Near-Storage Processing for Solid State Drive Based Recommendation Inference with SmartSSDs®

Mohammadreza Soltaniyeh
m.soltaniyeh@cs.rutgers.edu
Department of Computer Science,
Rutgers University
USA

Veronica Lagrange Moutinho Dos Reis
veronica.l@samsung.com
Samsung Semiconductor, Inc.
USA

Matt Bryson
matt.bryson@samsung.com
Samsung Semiconductor, Inc.
USA

Xuebin Yao
xuebin.yao@samsung.com
Samsung Semiconductor, Inc.
USA

Richard P. Martin
rmartin@scarletmail.rutgers.edu
Department of Computer Science,
Rutgers University
USA

Santosh Nagarakatte
santosh.nagarakatte@cs.rutgers.edu
Department of Computer Science,
Rutgers University
USA

ABSTRACT

Deep learning-based recommendation systems are extensively deployed in numerous internet services, including social media, entertainment services, and search engines, to provide users with the most relevant and personalized content. Production scale deep learning models consist of large embedding tables with billions of parameters. DRAM-based recommendation systems incur a high infrastructure cost and limit the size of the deployed models. Recommendation systems based on solid-state drives (SSDs) are a promising alternative for DRAM-based systems. Systems based on SSDs can offer ample storage required for deep learning models with large embedding tables. This paper proposes SmartRec, an inference engine for deep learning-based recommendation systems that utilizes Samsung SmartSSD®, an SSD with an on-board FPGA that can process data in-situ. We evaluate SmartRec with state-of-the-art recommendation models from Facebook and compare its performance and energy efficiency to a DRAM-based system on a CPU. We show SmartRec improves the energy efficiency of the recommendation inference task up to 10× in comparison to the baseline CPU implementation. In addition, we propose a novel application-specific caching system for SmartSSDs® that allows the kernel on the FPGA to use its DRAM as a cache to minimize high latency SSD accesses. Finally, we demonstrate the scalability of our design by offloading the computation to multiple SmartSSDs® to further improve performance.

CCS CONCEPTS

• **Computer systems organization** → **Architecture**; *Neural networks*; *Reconfigurable computing*.

KEYWORDS

Recommendation Systems, Deep learning, FPGA, SmartSSD®, Near-Storage Computation

ACM Reference Format:
Mohammadreza Soltaniyeh, Veronica Lagrange Moutinho Dos Reis, Matt Bryson, Xuebin Yao, Richard P. Martin, and Santosh Nagarakatte. 2022. Near-Storage Processing for Solid State Drive Based Recommendation Inference with SmartSSDs®. In *Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3489525.3511672

1 INTRODUCTION

Personalized recommendation systems are widely deployed in a variety of data center applications, including social media [6], entertainment services [27], and search engines [22] to improve users' experience. Deep learning-based solutions offer high accuracy and scalable solutions for these recommendation systems. Unsurprisingly, AI-driven recommendation models account for a significant portion of the cycles in the data centers [6, 29].

Neural recommendation models capture both dense and categorical features to achieve higher prediction accuracies. The categorical features (sparse features) are the key components of a neural recommendations model that distinguishes them from other types of neural models, such as convolution neural networks (CNNs). These categorical features are represented as large embedding tables. The size of the embedding tables can exceed hundreds of Gigabytes of storage [6, 18]. In many cases, the size of the embedding tables is limited to the available main memory on the servers [6]. One alternative is to store these large embedding tables in solid-state drives (SSDs). These SSDs offer higher storage capacities than main memory (DRAM). However, they exhibit slower read and write performance, which can be a major bottleneck for adopting an SSD-based neural recommendation system in the cloud.

There are two main approaches to improve the latency of an SSD-based neural recommendation system. The first approach is to perform the computation near the storage to fully utilize the internal SSD bandwidth and reduce round-trip data communication overheads [24]. The second approach is to cache frequently accessed embedding vectors in main memory (DRAM) to minimize the number of slow SSD read and write operations [3, 26].

ICPE '22, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9143-6/22/04…$15.00
https://doi.org/10.1145/3489525.3511672

A SmartSSD® [25] is an SSD with an on-board FPGA that can process data in-situ. The FPGA on a SmartSSD® offers computational capabilities to the storage device. FPGAs offer a high degree of parallelism within an affordable power budget. On a SmartSSD®, the FPGA can be customized to perform certain operations in an energy efficient manner when compared to a CPU. In addition, SmartSSD® includes an external DRAM accessible by the FPGA and the CPU that enhances programmability and allows direct communication between the FPGA and the host CPU. In summary, a SmartSSD® has three main features that can make it an attractive solution for applications with high storage requirements. First, SmartSSD® can use an FPGA to improve performance and energy efficiency with custom hardware. Second, the computation scales with the storage. Using multiple SmartSSD® provides more storage as well as more computation capabilities. Third, offloading the computation to the FPGA on a SmartSSD allows the CPU to perform other tasks.

In this paper, we propose SmartRec, a near-storage inference engine for an SSD-based neural recommendation system using SmartSSDs®. SmartRec offloads the entire embedding table operations, including gather and aggregation computations, to the FPGA on a SmartSSD®. Besides the embedding table operations, other computation layers such as fully connected layers can also be fully or partially offloaded to the FPGA on the SmartSSD®.

SmartRec performs the computation near the storage utilizing the higher bandwidth internal link between the SSD and the FPGA. Further, SmartRec customizes the FPGA on the SmartSSD for the recommendation inference task. It utilizes the FPGA's DRAM as a cache to minimize slow SSD accesses. Finally, SmartRec utilizes multiple SmartSSDs® to improve performance by harnessing data parallelism. Hence, SmartRec improves the performance and energy efficiency of the recommendation inference task.

To evaluate SmartRec, we built a prototype of one of the state-of-the-art recommendation models from Facebook called DLRM [17]. We studied three classes of the DLRM models that represent both compute-intensive as well as memory-intensive models. Our results show SmartSSDs® are up to 10× more energy-efficient than CPUs for the recommendation inference task. SmartRec can achieve a similar inference time as the CPU for smaller batch sizes, despite having limited resources compared to a high-end CPU. We demonstrate that our caching technique improves the inference time of an SSD-based recommendation system. These results will motivate future work on improving the locality of embedding table accesses. Finally, we show that our design can scale to multiple SmartSSDs® to take advantage of the available data parallelism.

