iMARS: An In-Memory-Computing Architecture for Recommendation Systems

Mengyuan Li[1] Ann Franchesca Laguna[1] Dayane Reis[1]
Xunzhao Yin[2*] Michael Niemier[1] X. Sharon Hu[1*]
[1]Department of Computer Science and Engineering, University of Notre Dame, USA
[2]College of Information Science & Electronic Engineering, Zhejiang University, China
[*]Corresponding authors emails: xzyin1@zju.edu.cn, shu@nd.edu

ABSTRACT

Recommendation systems (RecSys) suggest items to users by predicting their preferences based on historical data. Typical RecSys handle large embedding tables and many embedding table related operations. The memory size and bandwidth of the conventional computer architecture restrict the performance of RecSys. This work proposes an in-memory-computing (IMC) architecture (iMARS) for accelerating the filtering and ranking stages of deep neural network-based RecSys. iMARS leverages IMC-friendly embedding tables implemented inside a ferroelectric FET based IMC fabric. Circuit-level and system-level evaluation show that iMARS achieves 16.8× (713×) end-to-end latency (energy) improvement compared to the GPU counterpart for the MovieLens dataset.

1 INTRODUCTION

Recommendation systems (RecSys) have been used in various applications to suggest items such as movies, music, books, shopping items, websites, etc. based on a user's previous behavior, as well as historical data from other users. The large amount of available data allows RecSys to leverage deep neural networks (DNN). DNN-based RecSys have been widely adopted by companies like Facebook [1] and Google [2] to improve their online services.

The typical DNN-based RecSys inference process [2] includes two stages: *filtering* and *ranking*. In the filtering stage, the system selects a set of candidate items from a large item database to be recommended based on a user's behavior. The ranking stage then computes the probability of choosing each candidate item. The items with the highest probability are finally returned to the user. Both the filtering and ranking stage use embedding tables (ETs) to capture and store user behaviors and item characteristics. The large ETs make the operations on them memory-bandwidth limited.

Several algorithm and hardware solutions, e.g., ET compression [3], have been proposed to alleviate the memory-bandwidth bottleneck. Near-memory computing has also been leveraged to solve the memory-bandwidth problem by bringing the compute units closer to the memory [4]. However, these solutions only focus on the ranking stage and not the filtering stage which are still impeded by memory bottlenecks due to the huge amount of data transfers.

In-memory-computing (IMC) is a computational paradigm that can alleviate the data transfer overhead between the memory and the compute unit by performing logic and arithmetic operations inside the memory unit itself. Different IMC kernels have been proposed, such as content addressable memories (CAMs), crossbars, general purpose computing-in-memory (GPCiM) and configurable memory arrays (CMAs). CAMs [5] can perform parallel content-based searches in the memory itself, crossbar arrays [6] can perform matrix-vector multiplications, and GPCiM [7] performs Boolean logic and arithmetic operations in memory. CMAs combine the functionalities of random access memory (RAM), CAM, GPCiM and crossbar in a single memory array. Ferroelectric field-effect transistors (FeFET) based crossbars, CAMS, GPCiM and CMAs have been designed [8, 9] and have shown that FeFET-based circuits are denser and faster than CMOS-based and ReRAM-based circuits. FeFETs can be easily integrated with the CMOS fabrication process and large-scale FeFET memories have also been fabricated in [10].

In this paper, we propose iMARS, an IMC architecture for RecSys. iMARS exploits an FeFET-based CMA fabric that combines the functionalities of RAM, ternary CAMs (TCAMs) and GPCiM. The FeFET-based CMA (from [9]) can switch its functionality to implement (i) TCAM-based searches to realize nearest neighbor search (NNS) in the filtering stage; and (ii) GPCiM-based arithmetic logic to implement the additions/accumulations in the filtering and ranking stage. **Specific contributions of our work include (1) an integrated IMC fabric to simultaneously accelerate the filtering and ranking stages; (2) an IMC-friendly computation flow to facilitate mapping the RecSys algorithms to iMARS; (3) support for all ET related operations *in memory* by combining TCAM and GPCiM functionality in the CMA fabric; (4) a two-level memory hierarchy and corresponding in-memory adder trees to store the large ETs.**

We have evaluated the latency and energy benefits of iMARS based on two widely used RecSys models: YoutubeDNN [2] on the MovieLens dataset [11] and Facebook DLRM [1] on the Criteo Kaggle dataset. The results show that for the MovieLens dataset, iMARS achieves a 16.8× (713×) end-to-end speedup (energy improvement) against the GPU implementation. For the Criteo Kaggle dataset which is widely used for the ranking task, the ranking model in Facebook DLRM is accelerated with iMARS, which leads to 13.2× (57.8×) improvement in latency (energy) improvement.

