second, the additional data copying and reformatting would bring additional latency overhead. Third, optimization on embedding (in SSD) and MLP (host-side) separately would be hindered by the long inner concatenation between embedding and MLP layers. For solutions with host-side FPGA, the additional round-trip data communication (hundreds of milliseconds) and synchronization overheads could defeat the benefits. Hence, this paper proposes to offload the entire recommendation system into SSD.

However, in-storage computing is more sensitive to resource consumption and energy efficiency than near-memory acceleration. On the one hand, the additional computing resource deployed in SSD cannot be as powerful as the near-memory accelerators due to the monetary cost. On the other hand, high power consumption often leads to high temperature, which could be detrimental to SSD lifetime. It is not feasible to directly apply previous research of FPGA-based DNN acceleration to in-storage computing for recommendation systems (ISC-RS). Hence, this paper proposes a complete in-storage computing solution for accelerating the entire offloaded recommendation system based on a

Table I
DEFINITION OF PARAMETERS AND ABBREVIATIONS.

Symbol/Module Name	Definition
L_{bot} , L_{top}	Number of bottom, top MLP layers
II	Initiation interval for MLP layers
D_{width}	DRAM bit-width (bandwidth)
N_{batch}	Batch size supported in single I/O
i, j	Index for bottom, top layers
$T_{\text{emb}}, T_{\text{bot}}$, T_{top}	Time cost of embedding, bottom, top MLP layers
T_{EV}	Read latency of one embedding vector
b_{EV}	Read bandwidth of vector read for flash array
M, N	Number of EV tables and lookups for each table
P_{size}	Page size
$EV_{\text{dim}}, EV_{\text{size}}$	Dimension, Size of single EV vector
N_{fmul}, N_{fadd}	Number of fp32 multiplication and addition unit
R, C	Input and output of one FC layer
FTL	Flash Translation Layer
FMC	Flash Memory Controller
RM Reg.	RM Registers
EV Translator	Embedding Vector Translator
EV-FMC	Embedding Vector Flash Memory Controller
EV Sum	Embedding Vector Sum
MUX	multiplexer
DEMUX	demultiplexer
fadd	floating-point adder

low-end FPGA, which can improve both the latency and throughput performance for the embedding-dominated and MLP-dominated models.

IV. RM-SSD DESIGN

In this section, we present RM-SSD, an in-storage computing system based on FPGA for accelerating SSD-based recommendation inference. We first present a system overview for the end-to-end RM-SSD. Second, we present the detailed design of the Embedding Lookup Engine, which focuses on the access latency reduction for embedding vectors. Then, we introduce the MLP Acceleration Engine inside SSD, which aims at optimizing the MLP-dominated models. Finally, we introduce the software integration component.

Notations for all parameters and abbreviations are listed in Table I.

A. System Overview

The NVMe SSD usually consists of flash memory channels and multifunctional SSD controllers. As illustrated in Fig. 5, the RM-SSD controllers are composed of the conventional components (yellow blocks), **Embedding Lookup Engine** (orange blocks) and **MLP Acceleration Engine** (green blocks).

Conventional NVMe SSD controllers are mainly made up of three modules: *NVMe Controller*, Flash Translation Layer (*FTL*), and Flash Memory Controllers (*FMC*s). The NVMe Controller handles the communication between the host and SSD following NVMe standard specifications. FTL translates logical block addresses (LBAs) in the host space to

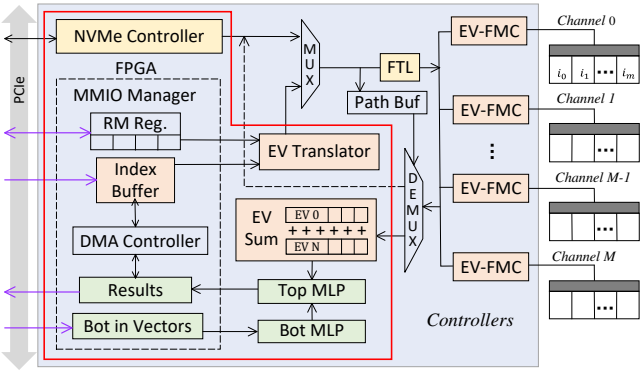


Figure 5. RM-SSD architecture.

physical block addresses (PBAs) in the flash memory space according to certain mapping strategies. Each FMC manages the data transfer between the corresponding flash channel and the outside controllers, and they work independently.

MMIO Manager is a component in RM-SSD for addressing the I/O stack issue mentioned in Section III-B. It serves for both the Embedding Lookup Engine and MLP Acceleration Engine. The proposed RM-SSD supports both block I/O requests and recommendation inference, which are handled by the NVMe Controller and MMIO Manager separately. The MMIO Manager supports both host-side memory interface and DMA mode data transmission. The host-side memory interface provides direct access to the RM Registers (*RM Reg.*), which are used to exchange small-sized control parameters with low latency, e.g., the number of lookups. The DMA transmission is used to transfer data blocks in bulk, including the lookup indices of embedding tables and bottom MLP inputs.

When a recommendation inference is needed, both engines in charge of embedding lookup and MLP are triggered. On one side, the metadata and indices are consumed by the Embedding Lookup Engine through the MMIO interface, and the corresponding sum results will be fed to Top MLP as partial input. On the other side, the input vectors of bottom MLP are broadcast to the Bottom MLP unit via the MMIO manager. The results will also be sent to Top MLP and concatenated with the results of embedding lookups as the full input for Top MLP. When the Top MLP finishes, the status register in RM Registers is converted to ready. At the host side, before reading the results from RM-SSD, the CPU first checks the result status register in SSD using MMIO. Once it turns ready, the final results of inference are transmitted to the host.

B. Embedding Lookup Engine

The Embedding Lookup Engine is composed of the following modules: Embedding Vector Translator (*EV Translator*), Embedding Vector Flash Memory Controller (*EV-FMC*), and Embedding Vector Sum (*EV Sum*).

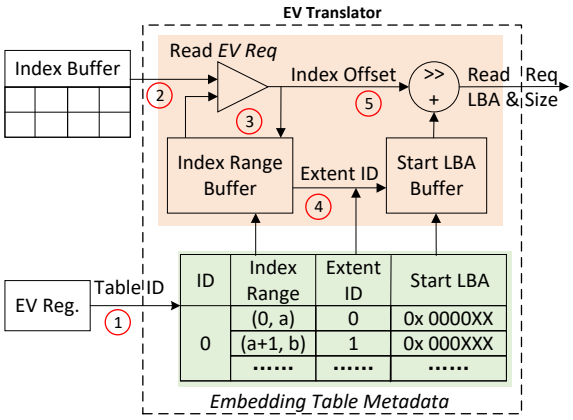


Figure 6. Embedding Vector Translator design.

