RecPipe: Co-designing Models and Hardware to Jointly Optimize Recommendation Quality and Performance

Udit Gupta
Harvard University
Facebook AI Research
Cambridge, MA, United States
ugupta@g.harvard.edu

Samuel Hsia
Harvard University
Cambridge, MA, United States
shsia@g.harvard.edu

Jeff (Jun) Zhang
Harvard University
Cambridge, MA, United States
jeffzhang@g.harvard.edu

Mark Wilkening
Harvard University
Cambridge, MA, United States
wilkening@g.harvard.edu

Javin Pombra
Harvard University
Cambridge, MA, United States
javinpombra@college.harvard.edu

Hsien-Hsin S. Lee
Facebook AI Research
Cambridge, MA, United States
leehs@fb.com

Gu-Yeon Wei
Harvard University
Cambridge, MA, United States
gywei@g.harvard.edu

Carole-Jean Wu
Facebook AI Research
Cambridge, MA, United States
carolejeanwu@fb.com

David Brooks
Harvard University
Cambridge, MA, United States
dbrooks@eecs.harvard.edu

ABSTRACT

Deep learning recommendation systems must provide high quality, personalized content under strict tail-latency targets and high system loads. This paper presents RecPipe, a system to jointly optimize recommendation quality and inference performance. Central to RecPipe is decomposing recommendation models into multi-stage pipelines to maintain quality while reducing compute complexity and exposing distinct parallelism opportunities. RecPipe implements an inference scheduler to map multi-stage recommendation engines onto commodity, heterogeneous platforms (e.g., CPUs, GPUs). While the hardware-aware scheduling improves ranking efficiency, the commodity platforms suffer from many limitations requiring specialized hardware. Thus, we design RecPipeAccel (RPAccel), a custom accelerator that jointly optimizes quality, tail-latency, and system throughput. RPAccel is designed specifically to exploit the distinct design space opened via RecPipe. In particular, RPAccel processes queries in sub-batches to pipeline recommendation stages, implements dual static and dynamic embedding caches, a set of top-k filtering units, and a reconfigurable systolic array. Compared to previously proposed specialized recommendation accelerators and at iso-quality, we demonstrate that RPAccel improves latency and throughput by 3× and 6×.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; *hardware*; • **Hardware** → integrated circuits.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480127

KEYWORDS

personalized recommendation, deep learning, datacenter, hardware accelerator

ACM Reference Format:
Udit Gupta, Samuel Hsia, Jeff (Jun) Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin S. Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. 2021. RecPipe: Co-designing Models and Hardware to Jointly Optimize Recommendation Quality and Performance. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3466752.3480127

1 INTRODUCTION

Deep neural network (DNN) based recommendation systems constitute an overwhelming fraction of AI cycles in production data centers (e.g., Facebook, Google, Alibaba) [1, 20, 24, 25, 35, 58, 65–67]. To improve content personalization in a wide range of services (e.g., search, e-commerce, movie and video-streaming, social media), the size of production recommendation models has grown by over 10× between 2017 and 2020 [45, 63, 64].

In response to the dramatic increase in infrastructure demands from the ever-increasing model complexity, system- and architecture-level solutions are customized for DNN-based recommendation, including inference schedulers [18], near memory processing hardware [38, 42, 56], and specialized accelerators [9, 30, 33]. These prior solutions assume fixed models, leaving significant room for efficiency optimization. Evidenced by recent work optimizing DNNs for computer vision and natural language processing [19, 22, 27, 49, 51, 53, 60], co-designing models with hardware is an effective approach. However, the model accuracy requirement for recommendation tasks is stringent [11, 66, 67], making the model-hardware co-design space challenging to navigate.

While accuracy represents a model's ability to predict whether users will like *individual* items, production services are designed to serve users a personalized *collection* of relevant items [5, 32]. As

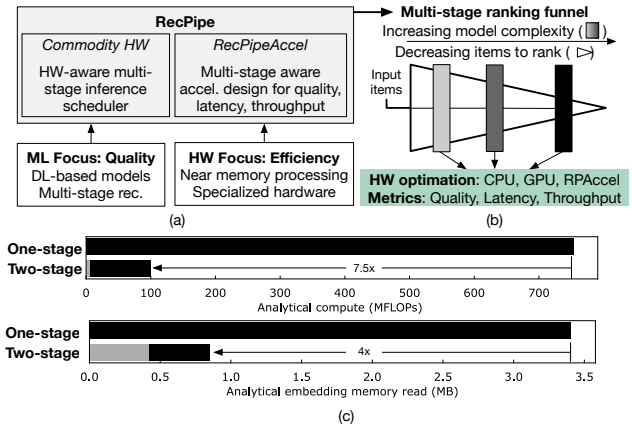

Figure 1: (a) Compared to prior work from machine learning and hardware researchers, this work jointly optimizes quality and performance. (b) RecPipe co-designs models and hardware across multi-stage recommendation pipelines. (c) Transforming monolithic models into multiple stages reduces overall compute demand and embedding memory accesses by 7.5× and 4.0×, respectively.

such, while accuracy is intrinsic to models, *quality* is optimized by improving model accuracy *and* increasing the number of items ranked at the same time. The more holistic, application-level quality objective allows system architects to judiciously trade off accuracy for performance, opening new design spaces for system optimization.

Ranking all items with complex models is wasteful—only a small portion of items are relevant to individual users. Traditionally, recommendation engines achieve high quality by ranking a large number of input candidate items using complex DNNs. The combination of large input working set sizes and complex models incurs high performance overheads. Alternatively, one can decompose a monolithic ranking model into multiple stages to maintain overall quality at higher performance [31, 37, 65]. By splitting the monolithic model into two, a frontend model coarsely *filters* relevant items while a more accurate backend model finely *ranks* items to serve. Further segmenting the pipeline into additional stages creates a ranking funnel (Figure 1 (b)) where complex models only rank items requiring accurate differentiation. For the Criteo dataset and Facebook's Deep Learning Recommendation Model (DLRM) [36, 47], Figure 1(c) shows that, at iso-quality, compared to single-stage, multi-stage recommendation reduces memory and compute demands by 4.0× and 7.5×, respectively. This system-level view optimizing quality and efficiency motivates a new generation of hardware solutions for multi-stage recommendation.

Driven by this motivation, we propose *RecPipe*, a system to co-design recommendation models and hardware to improve both quality and performance (Figure 1(a)). Frontend stages pair lightweight models (e.g., low compute and memory demands) with large input sizes, exposing *data-parallelism*. Backend stages pair heavyweight models (e.g., billions of FLOPs, many GBs of storage) with small input sizes, exposing *model-parallelism* instead. RecPipe's system solutions exploit these distinct parallelism opportunities to jointly optimize quality, throughput, and tail-latency.

To understand the limits of commodity platforms, RecPipe implements an inference scheduler that maps each recommendation stage across heterogeneous hardware (e.g., CPU, GPU) to maximize performance. We find the optimal mapping depends on the application level targets and underlying hardware. Despite the tight co-design between models and hardware, we find commodity CPU-GPU systems do not fully exploit the benefits of multi-stage recommendation as they suffer from low utilization and high PCIe communication overheads between stages.

To address these limitations, we design *RecPipeAccel* (RPAccel), a specialized accelerator for multi-stage recommendation. Starting with a TPU-like, systolic array-based, recommendation accelerator [30], RPAccel's hardware optimizations improve efficiency at low area and power overheads. First, RPAccel implements a reconfigurable systolic array that allows the hardware to concurrently process models across recommendation stages. RecPipe's inference scheduler provisions the fraction of systolic array resources to devote to frontend and backend stages based on application load, balancing latency and throughput. Next, RPAccel eliminates high PCIe communication overheads to the host processor by implementing multiple on-chip filtering units to identify the top-k user-item interactions between stages. Finally, to overlap frontend and backend query processing, RPAccel breaks queries into sub-batches to pipeline stages and pre-fetch embedding vectors in separate caches.

The main contributions of this work include:

(1) We propose a new system, RecPipe, that enables design space exploration and optimization for multi-stage recommendation inference. The framework integrates data sets (e.g., MovieLens [23], Criteo [36]), models (e.g., neural matrix factorization [26], DLRM [47]), and hardware (e.g., CPU, GPU, simulated accelerators) to study trade-offs among quality, tail-latency, and throughput.
(2) We show designing and efficiently scheduling multi-stage pipelines for available commodity hardware platform reduces tail-latency by 4× and 3× on CPUs and heterogeneous CPU-GPU hardware, respectively.
(3) We design RPAccel, a novel accelerator that exploits the distinct properties of multi-stage recommendation to jointly optimize quality, latency, and throughput. Compared to a state-of-the-art baseline accelerator [30], RPAccel's software and hardware optimizations reduce tail-latency by 3× and increases throughput by 6×, at iso-quality as well as small area (7%) and power (27%) overheads.