2 BACKGROUND

We provide background on deep learning-based recommendation models and the SmartSSD® architecture.

2.1 Recommendation Systems

Recommendation systems suggest the most relevant content or items of value to users. The recommendation systems form the basis of various online services such as shopping [23], entertainment [27], and social media [17] that requires real-time responses. Deep learning-based recommendation systems are the state-of-the-art methods deployed on the cloud and can respond to a user's request in real-time. The *inference* task of these deep learning-based recommendation systems account for a significant fraction of the computation in modern data-centers [6, 8].

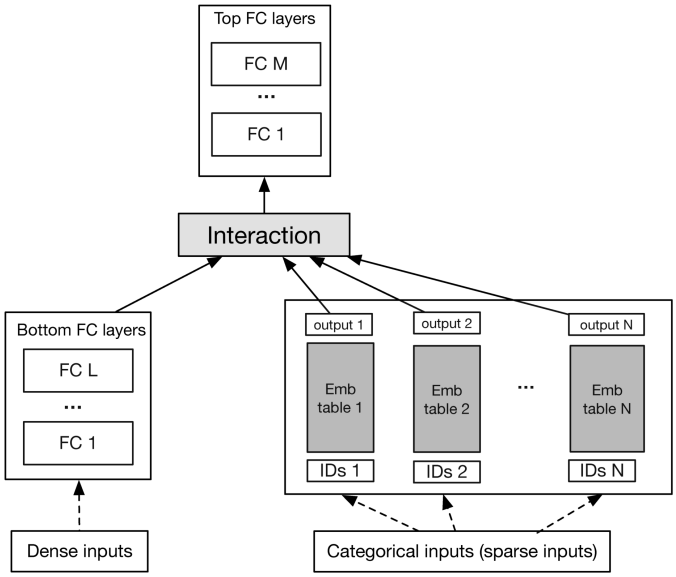
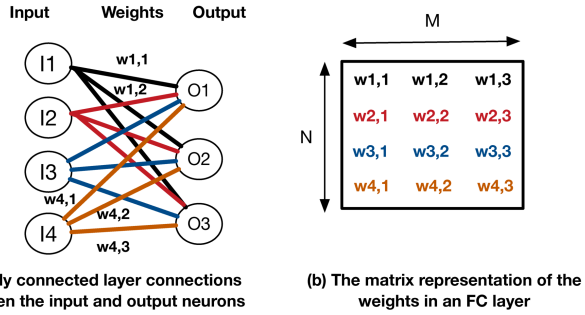


Figure 1: A high-level overview of deep learning-based recommendation models. The model consists of fully connected (FC) layers in the bottom and top layers and several embedding tables.

Algorithm 1: SparseLengthsSum (SLS) pseudo-code

```
1  Emb ← Embedding Table : RXC
2  IDs ← Vector : M
3  Out ← Vector : C
4  procedure SLS( Input Emb, IDs, Output Out)
5  for each ID ∈ IDs do
6  |   Emb_vector = Emb[ID]
7  |   for each c ∈ C do
8  |   |   Out[c] += Emb_vector[c]
9  |   end
10 end
```

Architecture of Recommendation Systems. Figure 1 shows the overall architecture of a neural network-based recommendation system. The model comprises two main components: *fully connected* (FC) layers (or dense layers) and *embedding tables*. There are two sets of FC layers: the bottom and top layers. An *interaction layer* combines the output features from the bottom layers and passes them to the top FC layers. The FC layers are used to extract the dense features similar to other classical neural networks like CNNs. In contrast, the operation on the embedding table processes the categorical or sparse features. FC layers and embedding tables stress different parts of the system. FC layers require compute capabilities. In contrast, the operation on the embedding tables stresses the memory system as they perform multiple irregular memory accesses. The recommendation models vary depending on their FC layers, their embedding tables sizes, and the number of



(a) Fully connected layer connections between the input and output neurons

(b) The matrix representation of the weights in an FC layer

Figure 2: An FC layer and how the weights between the input and output neurons can be transformed into a matrix before performing a matrix-vector multiplication. The labels are shown only for some of the weights.

lookups. The recommendation system can be compute-bound or memory-bound depending on the parameters of the model.

Fully-connected Layers. Most recommendation systems take advantage of the fully connected layers to extract dense features from the input vector. In a fully connected layer, all the inputs neurons are connected to all the outputs neurons in the next layer (See Figure 2(a)). Figure 2(b) shows how the weights between the input and output neurons are transformed into a matrix. The number of rows and columns in the matrix matches the number of input and output neurons in a layer. For the example in Figure 2, there are 4 inputs and 3 outputs neurons. A single FC layer computation is a matrix-vector multiplication. The weights are learned during the training phase and do not change during the inference phase. It is common to process multiple inputs together as one batch. All input vectors in a batch are combined to build a matrix. Hence, the computation with a batch is organized as a general matrix-matrix multiplication (GEMM) operation.

Processing Categorical Inputs. The recommendation model processes categorical input features (sparse features) using embedding table operations. Each row of the embedding table is a unique embedding vector typically comprising tens of features (i.e., number of columns in the table). A set of embedding vectors (specified by a list of IDs) is gathered and aggregated using a sparse length sum operation for each inference task. Algorithm 1 shows the pseudo-code for sparse length sum operation. Each model comprises multiple embedding tables.

Processing categorical inputs requires a large amount of storage, performs irregular memory accesses, and has low compute intensity. Production-scale embedding tables can have billions of rows. Aggregate storage need for these embedding tables in a neural recommendation model can be several terabytes [26]. Further, categorical input features are sparse. A small fraction of the embedding vectors are accessed in each inference task. Hence, embedding table operations perform many random and irregular accesses to such tables in memory. The computation intensity of the operation on embedding tables is orders of magnitude lower than the computation for other neural networks such as CNN and RNNs.

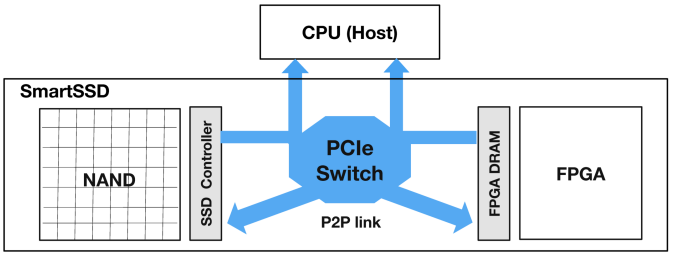


Figure 3: The overall architecture of a SmartSSD® device.