2 BACKGROUND

In this section, we review the basics of DNN-based RecSys and related work on hardware accelerators for RecSys. Furthermore, we discuss the IMC circuits (TCAMs, GPCiMs, CMAs and crossbars) and technologies (i.e., CMOS, FeFETs) employed in iMARS.

2.1 Recommendation Systems

RecSys are composed of a filtering and a ranking stage [2]. The filtering stage (Fig. 1(a)) aims to reduce the number of computations needed in the ranking stage by determining a set of candidate items (e.g. $O(100)$) from the entire item database (e.g. $O(10^6)$). The ranking stage (Fig. 1(b)) aims to find the item with the highest score for a specific user from the candidate items. The filtering stage uses a DNN to characterize the user behavior as a single embedding vector.

DAC '22, July 10–14, 2022, San Francisco, CA, USA
2022. https://doi.org/10.1145/3489517.3530459

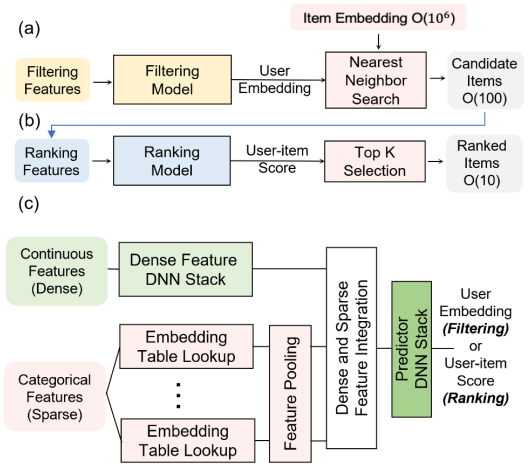

Figure 1: Configuration of the (a) filtering and (b) ranking stages. (c) General DNN model used in the filtering and ranking stages.

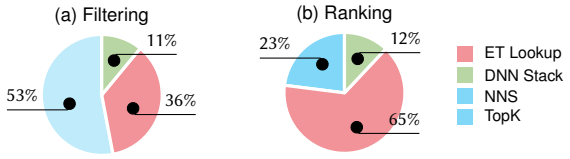

Figure 2: Operation breakdown of the filtering and ranking stages on the MovieLens dataset.

Based on this user behavior, the candidate items to recommend can be found by using the NNS on the item ET as shown in Fig. 1(a). The goal of the ranking stage is to evaluate each user-item pair and predict the score of each candidate item for a specific user. The score is defined as the click-through rate (CTR) and indicates the likelihood that an item will be clicked. Based on the scores, the top-k items ($O(10)$) are returned to the user (Fig. 1(b)).

Both DNN models in the filtering and ranking stages follow the configuration depicted in Fig. 1(c). DNNs are employed to generate the embedding vector representing the user behavior or predicting user-item pair scores. The models take advantage of both continuous (dense) and categorical (sparse) features. Dense features can be directly processed by a DNN while sparse features are captured by large ETs with sparse lookup and pooling operations. These ET operations (lookups, NNS) contribute a significant portion of the run time in RecSys as shown in the operation breakdown of the MovieLens dataset using the YouTubeDNN RecSys [2] in Fig. 2.

Existing efforts on accelerating RecSys include field programmable gate array (FPGA)-based accelerators. E.g., FleetRec [12] and MicroRec [13] alleviate the memory-bandwidth bottleneck of the ETs in RecSys by using FPGAs with high bandwidth memory. Near-memory computing has also been considered to alleviate the memory bottleneck. RecNMP [4] uses a dual in-line memory module-based near-memory computing that can support sparse embedding models. Most of these existing hardware accelerators only focus on a single aspect of RecSys, i.e., ranking, filtering, DNNs or ETs. Though it is possible to simply cascade these accelerators to implement RecSys, such a simple-minded approach results in higher hardware cost due to duplicated components.

2.2 In-Memory Computing Circuits

IMC circuits can alleviate the memory bottleneck and provide parallelism to RecSys operations. This section discusses different types of IMC circuits such as TCAMs, crossbars, GPCiMs and CMA.

TCAMs enable parallel searches based on the Hamming distance for a query against a large stored database in $O(1)$ time [5]. Using the threshold-match mode of the TCAMs, we can retrieve the row entries in the array nearest to the query under the threshold distance by parallel searches. In a TCAM array with r rows and c columns, the TCAM cells in a row are serially connected through a common matchline. Each TCAM cell, s_{ij}, performs an XOR operation between bit j of query Q and the bit j stored at s_{ij}. Each matchline implements a logic AND of all the cells connected to it.