2 MOTIVATION: WIDENING DESIGN SPACE BY OPTIMIZING FOR QUALITY OVER ACCURACY ALONE

Prior work on specialized systems for deep learning co-optimizes for model accuracy and run-time efficiency (performance, power, and energy) [19, 22, 27, 49, 51, 53]. For neural recommendation however, hardware designers must go one step further, beyond accuracy, and optimize for quality. In this section, we first describe recommendation model architectures and conduct a model hyperparameter sweep. Then, we introduce the quality metric used in this work, showing the fundamental difference between accuracy and quality.

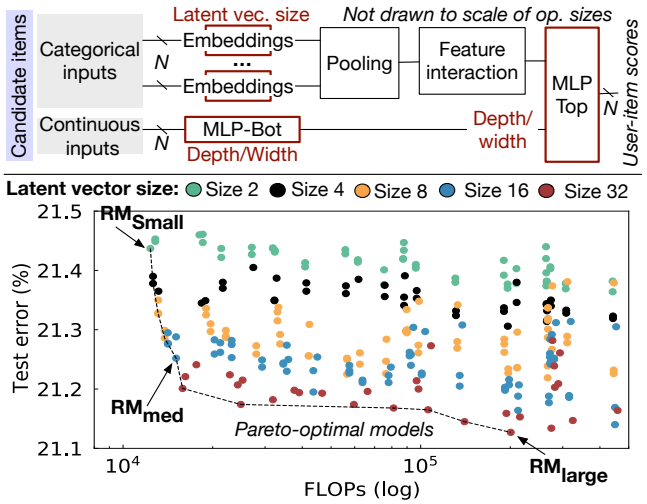

Figure 2: (Top) General recommendation model architecture configured by embedding dimension and MLP size (outlined in red). (Bottom) Hyperparameter sweep shows tradeoff between model complexity and error.

Model	RM_{small}	RM_{med}	RM_{large}
Embedding Dim.	4	16	32
MLP-Bottom	13-64-4	13-64-16	13-512-256-128-64-32
MLP-Top	64-1	64-1	96-1
Model Size	1GB	4GB	8GB
FLOPs	12K	17K	200K
Model Error	21.36%	21.26%	21.13%

Table 1: Pareto-optimal recommendation models.

2.1 Training hyperparameter sweep

Figure 2(top) lays out the general architecture for DNN recommendation models [20, 47]. Continuous input features are processed with DNN layers, e.g. Multi Layer Perceptrons (MLP), while sparse input features are processed using embedding tables. Embedding tables are organized as a collection of embedding vectors with tens to hundreds of latent features. Latent features map sparse inputs to low-dimensional, dense ones. By configuring the main network components (i.e., MLP depth/width, embedding latent vector dimension), highlighted in red, we realize models with varying storage capacity, compute demands, and accuracy.

Figure 2(bottom) shows a hyperparameter sweep by tuning the main network parameters for Facebook's DLRM trained on the Criteo dataset [36, 47]. Increasing the computational complexity of models reduces the test error. Models with 12K FLOPs and 200K FLOPs observe an error of 21.36% and 21.13%, respectively. Note, a 0.23% decrease in error is large given the high sparsity of user-item interactions in recommendation use cases [66, 67]. Recent industry publications show reductions of even 0.1% error greatly improve user experience in real world applications [66, 67]. Table 1 shows the tradeoff between model error and complexity across three Pareto-optimal networks (i.e., RM_{small}, RM_{med}, RM_{large}).

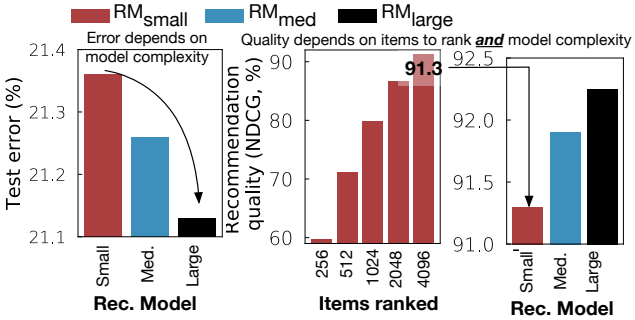

Figure 3: While accuracy depends only on model size (left), recommendation quality depends on number of items ranked (center) *and* model size (right).

2.2 Quality versus accuracy

A model's accuracy represents its ability to correctly predict a user will positively interact with a *single* item. However, in recommendation, models rank thousands of items opening the door for measuring overall quality. Quality measures the relevance of the *entire collection* of items presented to users based on their personal preferences. Following recent work from machine learning and recommendation systems researchers, we use normalized discounted cumulative gain (NDCG) to quantify the quality of the ordered list of output items. NDCG [5, 32] is the ratio between the measured and the ideal ordering, each of which is computed using discounted cumulative gain (DCG): for a list of N items, $DCG = \sum_i^N \frac{Rel_i}{log_2(i+1)}$. Rel_i represents item i's score in the measured or ideal list and $log_2(i+1)$ discounts its relevance—dividing the score by the item's position in the list.

Widening design space. Compared to accuracy, optimizing for quality opens new system design opportunities. For the Criteo dataset, Figure 3 illustrates the impact of varying the number of items ranked (x-axis) and model architecture (i.e., RM_{small}, RM_{med}, RM_{large}) on quality. Empirically, the improvement in quality from increasing number of items ranked overshadows that from larger, more accurate models. Note, the highest quality of 92.25 can be achieved by ranking all 4096 items with RM_{large}. However, as shown in Figure 1, decomposing monolithic models into multiple stages, where small models filter relevant items and large models perform fine-grained ranking, improves computational efficiency at iso-quality. At the frontend, candidate items are coarsely ranked with models that incur memory and compute demands. This reduces the list of candidate items (i.e., working set size) incrementally over the stages. Subsequent stages use larger models for finer-grained ranking. Going beyond accuracy, quality depends on the number of stages, network architectures, and the number of items ranked at run-time: widening the design space to co-optimize performance and quality.

3 RECPIPE DESIGN: A SYSTEM TO OPTIMIZE MULTI-STAGE RECOMMENDATION INFERENCE

We propose RecPipe, a novel system to explore the model- and hardware-level design space to collectively optimize recommendation quality, tail-latency, and system throughput. Figure 4(left)

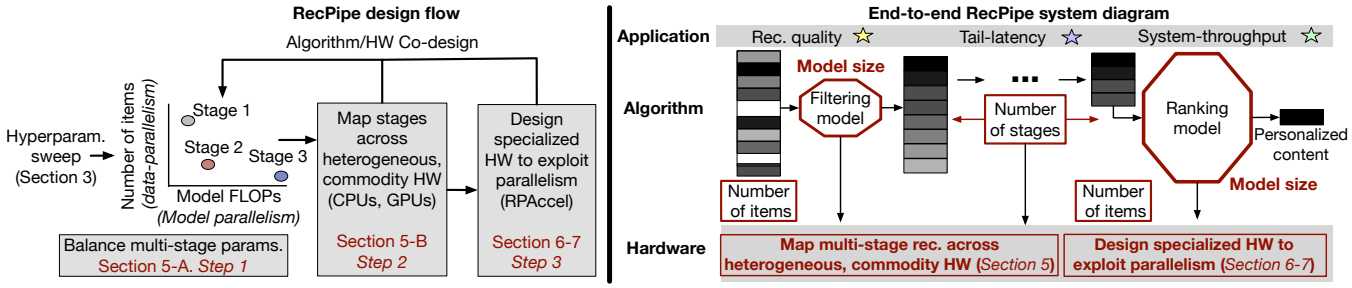

Figure 4: The structure of a multi-stage recommendation pipeline. Highlighted in red, RecPipe explores a variety of recommendation model and hardware infrastructure parameters to balance quality, latency, and throughput.