2.2 SmartSSD® Architecture

We provide a background on some features of a SmartSSD® device. Figure 3 shows the high-level internal architecture of a SmartSSD®. It consists of a NAND array (storage component), an SSD controller, a reasonably sized FPGA, and a DRAM module. Table 3 provides the technical specifications of the FPGA in the SmartSSD®. The FPGA can communicate directly with the storage component, known as Peer-to-Peer (P2P) data transfer, that facilitates near-storage computation. The P2P data transfer eliminates the unnecessary round-trip traffic between the SSD to the CPU (host) and from the CPU to an FPGA.

The DRAM on the SmartSSD® is accessible by the FPGA. There is a specific region of memory called the common memory area (CMA) in the DRAM that is accessible both by the FPGA and the CPU (host). This CMA region is used for directly transferring the data between the storage component and the FPGA. While the CPU host is not involved in the data movement from the SSD to the FPGA with the peer-to-peer transfer, it initiates the data transfer. Besides, the host and the FPGA on a SmartSSD® can communicate with the CPU by mapping the CMA region to the host's address space. The SmartSSD® supports the OpenCL programming model and thus uses OpenCL APIs for kernel launch, memory allocation, and data transfers.

Computation can be offloaded to the FPGA on a SmartSSD® either completely or partially. Additionally, the result of the computation performed by the FPGA can be either directly written back to the SSD or the main memory accessible by the CPU.

3 SMARTREC DESIGN

This paper presents SmartRec, an FPGA acceleration engine for neural recommendation inference using SmartSSDs®. We offload two main computations of the neural recommendation models, namely sparse length sum and general matrix-matrix multiplication, to the FPGA on a SmartSSD®. We carefully design each hardware unit based on the available resources on a SmartSSD®. The embedding tables can be stored either in the DRAM or in the SSD. This feature allows SmartRec to support models with huge embedding table sizes. To accelerate large models where the embedding tables are stored on the SSD, we propose a novel caching technique designed for SmartSSDs® that allows the kernel on the FPGA to use its external DRAM as a cache for the data on the SSD. Our caching technique helps to minimize the high latency accesses to the SSD. The cache is managed by the host on the CPU. The kernel on the FPGA uses the information provided by the host to locate the data in its external

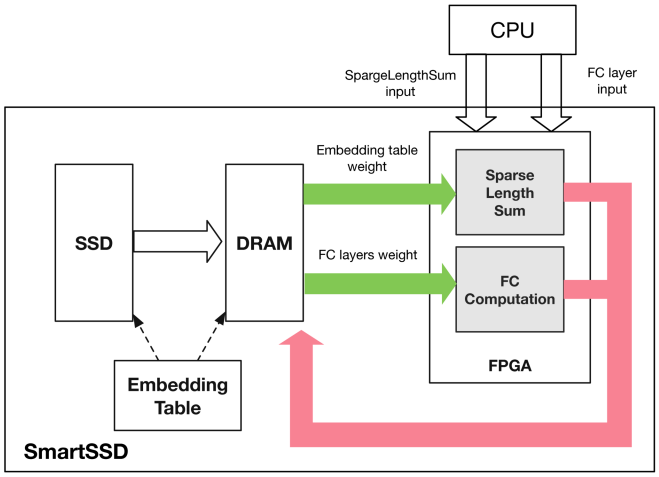


Figure 4: A high-level overview of the SmartRec architecture.

DRAM. Finally, we demonstrate how our design scales to multiple SmartSSDs® to perform the computation on different embedding tables in parallel.

3.1 Overview of SmartRec Architecture

Figure 4 depicts the overall design of SmartRec. In our design, we offload the sparse length sum and the matrix-matrix multiplication operations to the FPGA. Hence, the two main components of the FPGA in SmartRec are (1) the sparse length sum unit that features the embedding table lookups and (2) the matrix-matrix multiplication unit. The sparse length sum unit handles categorical inputs (*i.e.*, sparse features), and the latter perform the FC layers that extract the dense features. The FPGA reads the inputs from the DRAM and writes the result of the computation back to the DRAM. There are two options for storing embedding tables on a SmartSSD®. For smaller models, the embedding tables can be stored on the DRAM of the FPGA. Current SmartSSDs® support up to 4GB of DRAM memory. If the size of the embedding tables exceeds the FPGA DRAM's capacity, the embedding tables are stored on the SSD, which provides up to terabytes of storage. SmartRec targets the inference task where the values in the embedding tables and the weights in the FC layers are fixed. Hence, the embedding table and the weights of the FC layers can be placed either in the FPGA DRAM or the SSD before the computation. The host CPU provides the inputs to the kernel on the FPGA. The CPU generates two input vectors for each inference request: one for the FC layer and the other for the embedding table lookups (See Section 2). The FPGA performs the computation and writes the result back to the memory accessible by the CPU. Next, we will show different options for offloading the computation to the FPGA on a SmartSSD®.