Crossbars are an IMC structure where every input is connected to every output through cross-points that consist of memory elements and selectors. Crossbars can efficiently implement matrix-vector multiplications, and are thus ideal accelerators for DNN models such as convolutional neural networks [14].

GPCiMs [7, 15] perform general-purpose Boolean logic and arithmetic operations inside RAM. GPCiMs can employ specialized memory cells based on emerging technologies to perform current-based operations inside a memory array (e.g., [7]). Alternatively, customized memory peripherals (such as sense amplifiers) can be designed to operate with two memory words that are simultaneously selected by row decoders. In GPCiMs that employ customized sensed amplifiers, the voltage (or current flow) through a column-connected bitline is sensed and compared to one (or multiple) reference(s) so the results of Boolean logic/arithmetic can be produced.

CMAs [8, 9, 15] combine multiple IMC functionalities in the same physical structure. For instance, CMAs can work as either TCAM or GPCiM units at distinct times. Note that conventional TCAMs perform row-wise sensing as matchlines are placed along the row direction. GPCiMs, on the other hand, require the voltage drop (or current flow) through the vertically-connected bitlines to be sensed and compared to one (or more) reference(s) in order to produce the results needed for general-purpose computation. Due to this difference in the TCAM and GPCiM arrays, combining them in a single hardware structure requires additional memory peripherals to achieve re-configurability [8, 9].

Storing RecSys ETs requires significant amount of memory and substantial communication between memory and processing units, which could be alleviated by IMC architectures. The use of CMAs based on emerging technologies can be beneficial for implementing IMC-based RecSys compared to standard CMOS CMAs [15] due to the increased density of memory cells and lower standby power (a result of the device's non-volatility).

FeFETs have been used in various IMC based circuits such as CAMs, GPCiMs and CMAs [16, 17]. Previous work has demonstrated the benefits of FeFET-based CMAs over other emerging technologies such as ReRAMs [8, 9]. FeFETs have similar structure as metal-oxide-semiconductor field-effect transistors (MOSFETs) used in standard CMOS silicon, except a layer of FE oxide is deposited in the transistor's gate stack. Because of this, FeFETs are compatible with the CMOS fabrication process [10] and large-scale FeFET memories have been demonstrated [10]. For these reasons, we employ a FeFET-based CMA design for accelerating RecSys.

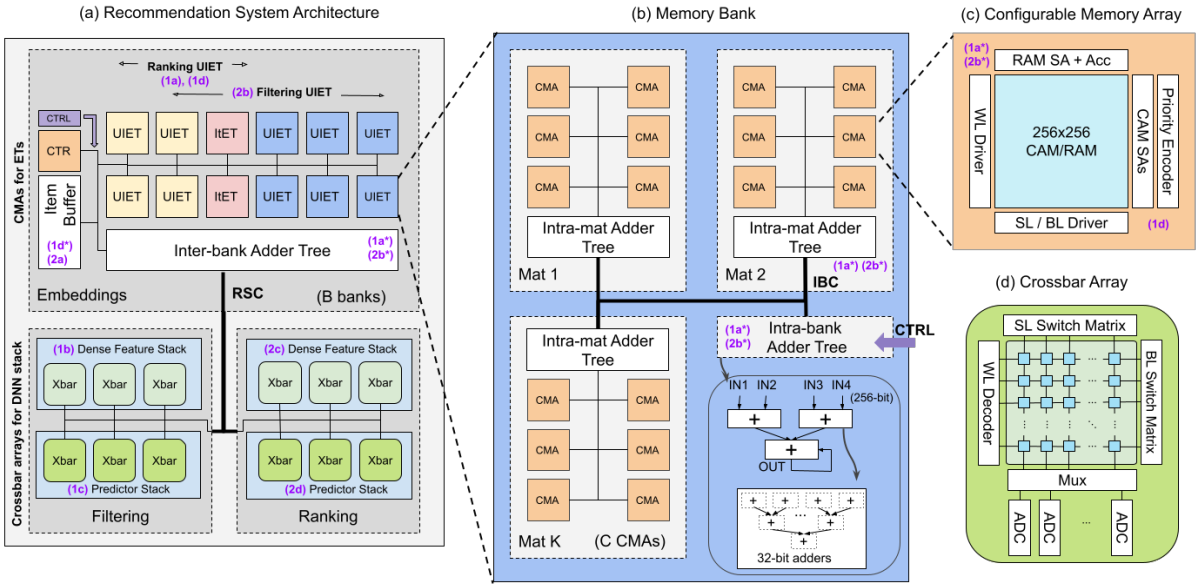

Figure 3: (a) Proposed iMARS architecture. (b) CMA bank for item embedding table (ItET) and user-item embedding table (UIETs). (c) CMA structure, which is capable of switching between the CAM/RAM/GPCiM modes. (d) Crossbar array. Labels (1a), ..., (2d*) shows the computation flow of the ranking and filtering stages as discussed in Sec. 3.3.