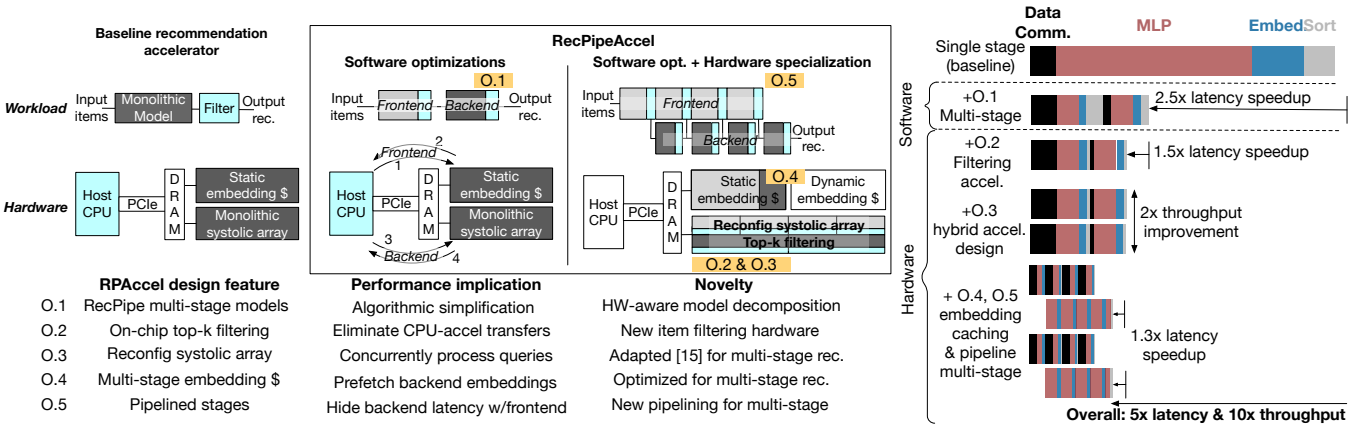

Figure 5: Comparison of baseline recommendation hardware accelerator and RPAccel. We describe RPAccel's five main innovations (i.e., O.1 to O.5) on the left and their performance benefits in the ablation study on the right.

shows RecPipe's multi-step design process. The input to RecPipe is a Pareto-frontier of recommendation models balancing model accuracy and complexity. To co-optimize quality and hardware efficiency on commodity platforms, RecPipe balances multi-stage parameters and statically schedules each stage across available hardware resources (i.e., CPUs and GPUs). Going further, RecPipe exposes distinct parallelism opportunities that are exploited by designing specialized hardware. Figure 4(right) illustrates the multi-stage recommendation pipeline and the design space optimized by RecPipe. The model-level and hardware-level design parameters are highlighted in red. We detail how RecPipe co-designs these parameters to maximize quality and performance below.

3.1 Hardware-aware multi-stage scheduling

RecPipe implements a post-training, inference scheduler customized for multi-stage recommendation. In step 1, RecPipe balances multi-stage modeling parameters. In step 2, RecPipe exploits the parallelism opportunities exposed from step 1, and maps stages across heterogeneous hardware.

Algorithmic scaling (Step 1). RecPipe exhaustively explores the design space of pairing Pareto-optimal recommendation models and number of items to rank at each stage in the multi-stage pipeline. In the frontend, lightweight models are paired with large working set sizes exhibiting high data-level parallelism; in the backend heavyweight models are paired with smaller working set sizes exhibiting high model-level parallelism. By collectively balancing model complexity and input working set size, RecPipe maximizes overall quality under strict latency targets and system loads.

Heterogeneous hardware mapping (Step 2). Given the distinct parallelism opportunities from the aforementioned algorithmic scaling step, RecPipe exhaustively explores the mapping of multi-stage models on available hardware at the stage granularity. We begin by considering commodity hardware platforms i.e., CPUs and GPUs. GPUs implement a highly data-parallel architecture that parallelize individual queries, especially in the frontend with large working set sizes. On the other hand, many-core CPUs can simultaneously process multiple queries providing high-throughput. RecPipe exploits these architectural differences to schedule each recommendation stage onto the underlying hardware. In fact, we find the optimal mapping of multi-stage recommendation varies across application-level targets (e.g., tail-latency, system load). Thus, RecPipe schedules multi-stage pipelines onto available hardware, based on algorithmic model parameters, architectural characteristics, and application-level requirements, to maximize quality and performance.

While achieving the maximal quality target and at iso-throughput, the scheduling optimizations reduce tail-latency by 4× on CPUs and a further 3× on heterogeneous hardware i.e., CPUs and GPUs (see Section 5 for details). However, despite these performance improvements there remains significant room for further optimization.

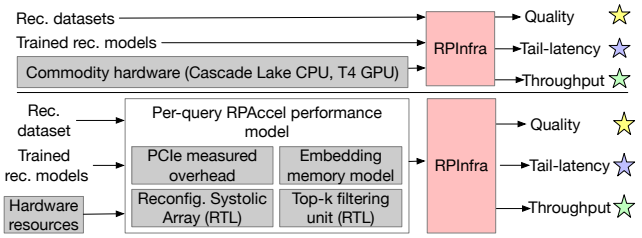

Figure 6: Evaluation methodology of RecPipe on commodity (top) and specialized (bottom) hardware.

In particular, the commodity CPU-GPU platforms suffer from two main drawbacks. First, GPUs exhibit low utilization when exploiting data-level parallelism in the frontend and model-level parallelism in the backend, primarily due to the high overhead of embedding lookups and memory transformation operations on GPUs [18]. Second, between stages, high PCIe communication overheads across the CPU and GPU limit achievable throughput. To address these limitations, and given the importance of data center-scale recommendation, RecPipe enables designing specialized hardware for multi-stage recommendation.

3.2 Custom hardware to accelerate multi-stage recommendation

Figure 5 illustrates the high-level architecture of the proposed recommendation accelerator, RPAccel. On the left, we start with a state-of-the-art accelerator baseline that minimizes inference latency for a single-stage recommendation model using a TPU-like monolithic systolic array and static cache for *hot-embeddings* [30]. The aforementioned software optimizations reduce workload complexity by decomposing the single-stage model into a multi-stage pipeline. Given the simplified workload, RPAccel is designed to concurrently process multiple models and queries, end-to-end. Figure 5(right) provides an ablation study for the proposed software and hardware optimizations, demonstrating significant latency and throughput improvement potential (i.e., O.1 to O.5).

By exploiting unique properties of multi-stage recommendation, RPAccel is designed to balance both inference latency and throughput based on application-level requirements.

- (O.1) RecPipe decomposes a single-stage model into multiple stages (2.5× latency reduction).
- (O.2) RPAccel comprises a top-k filtering unit to identify the k highest quality items based on predicted click-through-rate (CTR) to be ranked by subsequent stages; this eliminates host-accelerator communication between recommendation stages (1.5× latency reduction).
- (O.3) RPAccel implements a reconfigurable systolic array to concurrently process multiple stages and queries (2× hardware utilization and throughput). RecPipe's software scheduler (see above) splits the monolithic systolic array into multiple sub-arrays based on application-level targets (quality, latency, throughput) and multi-stage models.
- (O.4) RPAccel balances on-chip memory resources to statically cache *hot-embeddings* and dynamically prefetch embeddings for backend models (40% reduction in average memory access time). The static cache is provisioned for both frontend and backend stages; the dynamic cache prefetches embeddings for the backend as the frontend finishes sub-batches of the input query.
- (O.5) RPAccel breaks queries into sub-batches to pipeline – and thus – overlap computation from frontend and backend stages (1.3× latency reduction).

Machines	Cascade Lake CPU	NVIDIA T4 GPU
Frequency	2.8 GHz	585 MHz
Cores	64	2560
SIMD	AVX-512	FP32x64
Cache Sizes	1-16-22 MB	96-512 KB
DRAM Capacity	384 GB	15 GB
DDR Bandwidth	75 GB/s	300 GB/s
TDP	300 Watt	70 Watt

Table 2: Commodity hardware in experimental setup.

Parameter	RPAccel configuration
Frequency	250 MHz
Systolic Array SRAM capacity	8MB
Systolic Array MAC units	128×128 MACs
Embedding cache capacity	16MB
DRAM capacity	16 GB
DRAM bandwidth	64 GB/s
DRAM latency	100 cycles

Table 3: Fixed resources in RPAccel.

While achieving the highest quality target, compared to the baseline recommendation accelerator, RPAccel's optimizations collectively decrease tail-latency by up to 5× and increase throughput by up to 10× (see Section 6-7 for details).

4 EXPERIMENTAL METHODOLOGY

Figure 6 illustrates the evaluation methodology we use to study the system design implications of multi-stage recommendation. RecPipe encompasses a vast design space across multi-stage modeling parameters, hardware solutions, and application-level targets. To foster deeper understanding, we analyze cross-sections of the design space based on the application-level targets: iso-quality, iso-throughput, and iso-latency. This section details the methodology on both real, commodity hardware and simulated, specialized hardware.

Datasets and models. We evaluate RecPipe with three open-source datasets: Criteo Kaggle [36], MovieLens 1M [23], and MovieLens 20M [23]. We train neural matrix factorization models for both MovieLens datasets [25]. To provide intuition across the large design space studied in this work, we conduct a deep dive using Criteo and Facebook's DLRM [47]. On top of this deep dive, Section 8 summarizes results across all datasets. All models are implemented in PyTorch.