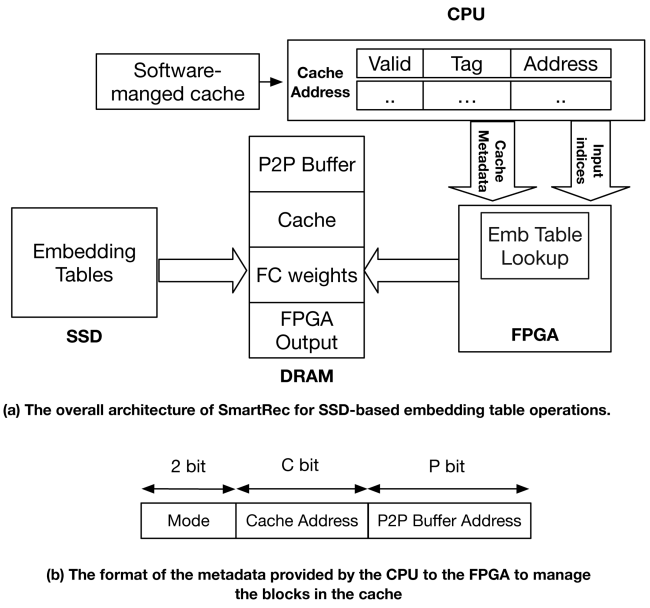
3.2 Offloading Options

In SmartRec, the computation of a recommendation inference task can be either completely or partially offloaded to the FPGA on a SmartSSD®. We evaluated both these options to identify the trade-offs in communication overheads and performance. In the first case, we offload all the layers in a recommendation model to the FPGA on the SmartSSD®. In this case, the CPU only initiates the data transfers and starts the computation, and has minimal involvement in the process. The second option offloads only the bottom layers, including the bottom fully connected layers and the embedding operations to the FPGA, and the CPU computes the top fully connected layers. The computations on the FPGA and the CPU can then be pipelined. Unlike the first option, the second solution involves the CPU in the computation. In our experimental evaluation, we will compare these two options in terms of performance. Next, we will discuss the details of each unit and its architecture.

3.3 The Sparse Length Sum Unit

Sparse length sum operation involves many random lookups to the embedding tables. The number of the embedding table lookups is a design parameter and is different for various models (See Table 2). On a SmartSSD®, the FPGA receives the list of IDs or row indices to be accessed for the sparse length sum operation. Each embedding table receives a different set of indices. The FPGA accesses the DRAM to read the corresponding entries and then performs the sum operation to calculate the final result. Algorithm 1 shows the pseudo-code for the sparse length sum operation. There are two ways to improve the performance of the lookups. One approach is to parallelize the lookups using multiple memory channels. This is possible if DRAM accessible by the FPGA supports multiple channels. The current SmartSSDs® are equipped with DDR4 memory that only exposes one DRAM channel to the FPGA. The second optimization is to use wider vectors to read the data on the DRAM. Wider vectors are recommended for DRAM accesses to minimize the DRAM controller overheads. Each embedding table has a certain number of features. The number of features represents the number of columns for each row of the embedding table. The sum operations are performed on each feature separately. One way to minimize the number of accesses is to use a wide vector composed of multiple columns. For example, with 32 features, where each feature is 4 bytes, the whole row can read in one access with a 1024 bit (128 bytes) vector.

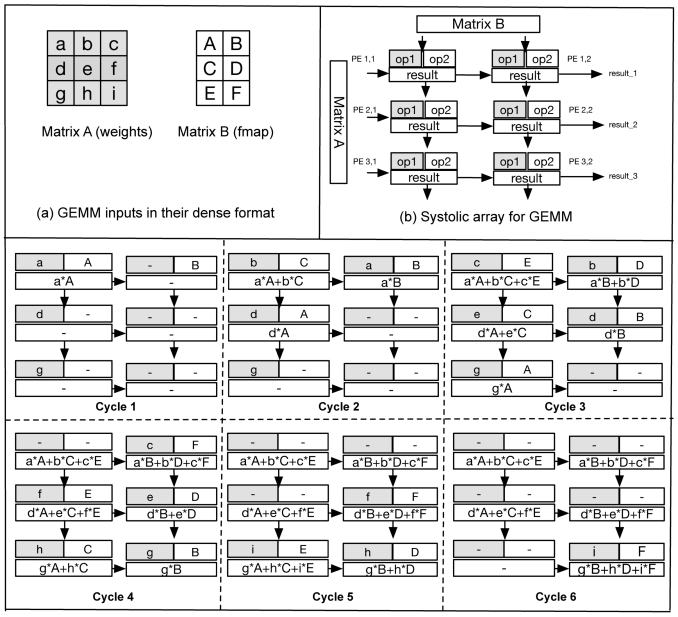
SSD-based Embedding Table Lookups. The aggregate storage requirements for many production-level recommendation models can exceed the FPGA DRAM capacity. This limits the capability of FPGAs to support the model with large embedding tables. The main advantage of SmartSSDs® is the presence of terabytes of SSD storage suitable for large embedding tables. Further, the data on the SSD is directly accessible by the FPGA via a P2P transfer in a SmartSSD® (see Section 2). This direct communication between the FPGA and the storage unit removes unnecessary traffic between the SSD to the host and from the host to the FPGA. It also allows the computation to occur near the storage. While using the SSD increases the storage capabilities, the SSDs have higher latency and lower bandwidth than the DRAM memory, degrading the inference performance. Despite irregular memory accesses, the embedding table operations can still benefit from caching techniques as a result of locality in their accesses [3, 24]. In this paper, we propose a novel software-managed cache tailored for SmartSSDs®. We use the FPGA's DRAM to cache some of the frequently accessed data on the SSD. This caching can minimize the total number of SSD



(a) The overall architecture of SmartRec for SSD-based embedding table operations.

(b) The format of the metadata provided by the CPU to the FPGA to manage the blocks in the cache

Figure 5: A high-level overview of our proposed software-managed cache designed for SmartSSDs®. (a) Different regions in the FPGA DRAM and the connections between the CPU and FPGA. (b) The metadata format is used to communicate the cache information between the CPU and the FPGA.



(a) GEMM inputs in their dense format

(b) Systolic array for GEMM

(c) Cycle by cycle execution of GEMM with an output-stationary dataflow

Figure 6: Illustration of SmartRec's GEMM unit. (a) Inputs to the GEMM unit. (b) A systolic array for the GEMM unit. (c) Illustration of GEMM computation at various steps. We show the current inputs and the partial results computed till a step for each PE. We demonstrate the output-stationary attribute of our design.

accesses and reduce the end-to-end inference time for an SSD-based recommendation system.

Figure 5 shows the overview of our cache design. We divide the DRAM into four regions. One region belongs to the data newly brought from the SSD to the DRAM via a P2P transfer (*e.g.*, P2P buffer). The second region used a cache to reserve some previously fetched blocks in DRAM (*e.g.*, cache buffer). In addition, we need space in the DRAM to store the weights for other layers in the model (e.g., FC layers) and the output of the FPGA. The sizes of each region are design parameters and are chosen based on different factors such as the number of lookups and the batch sizes. The P2P region should be large enough to accommodate all the data required for a batch size if none of the accesses hit in the cache.