3 IMC ACCELERATOR FOR RECOMMENDATION SYSTEMS

We propose iMARS, an architecture for RecSys that uses an IMC fabric to accelerate the DNN stack and ET related operations in both the filtering and ranking stage of RecSys. For the DNN stack, crossbars can be readily leveraged. For ET related operations, we leverage CMAs and adder trees placed in the memory periphery. Since different ETs play different roles depending on the stage, how to organize these tables inside the IMC fabric must be considered carefully. We elaborate our proposed architecture in Sec. 3.1 and computation mapping in Sec. 3.3.

3.1 iMARS Architecture

The iMARS architecture, shown in Fig. 3(a), consists of two types of IMC arrays: Crossbar arrays for the DNN stack (bottom) and CMA arrays for the ETs (top). The implementation of the DNN and the ETs inside iMARS is described below.

3.1.1 Embedding Tables. Some sparse features are used in both the ranking and filtering stage and hence can share the same ETs. We employ two different ETs: user-item embedding table (UIET) and the item embedding table (ItET). The UIET holds user-item features used by the filtering and ranking stage. Some UIETs are exclusively used by the filtering or the ranking stage while some UIETs are shared by the two stages. The item characteristics are stored in the ItET. To conserve memory space, the ItET can be accessed by both the filtering and the ranking stage.

For the UIETs, ET lookups and pooling operations are done in the filtering and ranking stage, respectively. For the ItET, two primary operations are required: (i) lookups and pooling on the ET to transform a sparse input feature into a dense vector; (ii) the NNS on the ET to return the candidate item IDs in the filtering stage. The iMARS architecture implements the two types of ETs inside its IMC fabric. CMAs integrate reads, searches, and in-memory logic/arithmetic operations in a single array, which makes them a suitable choice for implementing UIETs and ItETs.

While the circuits inside a CMA are described in [9], we introduce below the design of a novel, two-level hierarchy based on CMAs and adder trees to store and efficiently compute on the large number of items in the ETs as needed by the RecSys.

Hierarchical CMAs: To accomodate the large ETs, B banks of CMAs are deployed in iMARS. Fig. 3(b) depicts the structure of one CMA bank, which consists of M mats (labeled as Mat-1, Mat-2, ..., Mat-M). Each mat is comprised of C CMAs that work independently as the IMC engines in iMARS for performing lookups, searches and additions. An individual CMA is depicted in Fig. 3(c). It employs CAM sense amplifiers (SAs) based on a preset threshold, as well as searchline (SL) drivers and priority encoders to perform threshold based NNS. We chose to implement threshold based matching in the CAM based on a reference current generated by a dummy 1T+1FeFET cell, which can be adjusted to compensate for process variations or to change the sensitivity of the Hamming distance in the NNS operation. The RAM SAs, wordline (WL) and bitline (BL) drivers are used during lookups. Pooling operations are performed with in-memory additions (through an accumulator placed next to the RAM SA). More details on the CMA structure can be found in [9]. FeFET-based CMAs [9] are utilized to implement the ETs. When compared to CMOS-based counterparts, FeFET-based CMAs have higher density [8].

Adder trees: To support the accumulation of a large number of parameters, we develop a hierarchical, adder tree structure. Specifically, iMARS uses in-memory addition to sum, in a single memory array, embeddings comprised of 32 dimensions with int-8 quantization. To accumulate (sum up) the outputs of the CMAs for each mat, iMARS sums up C 256-bit (i.e., 32×8)-bit numbers leveraging a near-memory, 256-bit intra-mat adder tree placed in each mat.

Different mats can perform intra-mat additions in parallel. Once the intra-mat additions are completed, their results are accumulated across the K mats to produce a single 256-bit result (one output per memory bank). iMARS supports the addition of four 256-bit inputs inside a near-memory "Intra-bank Adder Tree" in one shot (bottom right of Fig. 3(b)). In other words, we design an "Intra-bank Adder Tree" with a fan-in of 4, a design choice made as a compromise between area footprint of the iMARS banks and performance of the intra-bank addition. In cases where more inputs need to be accumulated (i.e., when $K > 4$), multiple rounds of addition are needed using the same "Intra-bank Adder Tree".

The aforementioned design parameters B, M and C largely impact the area, capacity and the performance of iMARS. First, area footprint increases proportionally to B, M and C. Larger B, M and C increase the capacity of iMARS for storing embeddings. High capacity enables iMARS to accomodate big workloads. However, a large C implies a large fan-in for the "Intra-mat Adder Tree" inside each mat, which leads to parasitic effects that increases the delay for aggregating the outputs of multiple CMAs. A large K, in turn, implies that more mats are connected to (and sharing) the same communication bus, which increases the overall latency of the RecSys (to be discussed in Sec. 3.1.3).