Application-level targets. This work optimizes recommendation based on three application-level targets:

- Quality: We use NDCG [5, 32] to quantify recommendation quality of the top sixty-four items served. For commensurate analysis, final results are presented based on the highest quality achieved for each model and dataset: NDCG of 92.25 for Criteo (see Section 2).

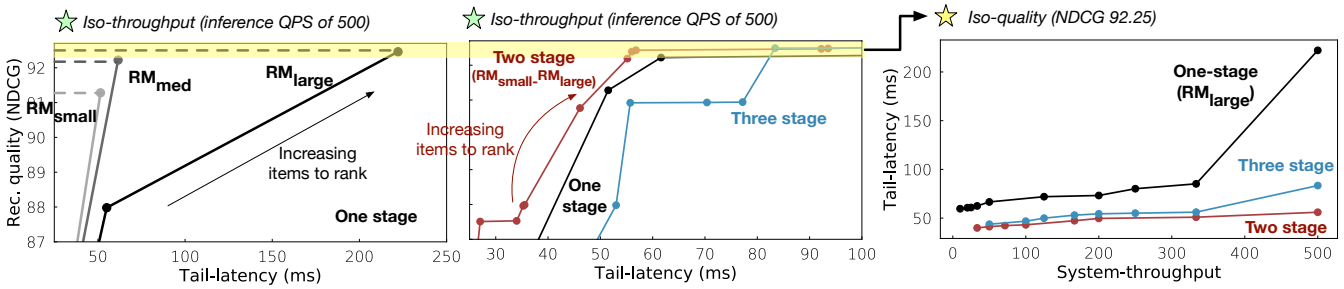

Figure 7: (Left) In single-stage recommendation, larger models achieve the higher quality at the expense of tail-latency. (Middle) Tuning multi-stage parameters improves quality under strict performance constraints. (Right) While achieving the highest-quality target, decomposing single-stage recommendation to multiple stages reduces tail-latency.

- Tail-latency: To maintain user-experience, recommendations must meet SLAs and be served under strict tail-latency targets [18], measured as 99^{th} percentile (*p99*).
- Throughput: Data-center recommendation systems must maximize throughput, measured as the queries processed per second (QPS). Queries follow a Poisson arrival rate.

Commodity hardware systems. To study the proposed designs in the context of data center scale recommendation, RecPipe runs datasets and models directly on real CPUs (server class Intel Cascade Lake) and GPUs (NVIDIA T4). Refer to Table 2 for detailed hardware specifications. Experiments on CPUs use multiple processes to exploit parallelism across cores—each core has a single PyTorch/MKL thread. GPUs use CUDA/cuDNN 10.1.

Accelerator modeling. RecPipe uses a two-step evaluation methodology to simulate specialized hardware.

First, we evaluate the latency of each query across each stage of the multi-stage pipeline. The latency per stage is computed as cumulative time from data transfers over PCIe, embedding lookups, MLP operations, and the top-*k* filtering units. Host-to-accelerator PCIe overheads are based on real measurements from the CPU-GPU system (see Table 2). For embedding lookups, we compute hit rates based on the cache locality of open-source datasets. Given the cache hit rates, we compute the memory latency of embedding operations using simple latency and bandwidth models for SRAM and DRAM. For MLP layers, We design and implement the systolic array and the top-*k* filtering unit in RTL to gather cycle-accurate performance measurements, including overheads from loading weights and activations from DRAM. Combining latency for all stages forms per-query performance model.

Second, the per-query latencies are fed into RecPipe which simulates the at-scale performance characteristics of RPAccel, measuring tail-latency, system-throughput, and quality, of processing tens of thousands of queries.

For area and power evaluations, we separately synthesize the reconfigurable systolic array, top-*k* filtering unit, and memories in a 12*nm* FinFET technology. As shown in Table 3, RPAccel implements comparable compute and memory resources to a data center TPU accelerator (40 Watt TDP) [35].

5 EVALUATION OF RECPIPE INFERENCE SCHEDULER ON COMMODITY HARDWARE

In this section we use RecPipe to efficiently schedule multi-stage recommendation onto heterogeneous hardware available in data centers. First, RecPipe balances the multi-stage modeling parameters—number of stages, models per stage, items to rank per stage—to co-optimize tail-latency, throughput, and quality. Next, RecPipe co-designs the multi-stage parameters for heterogeneous systems comprising CPUs and GPUs. We show the optimal configuration of multi-stage parameters depends on the underlying hardware. Furthermore, we show that while GPUs enable higher throughput and quality at low-latency targets, CPU-only execution achieves higher throughput under more relaxed latency targets.

5.1 Mapping multi-stage pipelines to CPUs

Figure 7(left) illustrates the tradeoff between tail-latency and quality for single-stage recommendation on CPUs. Following intuition, larger more complex models (e.g., RM_{large}) achieve higher quality at the expense of higher tail-latency.

Takeaway 1: *Carefully balancing multi-stage parameters unlocks higher recommendation quality and throughput at strict tail-latency targets.*

At a fixed system load (i.e., QPS of 500), Figure 7(center) shows tradeoff between tail-latency and quality for one-, two-, and three-stage designs. Exhaustively sweeping all possible combinations of models per stage and number of items to rank per stage, we show the Pareto-frontier results.

Compared to single-stage designs, Figure 7(center) shows multi-stage designs achieve higher quality under strict performance constraints. The single-stage design ranks all 4096 items with RM_{large}. The optimal two-stage design first processes 4096 items with RM_{small} followed by the top 256 items with RM_{large}, reducing tail-latency by 4× given the lower compute and memory demands.

The importance of optimizing for quality, not accuracy, can be seen by diving deeper into the two-stage design. To achieve high quality, the backend implements the most accurate network (i.e., RM_{large}); the frontend implements either RM_{med} or RM_{small}. While RM_{med} has higher accuracy, the benefits are overshadowed by the additional compute and memory requirements (see Table 1). In fact, with RM_{large} in the backend, both frontend options achieve

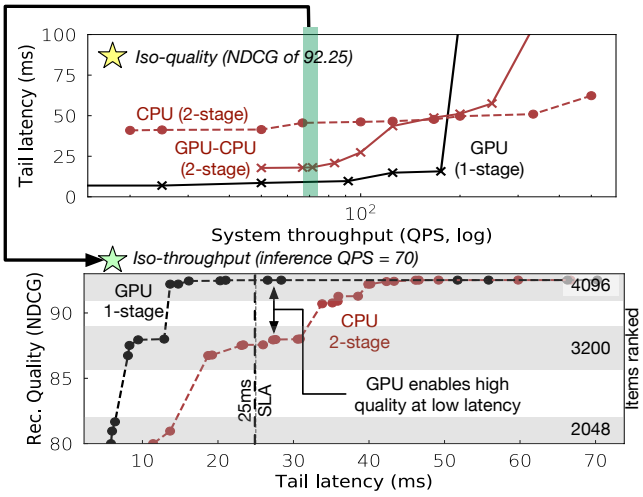

Figure 8: (Top) At iso-quality, mapping frontend (i.e., data-parallel) stages to GPUs reduces tail-latency by up to 3×; CPU-only execution achieves higher system throughput. (Bottom) At a lower system throughput (i.e., QPS of 70), the lower latency on GPUs can be traded off for higher quality compared to CPU-based execution.

the same quality (NDCG 92.25). But, the combination of RM_{med}-RM_{large} has a 1.6× longer tail-latency compared to RM_{small}-RM_{large}. Designers must jointly optimize for quality and performance.

In addition to quality, balancing multi-stage parameters improves throughput at strict tail-latency targets. Figure 7(right) shows the tradeoff between tail-latency and throughput, at the highest quality target (NDCG of 92.25). Compared with the one-stage system, the two-stage pipeline reduces tail-latency by 4.4× (QPS of 500). However, decomposing the pipeline into three stages decreases performance given additional queuing delays between stages, which overshadow the 30% reduction in compute between two- and three-stage designs. Note, the tradeoffs will vary across datasets—varying model complexities and items to rank per stage will impact the optimal configuration (see Section 8 for examples).

5.2 Mapping multi-stage pipelines to heterogeneous systems

Figure 8(top) illustrates the tradeoff between throughput and tail-latency while achieving the high quality target (NDCG of 92.25). Using RecPipe, we exhaustively evaluate all mappings between multi-stage recommendation and heterogeneous hardware and show the best configurations: one-stage GPU-only, two-stage GPU-CPU, and the two-stage CPU-only configurations in Figure 8(top). For the two-stage GPU-CPU design, RecPipe maps either the frontend or the backend to the GPU, running the other on the CPU. In particular, we show results for frontend running on the GPU and backend on the CPU as our empirical evaluations show it provides higher performance. We also evaluate mapping two stages to the GPU with multi-tenant execution. Our evaluations show this configuration is unable to extract the fine-grain parallelism from multi-stage's data dependency, incur longer latency than the one-stage GPU-only configuration.