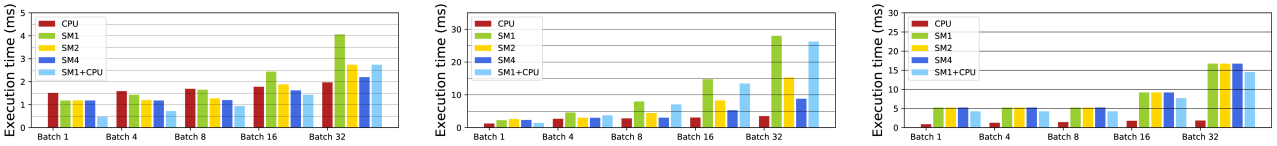
The cache (i.e., DRAM) is managed by the host CPU and therefore is a software-managed cache. The CPU issues a P2P read from the SSD to the DRAM when the data is not present in the cache. The host sends metadata to the FPGA for each embedding table access to convey the information about the location of each access. Figure 5(b) provides the details on the metadata provided to the FPGA by the CPU. The metadata consists of three parts. We use two bits to identify three different possible scenarios (explained below). Besides, the FPGA uses cache address and buffer address to locate the data depending on the mode bits as follows.

- **Mode 0** The data block is **not** present in the cache. The FPGA reads the block using the address specified by the *buffer address* bit. Additionally, the FPGA reserve the block at the address provided by the *cache address* bits.
- **Mode 1** Similar to mode 0 with the difference that the FPGA does **not** reserve the block in the cache.
- **Mode 2** The data is present in the cache at the address specified by the *cache address* bits.

Using a software-managed cache allows the designer to choose the right caching scheme based on the access pattern and profiling information. Finally, the FPGA in our design is oblivious to the cache policy used. Hence, the software-managed cache can be changed while the FPGA side remains intact, avoiding hours of compilation.

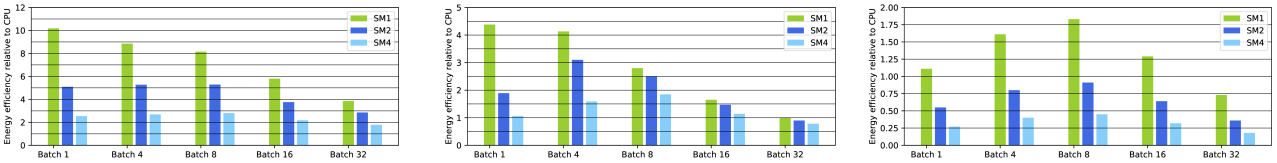
3.4 The Matrix-Matrix Multiplication Unit

We use a systolic array-based architecture for the GEMM unit. Many recent hardware-based GEMM accelerators use systolic arrays because there are more efficient than other methods [13, 20]. Figure 6 shows the details of the GEMM unit. We use an output-stationary dataflow where the partial results remain in the processing elements (PEs). At the same time, the two input matrices are streamed in the systolic array, one from left-to-right and the other from top-to-bottom. An output-stationary dataflow ensures maximum reuse of the output data. Figure 6(c) illustrates all the steps and partial results computed in the GEMM unit for the two example inputs in Figure 6(a). Each PE has a multiply-accumulate (MAC) unit. There are three buffers inside each PE. One buffer for each of the two inputs and one buffer is used as a work queue for the MAC unit. The inputs elements arrive in each PE in the proper order. Thus, there is no need to have additional logic to match the coordinates



(a) Execution time for RMC1. (b) Execution time for RMC2. (c) Execution time for RMC3.

Figure 7: Comparing the inference time of the FPGA on SmartSSD® and the baseline CPU for three DLRM models: RMC1 (fig a), RMC2 (fig b), and RMC3 (fig c).



(a) Energy efficiency for RMC1. (b) Energy efficiency for RMC2. (c) Energy efficiency for RMC3.

Figure 8: Energy efficiency of different number of SmartSSDs® over the baseline CPU for three DLRM models: RMC1 (fig a), RMC2 (fig b), and RMC3 (fig c).

Table 1: The FPGA resource utilization and frequency for three DLRM models.

Application	Frequency	LUT	BRAM	DSP	REG
RMC1	219 Mhz	21.5%	41.5%	20.6%	17.2%
RMC2	221 Mhz	21.6%	42.9%	20.6%	17.6%
RMC3	216 Mhz	21.5%	41.5%	20.6%	17.2%

of the elements of the two input matrices. The size of the systolic array depends on the available DSP resources on the FPGA. We used 16 rows and 8 columns (*i.e.*, a total of 128 PEs).

3.5 Scalability

Different recommendation models have a different number of embedding tables (See Table 2). For each inference request, there are multiple lookups for each embedding table. The operations on each embedding table can be performed independently. The embedding tables can be stored on different SmartSSDs®. Similarly, the embedding table operations can be performed by the FPGAs inside each SmartSSD® in parallel. The host can then collect the final outputs of each FPGA. This demonstrates one of the main features of SmartSSD®, where the computation scales with the storage.

4 EVALUATION

We evaluated the performance and energy efficiency of SmartRec for an end-to-end recommendation inference. SmartRec outperforms the CPU implementation in inference time for small batch sizes while it improves the energy efficiency for almost all scenarios.

4.1 Experimental Setup

Model Specification. We used the state-of-the-art recommendation model from Facebook called DLRM [17] for our performance evaluation. The DLRM model has three classes of recommendation models, namely, RMC1, RMC2, and RMC3 [6]. The three models highlight the diversity in the computation and memory accesses. Each class has different embedding tables and FC layers sizes. Table 2 summarizes the parameters of RMC1, RMC2, and RMC3 models. These parameters are based on prior works [17, 24] and open-source implementation of the DLRM model, which might be different from the parameters deployed on the cloud by Facebook.

Environment. We implemented the three classes of the DLRM model using the Xilinx HLS tool that translates C++ programs to the hardware description language (HDL). The HDL code is then used to generate the FPGA bitstream. We studied each model separately on a SmartSSD®. Table 3 shows the SmartSSD® configuration. Table 1 presents the logic utilization and the frequency for all the three models on the FPGA in a SmartSSD®.

Baseline System. We compared our prototype with a multi-thread CPU implementation of DLRM in Pytorch Version 1.9. Table 3 presents the details of the CPU we used for our experiments. We employed the open-source version of DLRM [17]. For all of our experiments, we run each experiment 10 times and report the median execution time.