3.1.2 DNN Stack. The DNN stack requires matrix-vector multiplications which can be implemented using crossbar arrays in iMARS (Fig. 3(a)). Two dedicated crossbar banks are employed to execute the ranking and the filtering DNN stack composed of fully connected layers. Each crossbar bank contains multiple crossbar arrays (Fig. 3(d)) in order to accomodate the respective DNN model. These crossbar banks both hold the DNN Stack to obtain dense features and the DNN stack that returns a user embedding during the filtering stage or a user item-score for the ranking stage. Crossbar arrays can leverage the FeFET technology [18].

3.1.3 Communication inside iMARS. Data among the different hardware components of iMARS need to be communicated. To ensure such communication does not incur too much overhead, we carefully design communication channels inside iMARS. There are two types of communication in iMARS: (1) communication among the different functional blocks and (2) communication between the mats in each CMA bank. While (1) leverages the RecSys communication (RSC) bus, (2) occurs through the intra-bank communication (IBC) network. The RSC bus and the IBC network are depicted in Fig. 3(a) and (b), respectively.

The RSC bus enables the exchange of inputs/outputs through the different hardware blocks in iMARS. While data traffic between the different hardware blocks is essential to the system's overall functionality, the data traffic through the RSC bus is not as intense as the traffic inside the memory banks that store the ETs. Data communication on the IBC network and the RSC bus is serialized to minimize the wiring overhead (thus, reducing the area of iMARS). iMARS is designed for a RSC bus with 256-bit capacity. The IBC, on the other hand, supports the transmission of 128 bytes of data (i.e., four 256-bit inputs) to be added up inside the intra-bank adder tree (bottom right of Fig. 3(b)) in one shot. Data communication on the IBC network is serialized when K >4 (the fan-in of the "Intra-bank Adder Tree"). The IBC network capacity and the number of "Intra-bank Adder Tree" fan-ins are design choices that must take into consideration the impact on area footprint, delay and energy. For instance, extremely wide buses may be impractical as they require too much area to be implemented. On the other hand, a narrow IBC would require many data fetches through the bus across K mats. The overhead of communication with the RSC bus/IBC network is accounted for in the results reported in Sec. 4.

Data traffic inside the RSC and the IBC is orchestrated by a controller circuit (indicated by the box labeled CTRL in Fig. 3(a)). The controller circuit consists of a clock generator and two counters that keep track of (i) the activated bank, and (ii) the mats inside the bank that are sending outputs for accumulation with the "Intra-bank Adder Tree" circuit. Data packets always travel through the IBC in a predetermined order, as defined by the counters (i.e., in Bank B, from Mat-1, Mat-2, ..., Mat-M in groups of four outputs, i.e., 128 bytes). The pre-defined communication pattern reduces conflicting accesses and eliminates the need for routers.

3.2 Embedding Table Mapping

The iMARS architecture uses CMAs to store ETs., which have varying number of entries (typically 3-30,000 entries). Each row on the CMA represents an entry of an ET. Designing CMAs with varying sizes is not practical. We determined the optimal array-level CMA to be the size of 256×256 based on circuit-level simulations. Some of the ETs can fit in a single CMA and some require multiple CMAs. The number of CMAs needed to store an ET is n/R where n is the number of entries in the ET and R is the number of rows in the CMA. If $n/R < C$, we only need one mat, otherwise the number of mats needed to be activated is equal to $n/(RC)$. Each sparse feature is mapped to a separate bank. Hence, the number of activated banks depends on the number of sparse features.

We quantize all ETs to 8-bit integer precision to reduce the memory requirement. We also replace the cosine-distance based NNS in the original filtering stage with the IMC-friendly Hamming-distance based NNS. To facilitate the Hamming distance search, we employ a locality-sensitive hashing (LSH) technique on the ItET [5]. Each row of the ItET includes the additional bits for storing the corresponding LSH values. Finally, a *fixed-radius near neighbor search* instead of top-k search is employed. The fixed-radius near neighbor search is amenable to the threshold-based match offered by the TCAM implementation, and reduces the total number of required operations. Algorithm-level evaluation will be presented in Sec. 4 to study the effects of the adjustment on accuracy. We use a 256 LSH signature length which requires 2 CMAs to store a single entry.