Takeaway 2: *Given architectural differences, the optimal multi-stage parameters vary on CPUs versus GPUs.*

Recall from our previous analysis, for CPU-only execution the two-stage design achieves the highest performance; on the heterogeneous system, however, the single-stage GPU-only configuration (solid black) achieves higher performance than multi-stage using both CPU and GPU (solid red). The reason is twofold. First, we observe comparable latency for RM_{small} versus RM_{large} on the GPU, overshadowing the benefits of decomposing models into finer-grained pipelines. Second, the multi-stage GPU-CPU design requires transferring more intermediate results across PCIe, incurring heavy queuing delays and limiting system performance.

Nonetheless, the multi-stage GPU-CPU design plays an important role. Recent work shows production-scale recommendation model sizes are growing rapidly—by an order of magnitude in just three years [45]. For production-scale models that are larger than the DRAM capacity available on GPUs (e.g., ~ 15GB on NVIDIA T4), designers will need to decompose models into multiple stages. Here, frontend stages run on the GPU in order to circumvent storage capacity limits and exploit data-parallelism with the larger input working set size; the backend models run on the CPU. Figure 8(top) shows that this multi-stage GPU-CPU design achieves up to 3× lower latency than the multi-stage CPU-only configuration.

Takeaway 3: *By maximizing throughput at low latency, GPUs unlock higher recommendation quality.*

Despite the GPUs achieving 3× lower latency than the CPU-only designs (see Figure 8(top)), the GPUs remain underutilized with an occupancy of 25%, and memory and power utilization of 10% and 45%, respectively. Improving utilization requires higher batching. Unfortunately, as we increase batching and system throughput (x-axis), the GPU-enabled designs suffer from a sudden degradation in tail-latency due to high queuing delays; in comparison, the CPUs sustain higher throughput by concurrently processing queries across cores (e.g., task-parallelism).

While the latency reduction from GPU's does not translate to higher throughput, it can enable higher quality. Figure 8(bottom) illustrates the tradeoff between tail-latency and quality for CPU- and GPU-based recommendation at iso-throughput. Following our previous results, we show the optimal configurations: single-stage GPU-only and two-stage CPU-only designs. Given the fixed models, RecPipe tradeoffs off latency for quality by increasing the number of items ranked per query. At a strict SLA target of 25*ms*, the CPU achieves an NDCG of 87, while the GPU achieves an NDCG of 92.25. The increase in quality is a direct result of GPU's data-parallel architecture allowing it to rank 4096 items compared to the CPU ranking only 3200 items at the 25 *ms* SLA. Thus, AI accelerators for recommendation must be evaluated not only for performance benefits but also on quality achieved under strict performance and resource constraints.

Limitations of commodity hardware. Based on the performance analysis above, we identify multiple limitations of commodity platforms running multi-stage recommendation. In particular, GPUs do not directly benefit from decomposing models into multiple stages. This is due to the limits of multi-tenant execution, under utilized hardware when separately exploiting data- and model-level parallelism across stages, and high PCIe data communication between stages. Given these limitations and the growing scale of

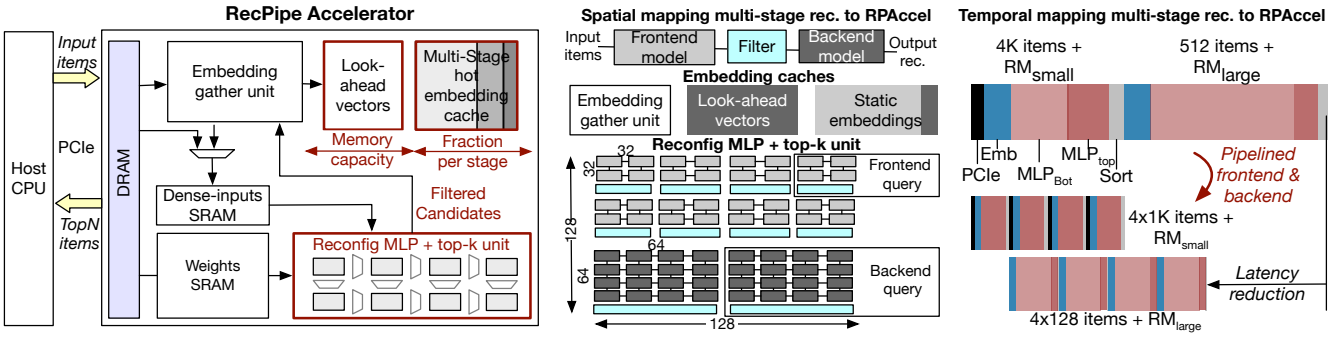

Figure 9: (Left) Overall design of the RecPipe accelerator (RPAccel) comprising an embedding gather unit with two on-chip caches for static and dynamic vectors, and a reconfigurable MLP and top-k filtering unit. (Middle) Static mapping of multi-stage recommendation onto RPAccel. Frontend and backend share both memory and compute resources. (Right) Temporal mapping of multi-stage recommendation onto RPAccel with pipelined frontend and backend models.

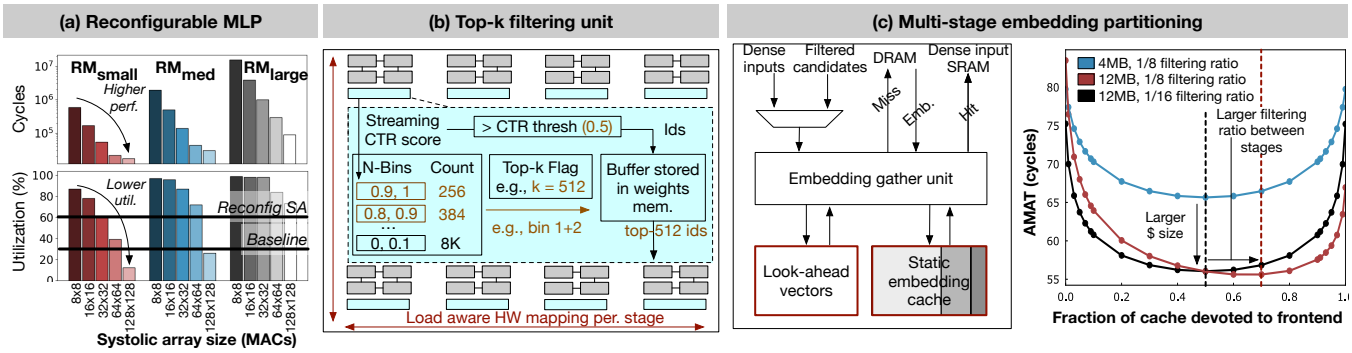

Figure 10: Design space exploration of RPAccel. (a) Larger systolic arrays suffer from low utilization on smaller models, motivating provisioning resources into sub-arrays for concurrent query processing. Compared to a monolithic array with 30% utilization, the reconfigurable array has a 60% utilization. (b) Top-k filtering unit designed to minimize area and power while eliminating host-accelerator PCIe communication overheads. (c) On-chip embedding cache resources must be asymmetrically provisioned across frontend and backend to minimize average memory access time.

personalized recommendation across Internet services [45, 66, 67], we use RecPipe to unlock the opportunities from multi-stage ranking by designing specialized hardware to provide high quality and infrastructure efficiency, in the following section.

6 ANALYSIS OF RECPIPEACCEL'S DESIGN SPACE

This section proposes RPAccel, a specialized accelerator tailored to multi-stage recommendation models. We start with a baseline TPU-like recommendation accelerator [30]. The baseline optimizes for low-latency single-stage inference, but suffers from low utilization and system throughput on multi-stage pipelines. To accelerate multi-stage recommendation, as summarized in Section 3.2, RPAccel comprises four main features that exploit distinct opportunities enabled by RecPipe: the pipeline execution, a reconfigurable MLP unit, a top-k filtering unit, and the partitioned embedding cache for hot-vectors across models and prefetched backend vectors.

6.1 Mapping multi-stage pipelines to RPAccel

Figure 9(left) illustrates the high-level architecture of RPAccel. Unlike prior art which accelerates *single-stage* model inferences alone, RPAccel is designed to process queries end-to-end: model inferences for *multiple stages* and *filtering* top-k user-item interactions between stages. Figure 9(center) shows how multi-stage recommendation is mapped onto RPAccel. Networks across the stages share accelerator memory and compute resources. For each stage, to produce predicted CTR scores for each user-item pair, RPAccel comprises an MLP and embedding gather unit. RPAccel implements a set of top-k filtering units to identify high-quality user-item pairs.