To address the read amplification mentioned in Section III-B, we adopt a two-stage vector-grained read strategy, including the device stage and flash channel stage. At the device stage, EV Translator translates embedding lookup indices, and EV Sum aggregates the scattered qualified embedding lookup results returned from EV-FMC, which is similar to the *SparseLengthSum* operator in the host framework. At the flash channel stage, EV-FMC only fetches the desired embedding vectors from flash channels.

1) ***Embedding Vector Translator****:* EV Translator is designed to parse the embedding vector index to LBAs. As shown in Fig. 6, it consists of embedding table metadata and Read EV Req. After creating embedding tables, the host side invokes a system call to get the file LBA information of each table, and then sends the start LBA and the length of all file extents (blocks) to the RM-SSD, through RM Registers. Since the dimension of each embedding vector is fixed, the embedding vector index range of each extent can be calculated. We keep the index ranges and the start LBA of all extents with table ID in the DRAM on the FPGA chip inside SSD for instant embedding vector lookup.

When a batch of embedding vector lookups arrives, the host side informs the EV Translator of the number of lookups and sends the indices batch to the Index Buffer in DMA mode. RM-SSD translates the embedding vector request according to the following steps as described in Fig. 6: (1) Once the number of lookups is received, the EV Translator will scan the metadata of each table for faster calculation in (3); (2) Read EV Req fetches one index from the Index Buffer each time; (3) The extent ID of the current index is calculated by checking the index ranges in parallel; (4) According to the extent ID, the start LBA of the embedding vector is determined; (5) Based on the index offset in this extent and the dimension of embedding vector EV_{dim}, we obtain the final LBA of the embedding vector. To achieve vector-grained reading, the size of read request is set to the EV_{size}, $EV_{\text{size}} = EV_{\text{dim}} \cdot \text{sizeof}(float)$.

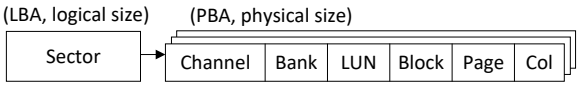


Figure 7. Format conversion from LBA to PBA.

2) ***Vector-Grained Flash Memory Controller****:* As illustrated in Fig. 7, the LBA with logical size generated by EV Translator is transformed to the PBA with physical size after going through the FTL, and the read request size is set to the EV_{size}. To fully utilize the parallelism of flash channels, we propose to stripe the embedding vector read requests over all flash channels and dies.

After FTL translates the logical addresses, the physical block read requests are assigned to the corresponding EV-FMC. Since FTL is shared with conventional block I/O operations, we add a multiplexer (*MUX*) based on round-robin scheduling to serve data requests. The read request path is also marked and then recorded in the read request *Path Buffer* for merging return data.

The PBA is organized in accordance with the multi-level hierarchy of flash arrays, from the lowest-level page to the highest-level channel. *Col* is recognized as the read offset in a page. The typical size of a page is from 4KB to 32KB. When performing a single page read, the whole page is flushed from the flash cell array to the flash page buffer (marked in gray in Fig. 5). After that, the buffered data are transferred to the outside controller at the speed of one byte per cycle. Instead of a whole page, only one vector data from the offset will be transferred, and the size is configured to EV_{size} when performing the embedding vector read request. We can drop the remaining data in this page due to the overall poor locality of the embedding workloads, which is analyzed in Section III-B2. The data transfer of vector read is reduced to $EV_{\text{size}}/P_{\text{size}}$ of the transfer time of page read. It is worth noting that, though flash arrays have a deep hierarchy of storage, all in/out data share one bus for each channel. In other words, vector-grained reads not only reduce the access latency of a single embedding vector but also increase the throughput of reading bulk embedding vectors.

3) ***Embedding Vector Sum****:* All returned data from flash channels are checked by the demultiplexer (*DEMUX*), which plays the opposite role of MUX. After fetching a request flag from Path Buffer, DEMUX checks if the returned data is the response of normal block I/O requests or embedding vector reading requests. If it belongs to the former, the processing flow remains unchanged as the original SSD handling: collecting one full page or multiple pages and then sending to the NVMe Controller.

The request for reading embedding vector is served by EV Sum. Once a single embedding vector is parsed by DEMUX, the embedding vector values will be accumulated by the floating-point adders (*fadd*) on FPGA. Each dimension of the embedding vector is independent of others, and the

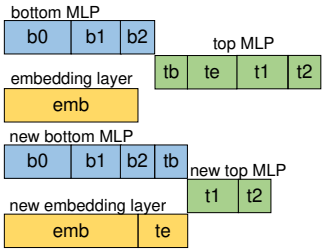


Figure 8. Optimization of ISC-RS by decomposing L_0 of top MLP.

accumulation time can be reduced significantly by utilizing the parallelism of FPGA. Once the number of accumulation equals to the number of lookups in each table, the result will be sent to the MLP Acceleration Engine as input vectors for Top MLP unit. Each embedding table can obtain one result vector, and the size of the united input vector of Top MLP is $EV_{\text{dim}} \cdot M$.

C. MLP Acceleration Engine

We exploit the characteristics of the recommendation model to achieve a resource-efficient design for the MLP Acceleration Engine. First, the long inner concatenation between embedding and MLP layers, which cannot be ignored in the DRAM-only version, can be eliminated by *Intra-Layer Decomposition*. Second, recommendation models usually consist of a series of small FC layers plus a large FC layer (for large recommendation systems). It is possible to fit all the layers on FPGA simultaneously so that the latency and throughput can be further improved by *Inter-Layer Composition*. Third, the unbalanced structure provides opportunities for balancing the time cost of each FC layer, which ensures no waste of resources for unnecessary computing. In addition, the non-MLP embedding can also be balanced with MLP layers with batch processing to maximize throughput with *Kernel Search Algorithm*.

Details of the proposed recommendation system-oriented MLP Acceleration Engine are as follows.

1) ***Basic FC layer design****:* One classical implementation for Matrix Multiplication (MM) is the systolic array, which takes MAC as the basic Processing Element (PE) [27]–[29]. This structured topology can be easily routed for large-scale MM. Given an FC layer with R inputs and C outputs, the time cost is $R \cdot C \cdot II$. It works for embedding-dominated ISC-RS, but for MLP-dominated ISC-RS, this design may not be efficient.