3.3 RecSys Operation Mapping

Given the iMARS architecture, careful mapping of the computation, how in Fig. 1 to iMARS is required in order to design the control sequence properly. The crossbars stored the trained weights of the DNN stack and the ItETs and UIETs store the trained ETs for the sparse feature vectors. Fig. 3 illustrates such a mapping where the labels (1a)-(2d*) indicates. We first discuss the operations in the filtering stage. **(1a)** The sparse features are sent to the corresponding ETs, i.e., UIETs and ItET for lookups and pooling. The embedding vectors of the features are obtained by looking up the stored ETs in the ItET CMAs and UIET CMAs using the RAM mode of the CMA. The retrieved embeddings are then aggregated (indicated by **(1b*)** in Fig. 3) by the in-memory adder, intra-mat and intra-bank adder trees. **(1b)** the dense features are sent to the pre-trained filtering sparse feature DNN stack which is implemented with the crossbar arrays. **(1c)** All features are then sent to the filtering DNN. The output of the filtering DNN is a user embedding vector (u_i in Fig. 3). **(1d)** The user embedding vector is then sent to the ItET to retrieve

the N nearest neighbors as candidate items inside the ItET CMAs. The indices of these retrieved (i.e., candidate) item embeddings are then stored in the item buffer (See **(1d*)** inf Fig. 3).

We now discuss the operations in the ranking stage. **(2a)** Each candidate item in the item buffer is then analyzed with respect to the user's preferences. **(2b)** The corresponding item embeddings are retrieved from the stored ET in the ItET and the ranking embeddings are retrieved in the stored ET in the ranking UIETs using the RAM mode. Note that some UIETs used in the filtering mode can be shared with the ranking stage. The item embeddings are pooled with the ranking embeddings either by concatenation or by an ADD operation using the in-memory adder, intra-bank and inter-bank adder trees. (See (2b*) in Fig. 3) The output forms a new set of embedding features. Together with the dense features are fed to the ranking DNN stack implemented in crossbars. **(2c)** the dense features are again obtained from the trained ranking DNN stack. **(2d)** The remaining crossbar arrays implement the ranking DNN with the pooled feature embeddings as input and return the click-through-rate (CTR) to the CTR buffer. The CTR buffer is a CMA that stores the CTR for each candidate item and the item index which are used for selecting the final top-k items. **(2e)** The CTR buffer then performs a top-k operation using the threshold match mode of the CMA by searching a vector of all 1's (the maximum allowable CMA input).

4 EVALUATION

We evaluated the RecSys implementations with two RecSys instances: (1) YoutubeDNN model [2] on the MovieLens 1M dataset, which includes both the filtering and ranking stage; (2) DLRM model [1] targeting at the ranking stage on the Criteo Kaggle dataset. The configurations of the two RecSys shown in Table 1 were implemented on Nvidia RTX 1080 GPU. We used the tools Nvidia-smi and lineprofiler to obtain energy and latency, respectively. We only compared with the GPU evaluation as other accelerators use older RecSys models and datasets.

We dimension B, M and C based on the largest dataset used in our evaluation (i.e., the Criteo Kaggle) with the configuration shown in Table. 1. Each ET have different number of entries. In this configuration, the maximum size of the ETs in the Criteo Kaggle is 30,000 entries. Since each CMA has 256 rows, 118 CMAs are required to store the embedding table. The number of arrays is rounded up to the nearest power-of-two value, i.e., 128. We choose C=32, which corresponds to 4 mats (M=4) working in parallel inside each bank, to have a balance between storing small and large ETs. The outputs from the 4 mats are accumulated at the bank level. Finally, the Criteo Kaggle dataset has 26 sparse features for ranking. Hence, we dimension iMARS with 32 banks (B=32) to accomodate all these features. The other features are also mapped accordingly, with some mats and CMAs deactivated in a bank according to the size of the ET. We use 26 activated banks, 104 activated mats and 2860 activated CMAs for the Criteo Kaggle dataset.

For the MovieLens dataset (also used in our evaluation), due to the much smaller number of rows per ET, we are still able to use the same architecture while keeping idle arrays deactivated. The MovieLens dataset uses 5 UIET for the filtering stage and 6 UIET for the ranking stage (Table 1), 5 of which are shared between the filtering and ranking stages. ETs have a maximum of 6040 entries and a minimum of 3 entries. We use 7 active banks, 8 active mat and 54 active CMA in the MovieLens dataset.

Table 1: RecSys configurations and memory mapping on iMARS

	Movielens		**Criteo Kaggle**	
Model	Youtube	Youtube	DLRM	
Stage	Filtering	Ranking	Ranking	
DNN Network	128-64-32	128-1	Bottom MLP 256-128-32	Top MLP 256-64-1
# UIET (Shared)	5 (5)	6 (5)	26	
# ItET	1		/	
# Row per ET	3000		28000	
# Bank	7		26	
# Mat	8		104	
# CMA	54		2860	

Table 2: Array-level evaluation of CMA, adder-trees and crossbars.