Takeaway 4: *Breaking queries into multiple sub-batches enables pipelined execution of frontend and backend stages.*

Figure 9(right) shows the temporal mapping of multi-stage recommendation onto RPAccel. To reduce latency, RPAccel pipelines frontend and backend stages by breaking queries into smaller sub-batches. As an example, Figure 9(right) shows RPAccel splitting a single query of 4K items into four smaller batches of 1K each, overlapping frontend and backend stages. The degree of sub-batching must be carefully balanced in order to maintain high utilization

and quality. Smaller batch-sizes incur higher inference overheads (e.g., weight loading) but can better overlap frontend and backend stages. Furthermore, splitting queries into n smaller batches can degrade quality as the top-k items in each stage are set by stitching the top-$\frac{k}{n}$ items in each batch. Using RecPipe, we ensure the system maintains high-quality and splits queries into four sub-batches for workloads studied in this paper.

6.2 Customization of RPAccel micro-architecture

Below we detail RPAccel's micro-architectural design space.

Takeaway 5: *Splitting monolithic systolic arrays into sub-arrays improves recommendation inference throughput by concurrently processing multiple models and queries.*

As recommendation comprises large input working set sizes, RPAccel implements a weight stationary, systolic array-based MLP engine [7, 35, 50]. To concurrently process multiple stages and queries, RPAccel dynamically splits a monolithic array into independent sub-arrays [15]. Figure 10(a) illustrates the benefit of a reconfigurable systolic array for multi-stage recommendation. We show the MAC utilization for various array sizes and models. Larger arrays achieve lower latency but suffer from lower utilization when processing small models (i.e., RM_{small}). In fact, when processing a two-stage pipeline, the fixed, monolithic array has an average utilization of only 30%, as it is overprovisioned for the frontend (i.e., RM_{small} ranking 4K items). Splitting the monolithic array into smaller units improves utilization to 60%, doubling throughput at comparable latency.

Note, RPAccel's reconfigurable systolic array is inspired by prior work which proposes a fission architecture to split monolithic arrays into sub-arrays for multi-tenancy [15]. Customized for multi-stage recommendation, RPAccel eliminates complex, omni-directional interconnects, incurring a lower area and power penalty (i.e., 13% area and 21% power in [15] versus 6% and 11% in RPAccel)[1], and extends the reconfigurability in response to application QPS and SLA targets.

Takeaway 6: *Implementing top-k user-item filtering units in specialized hardware eliminates PCIe overheads.*

Based on the predicted CTR, top scoring user-item interactions must be filtered and forwarded to subsequent recommendation stages. Prior recommendation accelerators only process MLP inference [30, 33]. Thus, the filtering step is offloaded to host-processors incurring high PCIe overheads [30]. To eliminate communication overheads, RPAccel implements a set of on-chip top-k filtering units (see Figure 9 middle, blue). One approach to identify the top-k user-item pairs is to sort all CTR scores. Unfortunately, sorting latency scales with the number of items to rank, potentially consuming tens-thousands of cycles for recommendation given large input sizes. Furthermore, existing hardware sorting units consume significant area and power [44].

Instead, RPAccel exploits two unique properties of recommendation inference to simplify the filtering unit. First, between stages, the final top-k user-item pairs need not be ordered—RPAccel implements an approximate, bucketing design. Second, the final MLP

[1]Following the baseline [15], we exclude on-chip SRAM when comparing area and power. Figure 11 includes SRAM overheads.

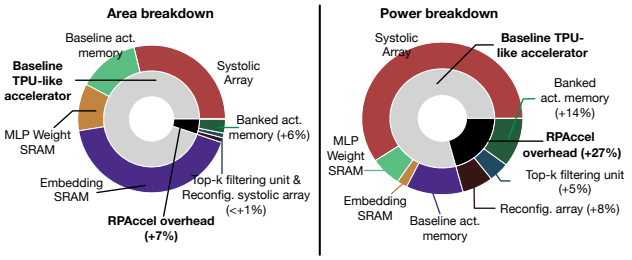

Figure 11: Compared to the baseline, RPAccel incurs 7% and 27% area (left) and power (right) overheads.

layer produces one CTR score per cycle leading to a streaming filtering unit design.

Figure 10(b) shows the resulting top-k filtering unit. The filtering unit maintains N bins (e.g., N=16). Each bin represents user-item pairs of a specific CTR score range between 0 and 1. As a new CTR score arrives every cycle, the filtering unit adds the user-item *id* to the corresponding bin and increments its counter. For example, Figure 10(b) shows the top bin counts user-item pairs with CTR scores between 0.9 and 1 (high quality). Based on the CTR score, the user-item pair *id* is stored in a dedicated portion of the systolic array banked weight SRAM. Storing all (4K) user-item *id* pairs consumes 12% of the weight SRAM. To reduce this overhead, RPAccel skips user-item pairs with low CTRs. Using RecPipe, we set a minimum CTR threshold of 0.5, reducing the overhead on weight memory to 3%. Once all user-item CTRs are categorized, the filtering unit identifies and copies at least top-k user-item pairs indicated from the highest n bins to DRAM. These *id*s uniquely reference continuous and categorical inputs for subsequent stages.

Given the streaming design, the performance overhead of the filtering step is set by the latency it takes to identify and send user-item *id*s from the highest bins to main memory. We find this takes a couple hundred accelerator cycles, negligible compared to model inference. Although each sub-array in RPAccel's reconfigurable systolic array requires a separate top-k filtering unit, the area and power overheads are small (see Figure 11) and there is no degradation in quality.

Takeaway 7: *Asymmetrically-provisioned embedding caches tailored for each of the multi-stage models minimizes memory access latency.*

Recent work shows embedding table operations suffer from irregular memory access patterns, low compute intensity, and high storage capacities [20]. Consequently, the performance of embedding table operations is bounded by embedding vector fetch latency. Prior work exploits the power-law distribution of embedding lookups to cache frequently accessed vectors on-chip [2, 30, 38]. The embedding caches in prior work however assume a single stage recommendation model.

Instead, RPAccel implements an embedding cache customized for multi-stage recommendation by comprising (1) a *static embedding cache* that is provisioned statically for hot embedding vectors from both frontend and backend stages, (2) a *look-ahead embedding cache* that stores embedding vectors for in-flight queries. It also prefetches lookups for later stages in RPAccel's pipeline optimization (Figure 9(right)). As shown in Figure 9(left), input embedding IDs arrive either from the host processor for frontend models or the output

of top-k filtering units for backend models. Based on the IDs, the embedding gather unit first checks if the corresponding vectors are in the caches. If yes, the embedding vectors are returned to the "Dense-input SRAM" to be processed by the MLP-top layers. If not, the embedding gather unit retrieves the vectors from DRAM to the look-ahead embedding cache.

Embedding cache provisioning. Following data center AI accelerators with 24MB capacity [35], we start with 16MB for embedding caches (8MB in MLP weights/activations). The size of the *look-ahead* cache is bounded by the number of items ranked in backend stages, size of embedding vectors, and maximum number of queries in flight. For the worst case we conservatively provision 4MB for the *look-ahead* cache. This leaves 12MB for the *static embedding* cache. Figure 10(c) shows the impact of asymmetrically provisioning memory for frontend and backend models on the average memory access time (AMAT) for embeddings. With a 128 byte cache line size, the embedding vector size of RM_{large}, we find the fraction of storage devoted to the frontend versus backend depends on the item filtering ratio between stages. Given a filtering ratio of one-eighth for Criteo, we provision equal memory capacity for the frontend and backend.

Area and power breakdown. Figure 11 illustrates the area and power overheads of the proposed optimizations compared to the baseline, TPU-like recommendation accelerator [30]. The combination of the reconfigurable MLP unit, top-k filtering unit, and multi-stage aware embedding cache incurs a total of 7% area and 27% power overhead, moderate compared to RPAccel's performance benefits (see Section 7).

7 EVALUATION OF RPACCEL AT-SCALE

In this section we evaluate the performance of RPAccel at-scale. Instrumenting RecPipe with the simulated RPAccel we study the proposed hardware solutions in terms of quality, tail-latency, and system-throughput. We study RPAccel using publicly available models and datasets; and also project the quality and performance trends for future recommendations.