We propose to apply the adder tree [30] to implement the sum operation in the kernel block MM. Assuming the kernel block sizes are kr and kc along the row and column dimension, respectively, the time cost is reduced to $\frac{RC}{kr}II + \log_2 krII \approx \frac{RC}{kr}II$ with resource consumption $kr \cdot (N_{fmul} + N_{fadd})$. It is worth noting that the recommendation model is much more sensitive to accuracy than other DNN models [11], [12]. Therefore, we still keep the MLP weights and embedding vectors in `FP32` precision without any quantization.

Another kernel dimension kc indicates the number of parallel PEs. Taking it into consideration, the time cost is reduced to $\frac{RC}{krkc}II$ with resource consumption $krkc \cdot (N_{fmul} + N_{fadd})$. To further reduce the resource consumption, we leverage the II cycles to pipeline the kc unit with one cycle, so that the $fadd$ and $fmul$ can be reused. Resource consumption is also reduced to $\frac{krkc}{II} \cdot (N_{fmul} + N_{fadd})$.

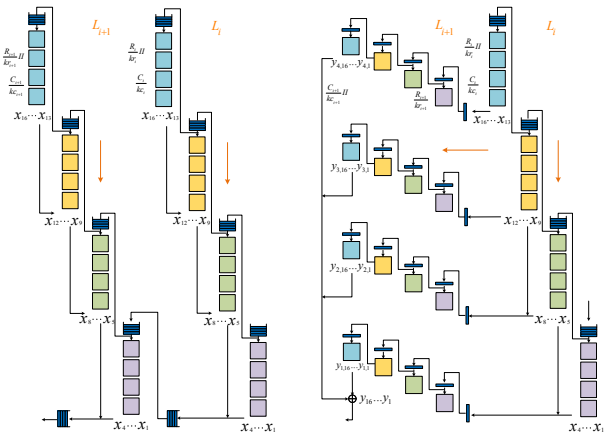
2) ***Intra-layer decomposition****:* In the recommendation model, there is a feature interaction operation (concatenation) after the bottom MLP and embedding layer, as shown in Fig. 1. The current recommendation system handles the concatenation after both of the bottom MLP and embedding layers finish and then feeds the united vector into the top MLP layer. It is suitable for the python framework, but it does not take full advantage of the FPGA hardware.

The weights of the concatenation mapping to the first FC layer (L_0) of top MLP are determined, so that RC can be decomposed into $R_bC + R_eC$. The bottom MLP and embedding layer can continue handling in parallel, until the final sum of their corresponding results. This decomposition makes sure that the performance of L_0 would not be hindered by either embedding layer or bottom MLP. Fig. 8 illustrates the model topology of RM-SSD.

3) ***Inter-layer composition****:* Due to the relatively small MLP model, it is possible to fit all layers of the recommendation model on FPGA. However, if all FC layers adopt the same scan pattern for kernel streaming, there would be a pipeline stall. Assuming all inputs stream along the column dimension with kc first, then repeat along the row dimension with stripe kr. As a result, only after all columns of L_i finish accumulation, the first kc columns of L_{i+1} can feed the first input vector. After that, the input stripes the left columns until the end as shown in Fig. 9(a). The latency of each ISC-RS would be the sum of the time cost on each layer, which is not efficient for pipelining.

We propose to compose the adjacent layers into a pair by exchanging the scanning direction of each layer alternatively in Fig. 9(b). Under this circumstance, L_{i+1} can perform MM by row scanning, while L_i is handled by column scanning, so that the time consumption of MLP can be reduced by half. We can get the time cost for embedding, bottom MLP and top MLP as follows[1].

[1]We use emb′, bot′ and top′ to represent the extended embedding layer, bottom MLP and shortened top MLP respectively. Index for new top MLP starts from 1.



(a) (b)

Figure 9. Optimization of ISC-RS by composing adjacent layers. (a) apply the same scan pattern. (b) exchange the scan pattern alternatively.

$$T_{\text{emb}'} = \max\left(N_{\text{batch}}\frac{MN}{b_{\text{EV}}}, \frac{R_e C_e}{kr_e kc_e} II\right) \tag{1a}$$

$$T_{\text{bot}'} = \sum_{i=0,2,\ldots}^{L_{\text{bot}'}} \max\left(\frac{R_i C_i}{kr_i kc_i}, \frac{R_{i+1} C_{i+1}}{kr_{i+1} kc_{i+1}}\right) II \tag{1b}$$

$$T_{\text{top}'} = \sum_{j=1,3,\ldots}^{L_{\text{top}'}} \max\left(\frac{R_j C_j}{kr_j kc_j}, \frac{R_{j+1} C_{j+1}}{kr_{j+1} kc_{j+1}}\right) II \tag{1c}$$

*4) **Kernel search algorithm**:* The kernel function Matrix Multiplication is critical for the resource utilization in the proposed RM-SSD. We propose a kernel search algorithm that can efficiently decide the kernel size for each FC layer, which ensures the lowest resource utilization and the optimal throughput for both embedding-dominated and MLP-dominated models.

Given the time cost for embedding layer, bottom MLP and top MLP as listed in equation 1, the optimization objective is to meet the goal in equation 2 by picking the suitable $K_{\text{bot}'} = \{(kr_0, kc_0), (kr_1, kc_1), ..., (kr_i, kc_i)\}$ and $K_{\text{top}'} = \{(kr_1, kc_1), ..., (kr_j, kc_j)\}$ pairs.

$$\begin{cases} T_{\text{bot}'} \leq T_{\text{emb}'} \\ T_{\text{top}'} \leq T_{\text{emb}'} \\ \arg\min\limits_{K_{\text{bot}'}, K_{\text{top}'}} \left(\sum\limits_{i}^{L_{\text{bot}'}} kr_i kc_i + \sum\limits_{j}^{L_{\text{top}'}} kr_j kc_j + kr_e kc_e\right) \end{cases} \tag{2}$$

The following rules are applied to achieve this goal.

Rule One: BRAM resource assessment. Fitting all weights on the BRAM is the preferred option. However, if

$$\sum_{i}^{L_{\text{bot}'}} B_i + \sum_{j}^{L_{\text{top}'}} B_j + B_e > B_{\max}$$

the off-chip DRAM will be used.