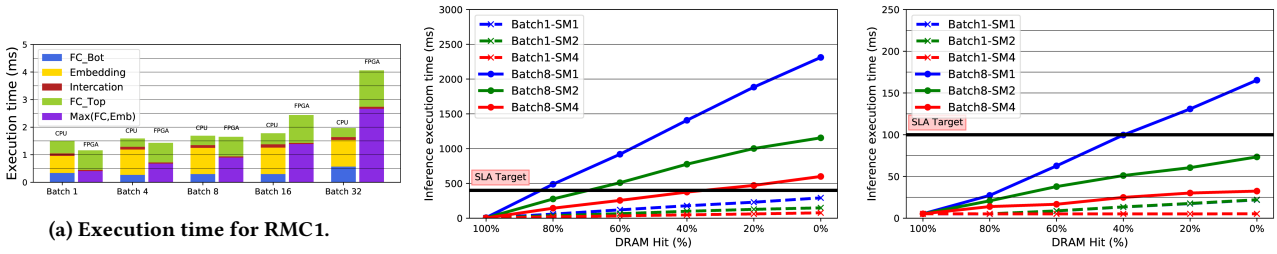
Measuring Power Consumption. To measure FPGA power consumption, we used the Xilinx Board Utility command line interface. We measured the power consumption of the CPU using Processor Counter Monitor (PCM) [1], which gives a set of application programming interfaces (APIs) to monitor the performance and energy metrics of Intel Processors.

4.2 Evaluation with A DRAM-based Recommendation System

We first study the scenario where the embedding tables fit in the FPGA's DRAM. Figure 7(a-c) compares the end-to-end inference time for three DLRM recommendation models for the baseline CPU

Table 2: Different classes of models in DLRM and their parameters.

Model	Description	Fully Connected		Embedding Tables			
		Bottom	Top	Number of Tables	Elements Per Table	Feature Size	Lookups
RMC1	Small FC Few Emb. Tables Large Emb. Tables	Layer1: 128 X 64 Layer2: 64 X 32	Layer1: 288 X 256 Layer2: 256 X 64 Layer3: 64 X 1	8	1000000	32	80
RMC2	Small FC Many Emb. Tables Small Emb. Tables	Layer1: 128 X 64 Layer2: 64 X 64	Layer1: 2112 X 128 Layer2: 128 X 64 Layer3: 64 X 1	32	50000	64	120
RMC3	Large FC Few Emb. Tables Small Emb. Tables	Layer1: 2560 X 1024 Layer2: 1024 X 256 Layer3: 256 X 32	Layer1: 352 X 512 Layer2: 512 X 256 Layer3: 256 X 1	12	100000	32	24



(a) Execution time for RMC1.

(b) Execution time for RMC2.

(c) Execution time for RMC3.

Figure 9: The execution time of the SSD-based DLRM models for batch sizes 1 and 8 under different DRAM hit rates. The SLA targets for each model are specified with a dark horizontal line.

Table 3: The details of the CPU and SmartSSD® configurations.

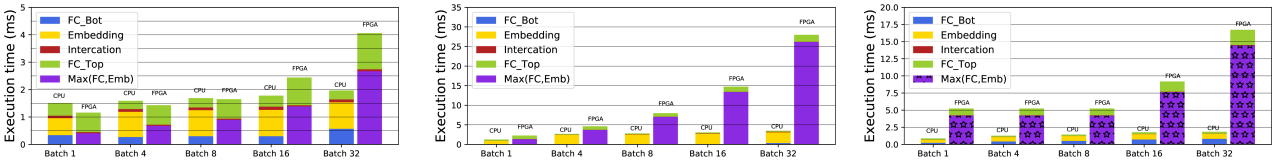
Platform	Configuration
CPU	Skylake machine, cores per socket=22, sockets=2, frequency=2.1 GHz DRAM capacity=768 GB, DRAM type= DDR4, DRAM frequency= 2666 Mhz, DDR bandwidth per socket= 249 GB/s L1 size= 32 KB, L2 size= 1024 KB, L3 size= 30976 KB
SmartSSD®	storage= 4 TB, FPGA LUT=391 K, FPGA BRAM= 503, FPGA DSP unit= 960 DRAM= 4 GB, DRAM type= DDR4, FPGA DRAM bandwidth= 19 GB/s

and the different SmartSSD® configurations. The figure compares the execution time for five different batch sizes (x-axis). In addition to using different numbers of SmartSSDs®, two different offloading options are presented. SM1, SM2, and SM3 show the execution time when all the layers are offloaded to one, two, and four SmartSSD®, respectively. SM1+CPU shows the case where the bottom layers (i.e., bottom FC and sparse length sum) are offloaded to FPGA on one SmartSSD® while the CPU performs the computation for the top FC layers. The two computations are then pipelined (see Section refsmartrec).

Performance. RMC1, RMC2, and RMC3 are diverse in terms of the number of embedding lookups, their FC layers sizes, and the embedding tables storage requirements. RMC2 incurs more memory lookups than the other two models. In contrast, RMC3 has larger FC layers sizes, making it more compute-intensive than RMC1 and RMC2. To compare the execution time of SmartSSD® and the CPU, we consider the systems' maximum theoretical memory bandwidth and floating-point operations per second (FLOPS). The CPU we used for our experiments has 13×, and 92× higher DRAM bandwidth and theoretical FLOPS than the FPGA on the SmartSSD (see Table 3).

For RMC2, the embedding lookups dominate the execution time. The FPGA is slightly slower than the CPU for smaller batch sizes. For larger batch sizes, the difference in execution between the CPU and the FPGA increases due to the FPGA's lower DRAM bandwidth than the CPU. For RMC3, the most compute-intensive model among the three models, the CPU outperforms the FPGA across all the batch sizes. Even for large batch sizes, the model parameters for FC layers are within the CPU's L1 and L2 cache sizes, unlike the FPGA, which has limited on-chip storage and floating-point units than the CPU. As a result, the FPGA experience higher latencies for larger batch sizes. For RMC1, the FPGA outperforms the CPU for small batch sizes. Nevertheless, as the batch size increases, the execution time of the FPGA increases at a higher rate than the CPU due to the reasons explained earlier.