7.1 RPAccel evaluation on open-source use cases

Takeaway 8: *By accelerating multi-stage recommendation, RPAccel achieves 3× lower latency and 6× higher throughput compared to baseline, single-stage designs.*

Given fixed hardware resources, Figure 12(top) illustrates the tradeoff between throughput and latency as we vary the RPAccel-provisioning decisions for all stages. The baseline follows Centaur [30]—a single-stage recommendation accelerator which implements a TPU-like systolic array [35] and uses the host-processor to filter top-k interactions. The baseline achieves a 6ms and 21ms tail-latency at the inference throughput of 200 and 400 QPS, respectively. While achieving the same quality, the single-stage RPAccel design achieves a 4.5ms and 9ms tail-latency at 200 and 400 QPS, respectively. Furthermore, decomposing recommendation into finer-grained pipeline enables a minimum latency of 2.1ms at 200 QPS or, at 6ms a throughput of 1300 QPS— 3× and 6× improvement over the single-stage baseline, respectively. The latency reduction

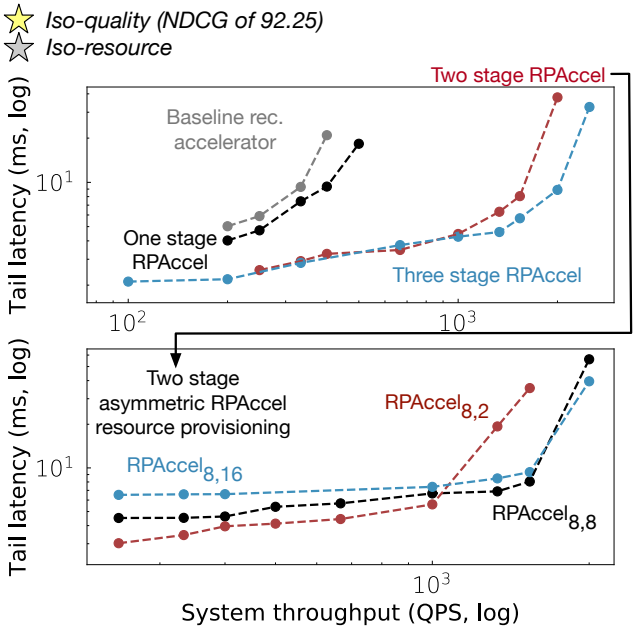

Figure 12: (Top) At iso-quality and hardware resources, co-designing multi-stage models with hardware enables lower tail-latency and higher system throughput. (Bottom) Asymmetrically provisioning RPAccel resources across stages further improves performance.

and throughput increase owe RPAccel's software and hardware optimizations.

Takeaway 9: *Asymmetrically provisioning accelerator based on multi-stage recommendation models resources unlocks lower tail-latency and higher system-throughput.*

Figure 12 (bottom) illustrates the benefit of asymmetrically provisioning RPAccel resources across stages. For a two-stage recommendation pipeline, the frontend is fixed with sub-arrays while the backend implements two (i.e., $RPAccel_{8,2}$), eight (i.e., $RPAccel_{8,8}$), and sixteen sub-arrays (i.e., $RPAccel_{8,16}$). All experiments assume iso-hardware resources while achieving the maximum quality target (i.e., NDCG of 92.25).

Compared with the homogeneous accelerator (i.e., $RPAccel_{8,8}$), aggregating the backend into fewer, larger arrays $RPAccel_{8,2}$ reduces the latency at low throughput by 1.5×. Similarly, at high system load, splitting the backend into multiple, smaller units $RPAccel_{8,16}$ reduces the latency by 1.4×. Given application-level latency and system targets, asymmetrically provisioning RPAccel resources across stages widens the design space of recommendation services. Building on prior art, RPAccel resources are dynamically reconfigured to meet varying targets, given workload demands [15, 41, 54].

7.2 RPAccel evaluation on future models

So far we have analyzed the performance of RPAccel on open-source use-cases. However, recent literature shows production-scale recommendation models are rapidly growing in size, outpacing DRAM capacity and even reaching TBs in size [45]. One promising

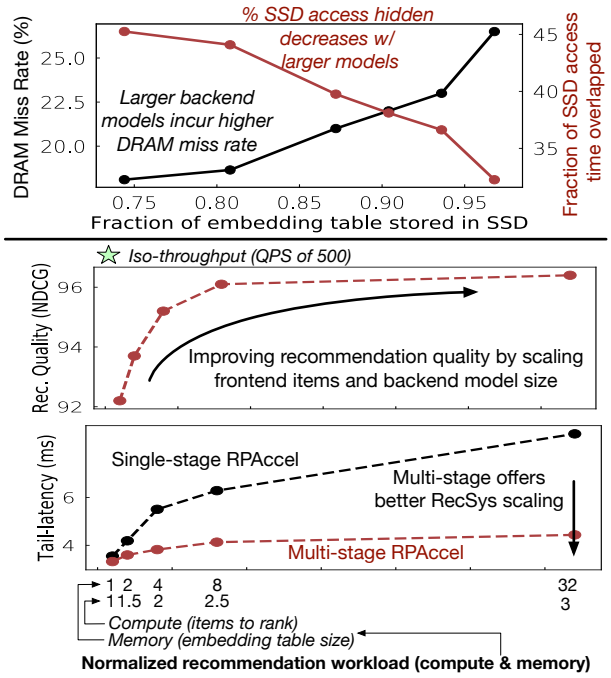

Figure 13: (Top) Projecting the performance impact of scaling recommendation models to higher capacities requiring SSD storage. (Bottom) Compared to the single-stage accelerator baseline, RPAccel provides graceful performance trends with future model sizes by also scaling items to rank to overlap frontend and backend stages.

path to enabling future, production-scale models is to use higher capacity memories such as SSDs [13, 56]. Here we consider the performance implications of SSDs on RPAccel.

Storing larger embedding tables in SSD lowers embedding locality and degrades performance. Figure 13(top) shows the impact of larger embedding tables on embedding locality. While frequently accessed embedding vectors are stored DRAM, a larger portion of these tables are stored in the SSD (x-axis). For example, increasing the size of RM_{large} by 32× requires storing 97% of the embedding tables in SSD. This also causes increases DRAM miss rates from 17% to 28%. Recall, RPAccel pipelines frontend and backend stages—allowing the accelerator to overlap long latency SSD accesses in the backend. However, Figure 13(top) shows with growing embedding table sizes, a smaller fraction of the accesses can be overlapped causing an increase in latency.

Takeaway 10: *Compared to baseline single-stage accelerators, RPAccel achieves higher quality and performance when scaling both frontend and backend stages towards future recommendation engines.*

In addition to scaling embedding tables in backend models (e.g., model size), one can also increase the number of items to rank in the frontend (e.g., compute demand). Figure 13(bottom) shows the impact of scaling both frontend and backend stages (x-axis) on quality. Starting from the baseline configuration, we project increasing model size by 32× and compute complexity from ranking 4K items to 12K items improves quality from an NDCG of 92.25 to 96.

Increasing the items to rank allows RPAccel to more effectively overlap the frontend and backend stages. Figure 13(bottom) shows the corresponding tail-latency impact on scaling compute and memory complexity assuming iso-throughput (QPS of 500). We show two configurations: single-stage (black) and multi-stage (red) RPAccel. By overlapping frontend and backend stages, the multi-stage design achieves higher performance for larger recommendation engines compared to the single-stage design. More generally, we show the importance of tightly-coupling algorithm and hardware scaling for future recommendation engines; RecPipe and RPAccel open such new co-design opportunities.

8 SUMMARY OF RECPIPE RESULTS

Figure 14 summarizes the performance benefits of the proposed solutions, co-designing models and hardware for multi-stage recommendation. The results show the tail-latency across three datasets (i.e., Criteo Kaggle, MovieLens 1M and 20M [23, 36]), system loads (i.e., QPS of 100, 500, 2000), and hardware platforms (i.e., CPU, GPU, Accel). The colored bars distinguish between one- (black), two- (red), and three- (blue) stage recommendation pipelines. Following our previous analysis, CPU designs assume CPU-only execution (Section 5.1). For GPU-based configurations, 1-stage designs represent GPU-only execution; 2-stage and 3-stage designs represent heterogeneous GPU-CPU execution (Section 5.2). *Accel* configurations assumes RPAccel-only execution (Section 6-7). Across the system loads and datasets, RecPipe reduces tail-latency by an average of 3.2× on commodity hardware; compared to prior recommendation accelerators, RPAccel reduces tail-latency by 4.3× on average.

Differences across system loads. Across different system loads, the optimal multi-stage configuration and hardware platform varies. For instance, with the Criteo dataset on GPU-enabled hardware, between low (QPS of 100) and medium (QPS of 500) loads, the optimal number of stages varies from one to two. Similarly, for Criteo, the optimal hardware backend between low and medium loads changes from GPUs to CPUs, respectively. Differences across system loads owe to varying system optimization strategies for maximizing throughput versus minimizing latency; for example, throughput is maximized by processing multiple queries concurrently while latency is minimized by accelerating individual queries.