Rule Two: Kernel size for DRAM employed layers. If DRAM is employed for a specific FC layer, the bottleneck of this layer would be the DRAM read bandwidth. Weight parameters fetched from DRAM are held by FPGA until the result is calculated by the kernel MM. Double Buffering is used to avoid reading data stalls. When buffer A is held by MM computations for II_B, buffer B is also used to read data for size II_D. To fully utilize the DRAM bandwidth, $II_D \geq II_B$ must be satisfied. To minimize the resource consumption, II_D is set to II_B, and thus the time cost for this layer is

$$\frac{R_i C_i}{kr_i kc_i} II = \begin{cases} \frac{R_i C_i}{D_{\text{width}}\, II_D} \cdot II_D = \frac{R_i C_i}{D_{\text{width}}}, & II_D \geq II_B \\ \frac{R_i C_i}{D_{\text{width}}\, II_D} \cdot II_B > \frac{R_i C_i}{D_{\text{width}}}, & II_D < II_B \end{cases}$$

$kr_i = D_{\text{width}}$ and $kc_i = II_B = II_D$ is set for the DRAM employed layer.

Rule Three: Batch size decision. $N_{\text{batch}=1}$ is the default setting. However, if

$$kr_i = kc_i = 2^{K_{\max}}, \forall i \in [0, L_{\text{bot}'})$$

$$kr_j = kc_j = 2^{K_{\max}}, \forall j \in [1, L_{\text{top}'})$$

$$kr_i, kc_i, kr_j, kc_j \in 2^K, 0 \leq K \leq K_{\max}$$

$T_{\text{bot}'}^{max} > T_{\text{emb}'}$ or $T_{\text{top}'}^{max} > T_{\text{emb}'}$, we increase batch size, $N_{\text{batch}} = 1, 2, 4, ...$, until both $T_{\text{bot}'}^{max} \leq T_{\text{emb}'}$ and $T_{\text{top}'}^{max} \leq T_{\text{emb}'}$ are both satisfied.

Rule Four: $K_{\text{bot}'}, K_{\text{top}'}$ decision. Large kr, kc pair is picked first and reduced to approaching the limit. To avoid layer pipeline bubbles, the following constraints must be met[2].

$$\begin{cases} kc_i \geq kr_{i+1} \\ kc_e = kc_b \geq kr_{j=1} \end{cases} \tag{3}$$

Considering the aforementioned kernel reuse pipeline in IV-C1, the kernel size minimum value requirements, except for the last layer.

$$R_i C_i \geq II, i \neq L_{\text{top}'} - 1 \tag{4}$$

Consider equation 1, the time cost of adjacent layers should be balanced.

$$\frac{R_i C_i}{kr_i kc_i} \approx \frac{R_{i+1} C_{i+1}}{kr_{i+1} kc_{i+1}} \tag{5}$$

In conclusion, with the constraints equation 3, 4, 5, the MLP layers in RM-SSD are designed to achieve the targets in equation 2, by leveraging equation 1.

[2]Here i indicates both bottom and top MLP layers.

D. Software Integration

The RM-SSD driver and user library are designed and optimized to take full advantage of the RM-SSD. We provide a C++ runtime library, which can be easily integrated with Python-based deep learning frameworks, e.g., PyTorch, Caffe2, using Cython. The following interfaces are provided.

- `RM_create_table(TableSize)` adopts the block I/O driver and goes through the file system as normal files. For security, the host first retrieves the access permission of the RM-SSD. Only when the user is qualified, the embedding tables can be written to the RM-SSD via NVMe. After the creation phase is finished, the owner and other file system related information are generated and persisted in the RM-SSD.
- `RM_open_table(TableID, TablePath)` is the open operation residing on MMIO path similar to the open on file system path (a one-time operation). The file system will first retrieve related information from RM-SSD (recorded in the creating phase) and then check if the user is authorized. After that, the host side invokes a system call to get the file LBA information of the table and send the start LBA and the length of all extents to the RM-SSD as mentioned in Section IV-B. This function will return a file descriptor (fd), which will be considered as the authentication in the phase of the reading output.
- `RM_send_inputs(fd, IndicesPerLookup, SparseIn, DenseIn)` is used to send the input parameters of each recommendation inference. The validity of fd will be verified first. If it is qualified, the IndicesPerLookup are sent to the RM-SSD through host-side MMIO, and the SparseIn and DenseIn arrays, which indicate the indices of embedding lookup and MLP input separately, are transmitted to the RM-SSD in DMA mode.
- `RM_read_outputs()` reads the final results in the form of the batch from the RM-SSD in DMA mode.

Optimization of system-level throughput. RM-SSD does not directly support large batch processing in a single inference. If large batch inferences come, they should be partitioned into several small batches, which can be supported by RM-SSD. To fully utilize the pipeline modules of the FPGA-based RM-SSD, we leverage the system-level pipeline mechanism. Before reading the results of current small batches, we pre-send the input parameters of the next small batches. The system throughput can benefit from two perspectives. First, the overhead of sending parameters can be hidden by the simultaneous RM-SSD processing. Second, the in-storage computing modules can keep processing without waiting.

V. Implementation

Commercial SSDs are often not programmable by users. We emulate the proposed RM-SSD by taking advantage of the Xilinx Virtex 57 Plus UltraScale XCVU9P card, which is attached to an Amazon EC2 F1 instance [31] through PCIe gen3 × 16. This card is mainly composed of an FPGA chip and 64GB (16GB × 4) off-chip DDR4 with 64-byte data width (D_{width}).

Table II
Performance and settings of the emulated SSD.

Capacity	32 GB
#Channels	4
Random 4K Read	45K IOPS
Latency T_{page}	20 us
Page read delay C_{page}	4000 cycles
EV read delay C_{EV}	$(0.293EV_{\text{size}} + 2800)$ cycles

Table III
Architectural features of DLRM models.

Model	Bottom MLP	Top MLP	DIM	Tables	Lookups	MLP size
RMC1	128-64-32	256-64-1	32	8	80	0.39MB
RMC2	256-128-64	128-64-1	64	32	120	1.23MB
RMC3	2560-1024-256-32	512-256-1	32	10	20	12.23MB

A. Parameter Settings

We implement the functionality of RM-SSD Controllers in FPGA and employ four DDR4 banks to emulate four flash channels. We modify the NVMe driver developed by the Insider framework [32] to imitate the working mechanism of the real NVMe protocol. The linear mapping function is applied in the FTL design, and each page data are scattered around the four DDR4 chips for higher throughput.