Component	**Operation**	**Energy (pJ)**	**Latency (ns)**
256×256 CMA	Write	49.1	10.0
	Read	3.2	0.3
	Addition	108.0	8.1
	Search	13.8	0.2
Intra-mat adder tree	256-bit Add	137.0	14.7
Intra-bank adder tree	256-bit Add	956.0	44.2
256×128 Crossbar	MatMul	13.8	225.0

4.1 Array-level Evaluation

We have designed the complete circuit of a 256×256 FeFET-based CMA, and simulated it in HSPICE by employing a Preisach based model for FeFETs [19] along with the CMOS Predictive Technology Model (PTM) from [20] with a 45nm technology node. In our evaluation, besides the memory cells inside each CMA, we also consider all the peripherals depicted in Fig. 3(c). The adder trees and communication network are implemented in Verilog and synthesized with Cadence Encounter RTL Compiler v14.10, with the NanGate 45nm open-cell library [21]. The crossbars are evaluated by the Neurosim tool [22] using a 45nm FeFET model. Table 2 summarizes the array-level figures-of-merit (FoM) for the different types of accesses supported by the CMA. Table 2 also includes the FoM for a 256×128 crossbar. These values are used for higher-level evaluations.

4.2 Accuracy Evaluation

We examined the algorithm-level performance (i.e., accuracy) of RecSys when using quantized data representation and when using different distance functions. We implemented a YoutubeDNN filtering model [2] on the MovieLens 1M dataset [11], where a FAISS-based distance search is used. We use the hit rate (HR), the # of hits (i.e., correct predictions) divided by the # of test users, as the accuracy metric. Three configurations are tested: (1) 32-bit floating-point (FP32) representation and cosine distance (2)8-bit Int and cosine distance (3) 8-bit Int and LSH-based Hamming distance and achieve HR to be 26.8% / 26.2% /20.8%, respectively. iMARS incurs around 5.4% accuracy degradation, which indicates that the distance function plays an important role in the accuracy. However, since the filtering stage only provides a coarse selection of the candidate items, such accuracy loss is tolerable as the accuracy is retained by the ranking stage.

4.3 Energy and Latency Evaluation

We estimated the energy and latency of iMARS based on the mapping and the simulated array-level FoM. As the latency and energy improvement of TCAM arrays and crossbars are well studied in previous work [5] [22], in this section we mainly focus on our findings

Table 3: ET operation comparison between the GPU and iMARS

Dataset		**MovieLens**		**Kaggle**
Stage		Filtering	Ranking	Ranking
Latency	GPU	9.27μs	9.60 μs	14.97 μs
	iMARS	0.21 μs	0.21 μs	0.24 μs
	Speedup	43.61×	45.17×	61.83×
Energy	GPU	203.97 μJ	211.26 μJ	329.34 μJ
	iMARS	0.40 μJ	0.46 μJ	6.88 μJ
	Reduction	516.05×	458.12×	47.90×

on ET lookup operations, which is also a bottleneck of the RecSys. We compare the latency and energy of TCAM-based LSH search with the LSH search on GPU as well as the original cosine search on GPU. Finally we report the end-to-end system comparison.

4.3.1 ET lookup operation. Table 3 shows the latency and energy consumption of the ET lookup operation in the two RecSys instances as well as on GPU. All the data are obtained for one item input. The ET lookup operation in iMARS includes the multiple lookups of CMAs, the intra-mat addition and intra-bank addition. To estimate the latency and energy on iMARS, we consider the worst case that all lookups for one ET happen in the same array. Multiple lookups in one array requires multiple read, write and in-memory add operations, incurring higher latency and energy. We also include the latency/energy overhead of communication due to wiring and serialization with the RSC bus/IBC network. Thus, under the aforesaid worst case, the iMARS takes 0.24 μs latency and 6.88 μJ energy for a single input on the Criteo Kaggle dataset and achieves 61.83× latency and 47.9× energy improvement. For the MovieLens dataset, the iMARS achieves 43.1×/45.6× speedup and 516.05×/458.12× energy reduction over the GPU counterpart on the filtering/ranking stages.

From Table 3, it can be seen that on both GPU and iMARS, the ranking stage takes more time and energy than the filtering stage for MovieLens because the ranking stage deploys one more ET than the filtering stage as the memory mapping shown in Table 1. Also, the latency and energy for the MovieLens dataset is smaller than another dataset because of the relatively small ET size. These improvements are attributed to the fact that iMARS reduces the data movement between the processor and memory by using in-memory ET lookup. Also the adoption of the FeFET technology contributes to part of the improvement.