Scalability. Using multiple SmartSSDs® is an effective way to improve the overall performance for the scenarios where the embedding table lookup dominates the inference time. In Figure 7(a-c), two (SM2) and four SmartSSDs® (SM4) outperform the one SmartSSD® (SM1) for RMC1 and RMC2, where the embedding lookups dominate the bottom layer execution time. Using multiple SmartSSDs® allows us to exploit the existing data parallelism by performing the operations on different tables in parallel. For RMC3, having multiple SmartSSDs® does not improve the overall inference time



(a) Execution breakdown for RMC1. (b) Execution breakdown for RMC2. (c) Execution breakdown for RMC3.

Figure 10: The execution breakdown of CPU and FPGA for different DLRM models.

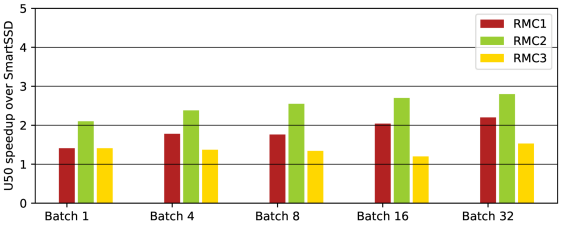


Figure 11: Comparing the performance of FPGA on SmartSSD® with Alveo-U50 FPGA featuring a HBM exterenal memory.

as the FC layers are the bottleneck for the bottom layers' execution time.

CPU-FPGA Pipeline. The light blue bars (SM1+CPU) in Figure 7 highlights the case where the bottom and top layers are performed by the FPGA and the CPU, respectively. The two computations are pipelined. This helps to improve the inference time (e.g., decrease the latency) compared to using one SmartSSD® (green bars) for all the cases.

Energy Efficiency. Figure 8(a-c) presents the relative energy efficiency of the different number of SmartSSDs® in comparison to the baseline CPU. Offloading the computation to a SmartSSD® improved the energy efficiency compared to the CPU for various settings across the three models and batch sizes. According to Figure 8(a-c), using one SmartSSD® achieves higher energy efficiency than using multiple SmartSSDs®. Overall, SmartSSD® is up to 10×, 4×, and 1.75× more energy efficient than the baseline CPU for RMC1, RMC2, and RMC3, respectively.

4.3 Evaluation with An SSD-Based Recommendation System

We also evaluated the performance of SmartRec for the cases where the embedding tables are stored on the SSD. This is necessary for recommendation models where the storage requirement for the embedding table exceeds the DRAM capacity. As we show in Section 3, we propose a caching scheme to use the FPGA external DRAM to cache some of the accessed blocks and take advantage of the localities in the accesses. We study the inference time under different cache hit rates. Figure 9 shows the execution time of different models on SmartSSDs® for batch size 1 and batch size 8. The X-axis shows the cache hit ratios. The 100% cache hit ratio is equivalent to the case where the embedding table fits on the FPGA DRAM. The zero percentage cache hit ratio highlights the scenario where none of the lookups are available on the FPGA DRAM. For each model, the service level agreement (SLA) requirements are shown with a horizontal line. The recently published work reports the SLA numbers from Facebook [5]. The SLA targets for RMC1, RMC2, and RMC3 are 100 ms, 400 ms, and 100 ms, respectively. For batch size 1, the execution time is within the required target for all three models. However, the execution time does not meet the target for larger batch sizes for a low cache hit rate. For example, for RMC1, for less than 80% cache hit rates, the execution time for one SmartSSD® (SM1) starts to exceed the SLA target. For the case of 4 SmartSSDs®, the execution time meets the requirement for all the cache hit rates. Another important observation from Figure 9(c) is that for RMC3, for almost all cache hit ratios, the execution time meets the target requirement since the majority of the execution time is spent on the FC layer computation. Therefore the operation on the embedding table has less impact on the overall inference time.

4.4 Performance Characterization

Execution Breakdown. Figure 10 show the breakdown of the execution time for the CPU and FPGA to understand better which layers are the bottleneck for each model. For SmartSSD®, we combine the execution of the FC bottom layers and sparse length sum operation on the embedding vectors since both of the computations run in parallel on the FPGA. To show which computation dominates the other, we marked the bars with ☆where the FC layers dominate the embedding lookups in execution time. For RMC1 and RMC2, the majority of the time is spent on the operations on the embedding table. In contrast, for RMC3, the execution time is dominated by the bottom FC layers computation.

High Bandwidth Memory for FPGA.. To characterize the external memory bandwidth's impact on the system's performance on the FPGA, we compared the execution time for the FPGA on a SmartSSD® with the FPGA that has access to high bandwidth memory (HBM). To do so, we compiled and ran the kernel on Xilinx Alveo-U50 FPGAs that features HBM memory. We used the exact kernel without changing the number of used DSP units or on-chop memory resources for the kernel on U50. Figure 11 compares the relative speedup for U50 devices over the SmartSSD® for the three DLRM models. Using HBM improved the performance by almost 3× for RMC2, the most memory-intensive model out of the three models. For RMC3, using HBM is less helpful since the FC layers computation dominates the execution time.

5 RELATED WORK

Neural network-based recommendation systems has got a lot of attention in recent years due to their role in numerous internet services including e-commerce [7, 28], social media [6, 17]. While several prior art studies designing efficient hardware and software for various neural networks such as CNNs and RNNs, fewer studies explored efficient hardware and software systems for the recommendation system. In this section, we overview some of the more closely related works.

Near-storage computation many commercial examples adopt near-storage acceleration. Oracle Exdata [2] and IBM Netezza [4] are the two examples where FPGA is deployed in the datapath between hard disk drive (HDDs) and the CPU. Similarly, some prior arts demonstrate the benefit of SmartSSD® in accelerating the database query performance and energy efficiency [15, 19, 21]. Unlike database query processing, the operation on the embedding tables exhibits random and irregular memory access to the memory.

CPUs and GPUs recommendation systems. [6] presents a collection of production-scale deep learning-based personalized recommendation system. In addition, they conduct in-depth system analysis to characterize the recommendation system and optimization suggestions for general-purpose architectures. [5] propose an efficient scheduling algorithm by taking into account the characteristics of query sizes and request arrival patterns for heterogeneous systems comprising of CPUs and GPUs. Similarly, [16] accelerates the recommendation system using a heterogeneous system composed of a CPU and a GPU.