The main read latency of an SSD comes from the flash channel T_{page}. We add a delay before each DDR4 in the FPGA in order to emulate the read latency. The FPGA runs at 200MHz (5ns). T_{page} can be divided into flash buffer flush T_{flush} and data transfer T_{trans}. The ratio of T_{flush} and T_{trans} is normally around 7:3 for reading a single page [3]. By applying the second-stage vector-grained reading strategy in Section IV-B2, T_{ev} is reduced to $\frac{EV_{\text{size}}}{P_{\text{size}}} \cdot T_{\text{trans}} + T_{\text{flush}}$. By replacing $T_{\text{fpga}} = 5\text{ns}$, $T_{\text{trans}} = 0.3T_{\text{page}}$ and $T_{\text{flush}} = 0.7T_{\text{page}}$, we get the final delay cycles $C_{\text{EV}} = \frac{T_{\text{EV}}}{T_{\text{fpga}}} = \lceil (60\frac{EV_{\text{size}}}{P_{\text{size}}} + 140)T_{\text{page}}(us) \rceil$ and $C_{\text{Page}} = \frac{T_{\text{page}}}{T_{\text{fpga}}} = 200T_{\text{page}}(us)$.

B. RM-SSD Evaluation

As mentioned in Section III-B, the performance of RM-SSD is related to the SSD random read performance. In RM-SSD, the page size is set to a minimum of 4KB. To align with a real SSD as much as possible, we first use `FIO` tool [33] to test the performance of a real NVMe SSD also on the AWS F1 instance. Then, we adjust the emulated SSD to achieve similar performance, including the page reading delay cycles C_{Page} and 4K random read speed. The final

[3]The ratio is from an industry partner

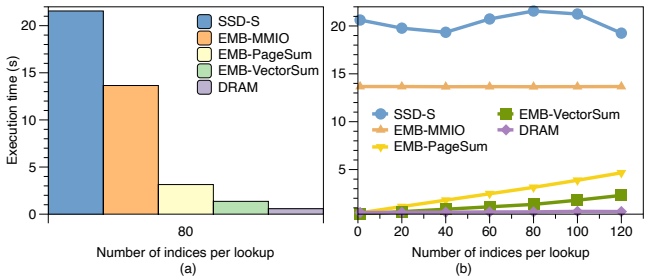


Figure 10. Performance of SLS operator with different implementations. (a) Execution time. (b) Sensitively to lookups.

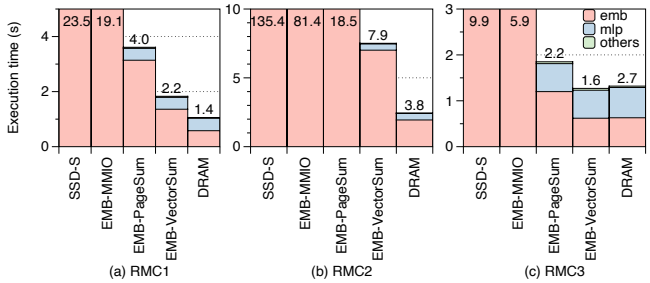


Figure 11. End-to-end performance of different SSD-based recommendation systems.

settings and performance of our emulated SSD are presented in Table II[4].

VI. Evaluation

A. Experimental Setup

We evaluate the RM-SSD on AWS EC2 F1 instance containing Intel Xeon E5-2686 v4 @ 2.30GHz CPU (8 vCPU), 122 GB DRAM (8 channels), and a Xilinx Virtex 57 Plus UltraScale XCVU9P card as described in Section V. We run benchmarks on Facebook's DLRM models as listed in Table III, which contains both embedding-dominated and MLP-dominated models [6], [20], [34]. For each set of DLRM model execution, we measure the elapsed time of 1k inferences in a steady state. The total size of embedding tables for each model is set to 30 GB. We use the same input trace as in Section III.

Besides RM-SSD, we have also implemented other SSD-based recommendation systems. Following Section III-B, all of them leverage the customized C++ *SparseLengthSum* operator with different embedding lookup strategies. **SSD-S** is the baseline version, which runs with limited DRAM size (1/4 size of total embedding tables), and embedding lookup is realized through fileIO. **EMB-MMIO** indicates that all embedding vector related pages are fetched to the userspace directly through MMIO with the granularity of page size and then sum operations performed by the host CPU. **EMB-PageSum** means that all embedding vector related pages are also read from flash channels, but sum operations are performed inside the SSD. **EMB-VectorSum** represents RM-SSD running with Embedding Lookup Engine only.

B. Evaluation on Embedding Lookup Engine

This section demonstrates the performance improvement brought by Embedding Lookup Engine. We first investigate the performance of Embedding Lookup Engine in a standalone SLS operator and then merge them with different DLRM models for end-to-end performance.

[4]In the MLP design, the DRAM is also used to save weights for MLP-dominated models. Only half DRAM on the FPGA board is used to emulate flash channels.

SLS operator performance. First, we compared EMB-VectorSum with SSD-S, EMB-MMIO, EMB-PageSum based on the DLRM-RMC1 configuration. As shown in Fig. 10(a), EMB-VectorSum outperforms the baseline SSD-S by 16×. The reason is three-fold. First, EMB-VectorSum bypasses most parts of the I/O stack and page cache, and allows direct access to SSD through the MMIO interface. As we can see from EMB-MMIO, the time consumption is reduced to 60%. Second, EMB-VectorSum takes full utilization of the internal parallelism of the flash arrays, which can be verified by the EMB-PageSum. Third, RM-SSD supports vector-grained access. Compared with EMB-PageSum, the read amplification of EMB-VectorSum is reduced further in the perspective of flash arrays due to higher random read bandwidth.

Then, by varying the number of lookups, we studied how the EMB-VectorSum is sensitive to the number of lookups of each table. In Fig. 10(b), the execution time increases linearly as lookups scale up. We also studied the sensitivity of table numbers and got the same trend. Actually, the performance is affected by the product of tables and lookups. It is reasonable as the flash arrays are not aware of tables.

End-to-End performance. Fig. 11 shows the end-to-end performance of all above mentioned SLS operators in recommendation inference. We present the breakdown of the embedding layer and MLP layers. The performance result is similar to the standalone SLS operators. Compared to SSD-S, EMB-VectorSum achieves up to 17× speedup. It even outperforms the ideal DRAM-only performance in RMC3 model with much lower memory capacity. When we look into the time breakdown, in RMC3 model evaluation, the MLP layers have become the dominating bottleneck for both of EMB-VectorSum. Moreover, recall the recommendation model architecture, the time consumption of MLP layers can be hidden for further throughput improvement by fully utilizing the pipeline mechanism on FPGA.