4.3.2 NNS operation. NN Search operations are needed in the filtering stage, and are realized by configuring CMAs to the CAM mode in iMARS. The utilization of the CAM search mode enables the NNS operation to be implemented in $O(1)$ time instead of $O(n)$. For the filtering stage on the MovieLens dataset with $O(10^3)$ items, the search latency using the original cosine distance on the GPU is around 13.6 μs and it consumes 0.34 mJ for one input. With LSH search with 256 signature length, the GPU spends 6.97 μs and 0.15 mJ. The latency and energy improvement over the GPU counterpart (LSH search) is 3.8e4× and 2.8e4× as shown in Table 2.

4.3.3 End-to-End. We compare the end-to-end improvements of iMARS over GPU in this section. For the GPU data, we only count DNN stack, ET lookup and NNS operation in the algorithm. For iMARS, the ET lookup operation and NNS operation are evaluated as we discussed before. The DNN stack is evaluated using Neurosim [22] (FoM shown in Table 2), which brings around 2.69× latency improvement compared to the GPU counterpart.

For the ranking model on the Criteo Kaggle dataset, iMARS achieves 13.2× latency improvement and 57.8× energy improvement over GPU. For the filtering and ranking stage together, iMARS achieves 16.8× and 713× latency/energy end-to-end improvement on the MovieLens dataset. That is, it can achieve 22025 queries/second compared with the 1311 queries/second on the GPU. The end-to-end improvement is dominated by the ranking stage because each user only goes through the filtering stage once in iMARS. However, for each user, the CTR needs to be calculated for each candidate item during the ranking stage.

5 CONCLUSION

We present iMARS, an IMC-based accelerator for recommendation systems. iMARS uses hierarchical IMC fabric consisting of both crossbars and CMAs to accelerate both the filtering and ranking stages of RecSys. We introduce an IMC-friendly embedding table organization and judicious computation mapping to maximize the benefit offered by IMC. iMARS achieves 22025 queries/second over 1311 queries/second on the GPU (16.8× speedup for the MovieLens dataset). Also, 713× end-to-end energy reduction compared with GPU is achieved by iMARS.

ACKNOWLEDGMENTS

This work was supported in part by NSF CCF-2028879 and CCF-1640081, and by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] M. Naumov, et al. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.
[2] P. Covington, et al. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, 2016.
[3] H.-J. M. Shi, et al. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *SIGKDD*, 2020.
[4] L. Ke, et al. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th ISCA*, pages 790–803. IEEE, 2020.
[5] K. Ni, et al. Ferroelectric ternary content-addressable memory for one-shot learning. *Nature Electronics*, 2(11):521–529, 2019.
[6] A. Ranjan, et al. X-mann: A crossbar based architecture for memory augmented neural networks. In *DAC*, pages 1–6, 2019.
[7] D. Reis, et al. Computing in memory with fefets. In *ISLPED*, 2018.
[8] X. Zhang, et al. FeMAT: Exploring In-Memory Processing in Multifunctional FeFET-Based Memory Array. In *ICCD*, pages 541–549, 2019.
[9] D. Reis, et al. Attention-in-Memory for Few-Shot Learning with Configurable Ferroelectric FET Arrays. In *26th ASPDAC*.
[10] S. Dunkel, et al. A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond. In *IEDM*, 2017.
[11] F. M. Harper et al. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.
[12] W. Jiang, et al. Fleetrec: Large-scale recommendation inference on hybrid gpu-fpga clusters. In *27th ACM SIGKDD*, 2021.
[13] W. Jiang, et al. Microrec: Efficient recommendation inference by hardware and data structure solutions, 2021.
[14] A. Shafiee, et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ACM/IEEE 43rd ISCA*, pages 14–26, 2016.
[15] S. Jeloka, et al. A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory. *JSSC*, 51(4), 2016.
[16] X. Chen, et al. The impact of ferroelectric fets on digital and analog circuits and architectures. *IEEE Design & Test*, 37(1):79–99, 2019.
[17] S. Beyer, et al. Fefet: A versatile cmos compatible device with game-changing potential. In *2020 IEEE International Memory Workshop (IMW)*, pages 1–4, 2020.
[18] Y. Luo, et al. Mlp+neurosimv3.0: Improving on-chip learning performance with device to algorithm optimizations. In *ISCA*, ICONS '19, New York, NY, USA, 2019. Association for Computing Machinery.
[19] K. Ni, et al. A circuit compatible accurate compact model for Ferroelectric FETs. In *VLSI Symposium*. IEEE, 2018.
[20] Y. Cao, et al. Predictive technology model. *Internet: http://ptm. asu. edu*, 2002.
[21] J. Knudsen. Nangate 45nm open cell library. *CDNLive, EMEA*, 2008.
[22] P.-Y. Chen, et al. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *TCAD*, 37(12):3067–3080, 2018.