FPGAs for recommendation systems. [9–11] use FPGAs to accelerate the inference of the deep learning recommendation inference. [9] uses Intel HARPv2, a package-integrated CPU+FPGA device for their inference engine, while [10] use FPGA with access to a high bandwidth memory (HBM) technology. In addition, they proposed a new data structure to reduce the number of DRAM accesses. Unlike their work, we study systems that can accommodate large embedding tables that exceed the main memory capacity.

Near-storage processing for recommendation systems. [14] address the memory bottleneck by exploiting DIMM-level parallelism in DRAM and supporting tensor operations, e.g., gather and reduction, within the DRAM. Similar to this work, they added memory-side-caching for frequently accessed entries. Similar to this work, Bandana [3] uses non-volatile memory for storing deep learning models. They suggest multiple techniques to increase the effective read bandwidth of NVM to address the limited read bandwidth compared to a DRAM system. RecNMP [12] proposes that provides a scalable solution for a wide range of sparse embedding for production scale models. They studied several optimizations such as memory-side caching, packet scheduling, and hot entry profiling to improve the performance. RecSSD [24] is another near data processing solution customized for neural recommendation inference. By offloading computations for key embedding table operations, RecSSD reduces round-trip time for data communication and improves internal SSD bandwidth utilization. In contrast to our work, they utilize the small embedding processor inside the SSD controller to perform the computation near the storage.

6 CONCLUSION

This work explores using SmartSSDs®, an SSD equipped with an FPGA, for a large-scale deep learning-based recommendation system inference task. We investigate various options to offload the neural recommendation model computation to the FPGA on a SmartSSD®. In addition, we propose various optimizations, such as a software-managed cache for a SmartSSD® that uses the FPGA external DRAM as a cache to hide the high latency SSD accesses. We demonstrate the scalability of our approach by offloading the computation on embedding tables to multiple SmartSSDs®. Our evaluation shows that SmartSSD® achieves high energy efficiency while offering higher storage capacity than a DRAM-based CPU system. Our results confirm that SmartSSDs® meet the target latency when there are sufficient localities in embedding tables accesses. This result motivates future work to explore techniques to improve the locality of accesses to the embedding tables.

REFERENCES

[1] [n. d.]. Processor Counter Monitor (PCM). Available at https://github.com/opcm/pcm.

[2] 2012. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. *White Paper* (2012).

[3] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. 2018. Bandana: Using Non-volatile Memory for Storing Deep Learning Models. arXiv:1811.05922 [cs.LG]

[4] Phil Francisco. 2011. The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics. *IBM Redbook* (2011).

[5] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 982–995. https://doi.org/10.1109/ISCA45697.2020.00084

[6] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. 2020. The Architectural Implications of Facebook's DNN-Based Personalized Recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 488–501. https://doi.org/10.1109/HPCA47549.2020.00047

[7] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and et al. 2019. Applying Deep Learning to Airbnb Search. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Jul 2019). https://doi.org/10.1145/3292500.3330658

[8] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 620–629. https://doi.org/10.1109/HPCA.2018.00059

[9] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. *Centaur: A Chiplet-Based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations*. IEEE Press, 968–981. https://doi.org/10.1109/ISCA45697.2020.00083

[10] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B. Preuß er, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions. In *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica (Eds.), Vol. 3. 845–859.

[11] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. *FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters*. Association for Computing Machinery, New York, NY, USA, 3097–3105. https://doi.org/10.1145/3447548.3467139

[12] Liu Ke, Udit Gupta, Carole-Jean Wu, Benjamin Youngjae Cho, Mark Hempstead, Brandon Reagen, Xuan Zhang, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, and Xiaodong Wang. 2019. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. arXiv:1912.12953 [cs.DC]

[13] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 821–834. https://doi.org/10.1145/3297858.3304028

[14] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 740–753. https://doi.org/10.1145/3352460.3358284

[15] Veronica Lagrange Moutinho dos Reis, Harry (Huan) Li, and Anahita Shayesteh. 2020. Modeling Analytics for Computational Storage. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 88–99. https://doi.org/10.1145/3358960.3375794

[16] Yang Li and Zhitao Dai. 2019. Design and Implementation of Hardware Accelerator for Recommendation System based on Heterogeneous Computing Platform. https://doi.org/10.2991/icmeit-19.2019.150

[17] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019). https://arxiv.org/abs/1906.00091

[18] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, Juan Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. arXiv:1811.09886 [cs.LG]

[19] Sahand Salamat, Armin Haj Aboutalebi, Behnam Khaleghi, Joo Hwan Lee, Yang Seok Ki, and Tajana Rosing. 2021. NASCENT: Near-Storage Acceleration of Database Sort on SmartSSD. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 262–272. https://doi.org/10.1145/3431920.3439298

[20] Mohammadreza Soltaniyeh, Richard P. Martin, and Santosh Nagarakatte. 2021. SPOTS: An Accelerator for Sparse CNNs Leveraging General Matrix-Matrix Multiplication. arXiv:2107.13386 [cs.AR]

[21] Mohammadreza Soltaniyeh, Veronica Lagrange Moutinho Dos Reis, Matthew Bryson, Richard Martin, and Santosh Nagarakatte. 2021. Near-Storage Acceleration of Database Query Processing with SmartSSDs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 265–265. https://doi.org/10.1109/FCCM51124.2021.00052

[22] Moshe Tennenholtz and Oren Kurland. 2019. Rethinking Search Engines and Recommendation Systems: A Game Theoretic Perspective. *Commun. ACM* 62, 12 (Nov. 2019), 66–75. https://doi.org/10.1145/3340922

[23] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery ; Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 839–848. https://doi.org/10.1145/3219819.3219869

[24] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. 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* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 717–729. https://doi.org/10.1145/3445814.3446763

[25] Xilinx-Samsung. 2021. SmartSSD. https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html

[26] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. 2019. AIBox: CTR Prediction Model Training on a Single Node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 319–328. https://doi.org/10.1145/3357384.3358045

[27] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending What Video to Watch next: A Multitask Ranking System. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997

[28] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. arXiv:1809.03672 [stat.ML]

[29] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1059–1068. https://doi.org/10.1145/3219819.3219823