C. Evaluation of Full RM-SSD

In this section, we present the performance of full RM-SSD with both embedding lookup engine and MLP acceleration. We compare RM-SSD with SSD-S, EMB-VectorSum,

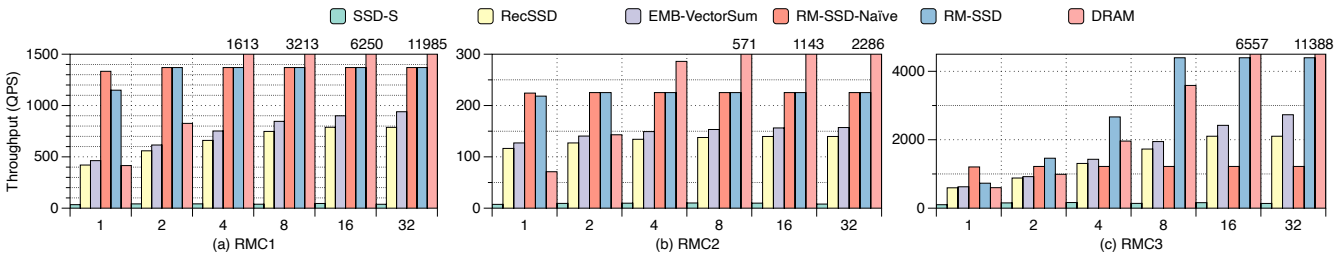


Figure 12. Throughput of different implementations of recommendation inference.

Table IV
I/O TRAFFIC REDUCTION OF DIFFERENT REALIZATIONS.

Model	RecSSD	EMB-VectorSum	RM-SSD
RMC1	1989	1989	31826
RMC2	1071	1071	137142
RMC3	546	546	10914

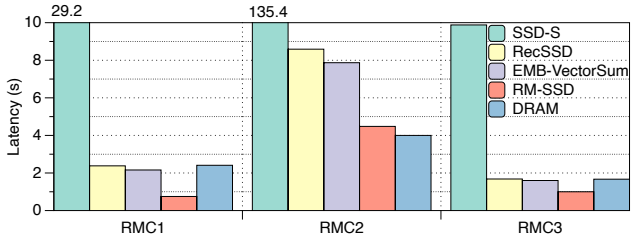


Figure 13. Latency of different implementations of recommendation inference.

RecSSD [20] and DRAM-only across all DLRM models.

RecSSD is the latest ISC work for accelerating SSD-based recommendation inference, which is realized on an ARM-based OpenSSD platform [35]. It offloads embedding lookup into SSD and leverages the host- and SSD-side cache to enhance performance. We follow RecSSD's design and implement it on our emulated SSD. The host-side cache is applied directly from their open-source library. The SSD-side cache cannot be supported by the emulated SSD, but this loss can be compensated by the performance gain from ARM to FPGA. The time consumption of embedding vector extraction and sum can be ignored for FPGA handling compared with embedding vector reading, while that occupies half of the total time in ARM-based RecSSD. Their evaluation also proves that SSD-side cache only brings marginal benefits. Actually, RecSSD in our evaluation achieves better performance than the original one.

Reduction of I/O traffic. Table IV shows read I/O traffic reduction of various SSD-based recommendation systems, compared with the baseline SSD-S. The I/O traffic of all optimized implementations decreased significantly (refer to Fig. 3). As a result of ISC, not only the read amplifications are all eliminated, but also the I/O traffic is reduced by the offloaded computations. RM-SSD requires close to zero I/O traffic, where all recommendation inferences are handled in the SSD. For batch size 1, it only reads 64 bytes (MMIO data-width) returned. Noting that, although EMB-VectorSum and RecSSD achieve almost the same I/O traffic, the content of I/O is different. For EMB-VectorSum, the return value is the final sum result. While for RecSSD, the return value indicates partial sum and would be merged with the host cache embedding vectors later.

RM-SSD performance. Fig. 12 and Fig. 13 show that RM-SSD achieves up to 36× throughput improvement and 97% latency reduction compared to the baseline SSD-S. In addition to the read amplification elimination and file system bypassing, it also benefits from the newly offloaded MLP acceleration. Since we design the MMIO interface to transfer input parameters and output results, the time overhead is negligible with only less than tens of microseconds (less than 1%) for each inference.

Compared with EMB-VectorSum, the latency is reduced by 42%-65%. The throughput improvement compared to RS-SSD is around 1.5-2×. It gains from the pipeline remapping on the FPGA board, with coarse-grained bottom and top MLP pipelining and fine-grained inter-layer and intra-layer optimization for ISC-RS.

As the batch size increases, RM-SSD exhibits different performance variations for different models. For embedding-dominated models RMC1 and RMC2, the throughput almost keeps the same value, which can also be considered as the throughput of embedding layers. For MLP-dominated RMC3, the throughput increases linearly with the batch size, from 1 to 4. When the batch size is 1 and 2, it is still MLP-dominated, and the throughput is constrained by the largest layer of MLP. However, when the batch size equals 4, the time of embedding vector lookup exceeds the MLP layers. At this stage, the MLP-dominated model has been converted to the embedding-dominated model. As the batch size increases further, the throughput maintains the maximum value.

RM-SSD can cut down the latency by up to 64% and achieve up to 2.6× throughput improvement compared with RecSSD. The reason is two-fold. First, for the embedding lookup module, we adopt the vector-grained access of embeddings, while RecSSD still adopts the page access.

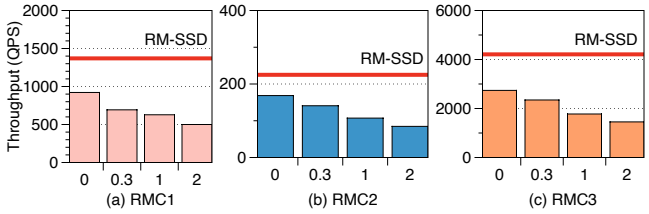


Figure 14. Performance comparison between RM-SSD (red-line) and RecSSD with four locality cases on input traces inspired by [20]. K=0,1,2 indicate locality distribution with 80%, 45%, and 30% hit ratio respectively. The locality of default synthetic input trace is 65% with K=0.3.

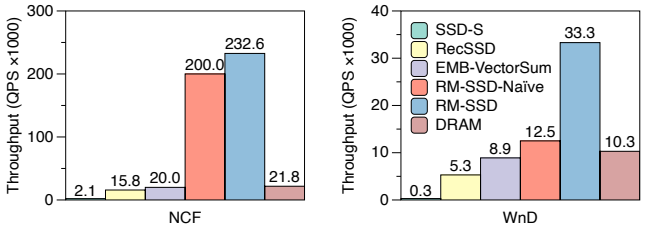


Figure 15. Performance of NCF and WND model.

After detailed analysis, we found that the implementation of RecSSD is similar to EMB-PageSum plus a userspace cache, and the performance gap between EMB-PageSum and EMB-VectorSum can be found in Section VI-B. Secondly, we take advantage of the MLP Acceleration Engine, which benefits from the pipeline mechanism and parallel PEs of the FPGA.

The performance improvement brought by the host-side cache of RecSSD cannot compete with the direct MLP acceleration. Moreover, it is extremely sensitive to the locality of workloads. As shown in Fig. 14, when decreasing the locality of input traces, the throughput of RecSSD degrades accordingly, while RM-SSD maintains the same throughput. Besides, the host-side cache of RecSSD is statically partitioned based on history input, which is not robust in the online running scenario.

More MLP-dominated models. To evaluate the extreme case of MLP-dominated models, we evaluate Neural Collaborative Filtering (NCF) [8] and Wide & Deep (WnD) [10], which only perform embedding lookup once for each embedding table. Fig. 15 demonstrates that RM-SSD is more effective for MLP-dominated models. It outperforms the baseline SSD-S by around 100×. Compared with RecSSD, the speedup of 6-15× confirms the importance of MLP acceleration. It even achieves better performance than the all-DRAM version. This is because, for these two models, the predominant MLP layers in DRAM can be accelerated by the SSD-side FPGA.

Table V
KERNEL SIZE OF EACH LAYER.

RMC	L_{b0}	L_{b1}	L_{b2}	L_b	L_e	L_{t1}	L_{t2}
1,2	4×2	2×4	–	4×2	4×2	2×4	4
3	16×8	8×2	2×4	4×2	4×2	2×4	4

Table VI
RESOURCE CONSUMPTION OF MLP ACCELERATION ENGINE.

Model	Unit	LUT	FF	BRAM	DSP
RMC1,2	MLP-naïve	154541	59032	237	612
	MLP	159338	60672	194	604
	MLP-op	19064	8294	85	41
RMC3	MLP-naïve	219671	82676	246.5	612
	MLP	284120	96598	320	928
	MLP-op	131720	49277	221.5	366
XCVU9P		1181768	2363536	2160	6840
XC7A200T		**215360**	**269200**	**365**	**740**

D. Resource Efficiency of MLP Acceleration Engine

In this subsection, we show the effectiveness of the proposed kernel search algorithm in Section IV-C4 for optimizing resource consumption FPGA. The II for kernel computing IV-C is 8. The default kernel size of each layer in RMC1 and RMC2 is 16×16, while that of RMC3 is 8×8, except for the first bottom layer with 16×8. When applying the kernel search algorithm for DLRM models in Table III, the kernel size of each layer can be revised to the value in Table V. Thanks to the intrinsic constraints of embedding access, the default and optimized kernel setting can achieve the same performance. The kernel size is the key impact factor of resource consumption on FPGA. As shown in Table VI, the resource consumption of default and optimized settings have a significant difference, especially for RMC3. Compared to the conventional MLP design (MLP-naïve), which is adopted by [36] for near-memory acceleration in recommendation systems. Fig. 12 shows that RM-SSD can achieve the same performance with one order of magnitude less resource for RMC1 and RMC2. For MLP-dominated RMC3, RM-SSD achieves 3.5× speedup compared to MLP-naïve with much less resource. Since RM-SSD targets a low-end FPGA in the enterprise SSD [37], e.g. XC7A200T [38], RMC3 cannot work with both default settings and naïve MLP design.

VII. RELATED WORK

Recent researches emphasize the importance of deep learning based personalized recommendation systems by performing detailed performance characterization of recommendation models in data center [6], [34], [39]–[44]. Those analyses demonstrate that recommendation models have unique memory bandwidth and compute requirements for large embedding tables with sparse lookup operations.

Based on this conclusion, some researchers exploit near-memory processing to accelerate recommendation systems. TensorDIMM [45] and RecNMP [26] utilize redesigned DRAM to accelerate embedding lookup. Centaur [36] takes advantage of the chiplet-based accelerator for accelerating both embedding and MLP layer. They apply the GEMM unit for processing MLP layer-by-layer, leading to FC bottleneck in some cases. Ref. [46], MicroRec [47], and FleetRec [48] leverage the high-bandwidth memory (HBM) in the FPGA accelerator, which is PCIe-attached to the host, to address the embedding layers. RecPipe [49] implements an inference scheduler to remap multi-stage recommendation engines onto a custom hardware accelerator.

There exits some work for addressing the insufficient memory in recommendation systems. Mixed dimension embeddings [50], compositional embeddings [51], and saec [52] compress the embedding layer to reduce the memory footprint. They are orthogonal to storage-based recommendation systems. Bandana [15] tries to store cold vectors in the non-volatile memory (NVM) as storage and cache a small number of hot vectors in the DRAM. However, they still access the embedding with coarse-grained 4KB reads, while our proposed RM-SSD leverages the vector-grained granularity to eliminate the read amplification. RecSSD [20] proposes utilizing in-storage computing to accelerate SSD-based recommendation inference. They also utilize the host-side and SSD-side cache to enhance the performance further. They only push down the embeddings into SSD, while our proposed RM-SSD offloads the end-to-end recommendation system and achieves great performance improvement. The detailed comparison can be found in Section VI-C.

VIII. Conclusion

We propose RM-SSD, a dedicated in-storage computing prototype with a low-end FPGA for accelerating large-scale recommendation inference under memory capacity-limited systems. We address both the embedding-dominated and MLP-dominated recommendation models for latency and throughput improvements. By eliminating the read amplifications of embedding lookup and accelerating the recommendation system-oriented MLP layers, RM-SSD can achieve 20-100× throughput improvement compared with the baseline SSD and 1.5-15× improvement compared to the latest work.

Acknowledgement

We sincerely thank anonymous reviewers for their constructive feedback. This work is partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Grant CityU 11217020 and 11218720, and the Facebook Research Grant.

References

[1] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in alibaba," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2018, pp. 839–848.

[2] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for YouTube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*, 2016, pp. 191–198.

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

[4] C. A. Gomez-Uribe and N. Hunt, "The Netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, no. 4, pp. 1–19, 2015.

[5] C. Hansen, C. Hansen, L. Maystre, R. Mehrotra, B. Brost, F. Tomasi, and M. Lalmas, "Contextual and sequential user embeddings for large-scale music recommendation," in *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys)*, 2020, pp. 53–62.

[6] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of Facebook's DNN-based personalized recommendation," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 488–501.

[7] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "WTF: the who to follow service at Twitter," in *Proceedings of the 22nd international conference on World Wide Web (WWW)*, 2013, pp. 505–514.

[8] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *WWW '17 Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017, pp. 173–182.

[9] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2015, pp. 1235–1244.

[10] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS)*, 2016, pp. 7–10.

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

[12] G. Zhou, N. Mou, Q. Pi, Y. Fan, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep interest evolution network for click-through rate prediction," *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, vol. 33, no. 1, pp. 5941–5948, 2019.

[13] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao, "Deep crossing: Web-scale modeling without manually crafted combinatorial features," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2016, pp. 255–262.

[14] S. Zhai, K. hao Chang, R. Zhang, and Z. M. Zhang, "DeepIntent: Learning attentions for online advertising with recurrent neural networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2016, pp. 1295–1304.

[15] A. Eisenman, M. Naumov, D. Gardner, M. Smelyanskiy, S. Pupyrev, K. M. Hazelwood, A. Cidon, and S. Katti, "Bandana: Using non-volatile memory for storing deep learning models," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 1, 2019, pp. 40–52.

[16] C.-J. Wu, R. Burke, E. Chi, J. A. Konstan, J. J. McAuley, Y. Raimond, and H. Zhang, "Developing a recommendation benchmark for MLPerf training and inference," *arXiv preprint arXiv:2003.07336*, 2020.

[17] M. Lui, Y. Yetim, Ö. Özkan, Z. Zhao, S.-Y. Tsai, C.-J. Wu, and M. Hempstead, "Understanding capacity-driven scale-out neural recommendation inference," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 162–171.

[18] W. Zhao, J. Zhang, D. Xie, Y. Qian, R. Jia, and P. Li, "AIBox: Ctr prediction model training on a single node," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, 2019, pp. 319–328.

[19] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li, "Distributed hierarchical gpu parameter server for massive scale deep learning ads systems," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 2, 2020, pp. 412–428.

[20] M. Wilkening, U. Gupta, S. Hsia, C. Trippel, C.-J. Wu, D. Brooks, and G.-Y. Wei, "RecSSD: near data processing for solid state drive based recommendation inference," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 717–729.

[21] M. Kim and S. Lee, "Reducing tail latency of DNN-based recommender systems using in-storage processing," in *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*, 2020, pp. 90–97.

[22] Samsung, "Samsung 980 Pro," https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/980pro/.

[23] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: opportunities and challenges," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2013, pp. 1221–1230.

[24] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. J. Desnoyers, and Y. Solihin, "Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines," in *Proceedings of the 11th USENIX conference on File and Storage Technologies (USENIX FAST)*, 2013, pp. 119–132.

[25] Y.-C. Hu, M. T. Lokhandwala, I. Te, and H.-W. Tseng, "Dynamic multi-resolution data storage," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, pp. 196–210.

[26] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating personalized recommendation with near-memory processing," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[27] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[28] J. Cong and J. Wang, "PolySA: Polyhedral-based systolic array auto-compilation," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[29] J. Wang, L. Guo, and J. Cong, "AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 93–104.

[30] B. Asgari, R. Hadidi, and H. Kim, "MEISSA: Multiplying matrices efficiently in a scalable systolic architecture," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 130–137.

[31] Amazon, "Aws F1," 12 2020. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1

[32] Z. Ruan, T. He, and J. Cong, "INSIDER: Designing in-storage computing system for emerging high-performance drive," in *USENIX Annual Technical Conference (USENIX ATC)*, 2019, pp. 379–394.

[33] J. Axboe, "FIO," http://freshmeat.sourceforge.net/projects/fio/.

[34] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 982–995.

[35] Cosmos-OpenSSD, "OpenSSD." [Online]. Available: https://github.com/Cosmos-OpenSSD/Cosmos-plus-OpenSSD/

[36] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 968–981.

[37] Western Digital Corporation, "EnterpriseSSD," https://shop.westerndigital.com/products/internal-drives/wd-gold-nvme-ssd#WDS768T1D0D.

[38] Xilinx, "XC7A200T," https://www.alibaba.com/product-detail/Xc7a200t-Electronic-Components-XC7A200T-1FFG1156I-Integrated_1600338229321.html.

[39] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in Facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[40] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at Facebook: Understanding inference at the edge," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.

[41] M. Naumov, J. Kim, D. Mudigere, S. Sridharan, X. Wang, W. Zhao, S. Yilmaz, C. Kim, H. Yuen, M. Ozdal, K. Nair, I. Gao, B.-Y. Su, J. Yang, and M. Smelyanskiy, "Deep learning training in Facebook data centers: Design of scale-up and scale-out systems," *arXiv preprint arXiv:2003.09518*, 2020.

[42] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 620–629.

[43] S. Hsia, U. Gupta, M. Wilkening, C.-J. Wu, G.-Y. Wei, and D. Brooks, "Cross-stack workload characterization of deep recommendation systems," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, 2020, pp. 157–168.

[44] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[45] Y. Kwon, Y. Lee, and M. Rhu, "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)*, 2019, pp. 740–753.

[46] Y. Zhu, Z. He, W. Jiang, K. Zeng, J. Zhou, and G. Alonso, "Distributed recommendation inference on FPGA clusters," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 279–285.

[47] W. Jiang, Z. He, S. Zhang, T. B. Preußer, K. Zeng, L. Feng, J. Zhang, T. Liu, Y. Li, J. Zhou, C. Zhang, and G. Alonso, "MicroRec: Accelerating deep recommendation systems to microseconds by hardware and data structure solutions," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 3, 2021, pp. 845–859.

[48] W. Jiang, Z. He, S. Zhang, K. Zeng, L. Feng, J. Zhang, T. Liu, Y. Li, J. Zhou, C. Zhang, and G. Alonso, "FleetRec: Large-scale recommendation inference on hybrid GPU-FPGA clusters," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2021, pp. 3097–3105.

[49] U. Gupta, S. Hsia, J. Zhang, M. Wilkening, J. Pombra, H.-H. S. Lee, G.-Y. Wei, C.-J. Wu, and D. Brooks, "RecPipe: Co-designing models and hardware to jointly optimize recommendation quality and performance," in *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, pp. 870–884.

[50] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2786–2791.

[51] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, "Compositional embeddings using complementary partitions for memory-efficient recommendation systems," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2020, pp. 165–175.

[52] X. Wu, H. Xu, H. Zhang, H. Chen, and J. Wang, "Saec: Similarity-aware embedding compression in recommendation systems," in *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*, 2020, pp. 82–89.