Differences across datasets. In addition to varying system loads, the optimal multi-stage configuration and hardware platform varies across datasets. For example with commodity hardware, on the Criteo dataset, CPUs achieve lower tail-latency than GPUs for system loads above 100 QPS; on the other hand, GPU-based designs outperform CPU-only execution for both MovieLens datasets. With RPAccel, tail-latency is optimized with the deeper three-stage pipeline for MovieLens-20M at 500 and 2000 QPS and all loads for Criteo; on MovieLens-1M however two-stage is optimal. Differences across datasets owe to the Criteo implementing DLRM [47] with higher embedding capacities while MovieLens implementing neural matrix factorization models dominated by MLP layers; furthermore, across stages the number of items to rank reduces by roughly 5×, 2.5×, and 4×, on Criteo, MovieLens 1M, and MovieLens 20M, respectively. These differences highlight the need to co-design multi-stage recommendation parameters with the underlying hardware early in the design process using frameworks like RecPipe.

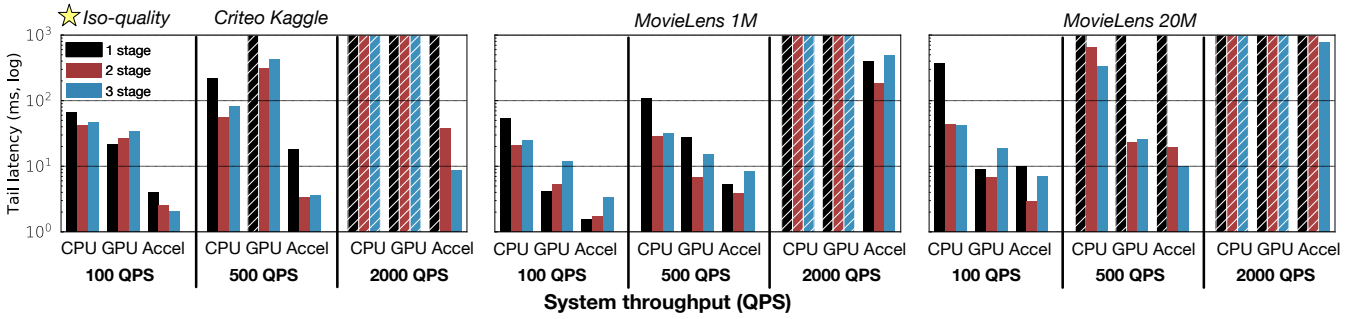

Figure 14: Summary of RecPipe results at iso-quality for the Criteo, and MovieLens 1M and 20M datasets. For each dataset, we show the tail-latency (log scale) for three system loads and hardware platforms. Configurations are greyed out when system load is not met. The optimal multi-stage design varies across loads, hardware platforms, and datasets.

Benefits of RPAccel. Compared with CPUs and GPUs, RPAccel significantly reduces tail-latency of multi-stage recommendation across different datasets and system loads. In fact, in many cases (e.g., Criteo and MovieLens20M datasets) RPAccel is optimized with deeper pipelines compared to commodity GPUs; This is a direct result of extracting data-level and model-level parallelism opportunities across multi-stage recommendation and eliminating high-overhead host-accelerator communications that RPAccel enables.

9 RELATED WORK

While systems and computer architecture researchers have proposed various solutions to optimize cloud-scale personalized recommendation models, relatively little work explores co-design opportunities between models and hardware to jointly optimize quality and performance, as well as the unique characteristics of multi-stage recommendation.

DNN-based recommendation models. To improve content personalization, recommendation models are growing rapidly in size and complexity [45, 65–67]. Tackling the growing model sizes, researchers have proposed techniques to compress embedding tables while preserving accuracy [14, 16, 52, 59]. Alternatively, one can decompose large monolithic models into multi-stage pipelines. Industry publications show multi-stage designs are used for serving content on Youtube [65] and Instagram [17, 31]. To balance recommendation quality and model complexity, machine learning researchers have explored a variety of modeling techniques to train each stage of the multi-stage pipeline [37]. However, in prior work, the multi-stage recommendation systems are designed to maximize quality, independent on the underlying hardware. RecPipe extends prior art by co-designing the multi-stage models and underlying hardware—commodity and specialized—in order to tightly co-optimize quality, tail-latency, and throughput for data center scale deployment.

Specialized recommendation hardware. Lots of research effort has been devoted to design specialized hardware for deep learning—especially MLPs, CNNs, and RNNs [4, 6–8, 19, 22, 35, 48–50, 53, 61, 62]. However, recommendation systems pose distinct challenges owing to their network architectures and use cases [20, 28, 47]. Given its importance, hardware proposals for accelerating recommendation models have begun to emerge [1, 3, 9–12, 18, 21, 29, 30, 33, 34, 38–40, 42, 43, 46, 55–57]. While prior work focuses on improving hardware efficiency given fixed workloads, RecPipe brings quality into the mix. Accounting for both quality and performance, this work co-designs multi-stage models and hardware. In addition to RecPipe's post-training inference scheduler on commodity hardware, we compare the proposed RPAccel to a state-of-the-art TPU-like baseline recommendation accelerator, Centaur [30]; compared to the baseline, we demonstrate that by co-designing models and hardware, RPAccel jointly improves recommendation quality, tail-latency, and throughput.

10 CONCLUSION

Given the growing prevalence of personalized recommendation, architects have invested significant resources improving recommendation inference efficiency. While proposed solutions tackle different compute and memory bottlenecks, they do not directly co-optimize quality and performance. In this work we propose RecPipe, a system for co-designing models and hardware to jointly optimize quality, tail-latency, and throughput. First, RecPipe splits monolithic models into multi-stage pipelines exposing unique system optimization opportunities. Next, we design an inference scheduler that maps multi-stage recommendation across CPUs and GPUs. Finally, we deign a novel hardware accelerator for multi-stage recommendation which achieves high-quality while improving latency and throughput by up to 3× and 6×, respectively, over a baseline TPU-like recommendation accelerator.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and artifact evaluation committee in providing valuable feedback in improving this work and corresponding artifact. The academic authors, at Harvard University, of this work were also supported in part by the NSF Graduate Research Fellowship, SRC JUMP Applications Driven Architecture (ADA) center, and Intel Corporation.

A ARTIFACT APPENDIX

A.1 Abstract

This section summarizes the artifact evaluation for this work. First, we provide the check-list for this artifact. Next, we describe the

directory structure for the code. Finally, the installation, experiment workflow, and evaluation illustrate how to use the artifact to reproduce results and extend the implementation.

A.2 Artifact check-list (meta-information)

- **Program:** PyTorch and Verilog RTL code.
- **Model:** Deep Learning Recommendation Model (DLRM) provided in artifact.
- **Datasets:** Criteo Kaggle, MovieLens 1 million, and MovieLens 20 million dataset publicly available.
- **OS environment:** Ubuntu 16.04 with CUDA 8.
- **Hardware setup:** Server class Intel CPU and Inference GPU (preferably Intel Cascade Lake CPU with NVIDIA T4 inference GPU)
- **Run-time requirements:** Require isolated server as experiments sensitive to resource contention, adding performance variability to tail-latency.
- **Execution:** Sole user.
- **Metrics:** Normalized discounted cumulative gain (NDCG), tail-latency, and throughput.
- **Output format:** Experiments produce console and text files. Expected output either in paper or code repository.
- **Expected disk space required (approximately)?:** 32 GB
- **Expected time to prepare workflow (approximately)?:** 32 GB
- **Expected time to complete experiments (approximately)?:** 24 hours
- **Publicly available?:** Yes
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.5146295

A.3 Description

A.3.1 How to access. Code for this paper can be accessed by cloning the public GitHub repository: https://github.com/harvard-acc/RecPipe

We have also published the initial artifact on Zenodo. The artifact DOI is: 10.5281/zenodo.5146295 . The Zenodo DOI URL is: https://doi.org/10.5281/zenodo.5146295 .

A.3.2 Hardware dependencies. To reproduce results found in this paper we suggest using commensurate hardware. This includes an Intel Cascade Lake CPU server class CPU (32 cores) with an NVIDIA T4 inference GPU. In the absence of this hardware, we suggest using similar hardware in terms of a many core CPU server with NVIDIA GPU.

A.3.3 Software dependencies. To define machine learning models the code uses PyTorch. To run on the NVIDIA GPU we use CUDA version 8 and cudnn version 6. A docker image is provided in the code to facilitate managing software dependencies.

We also provide the Verilog RTL for each component of RPAccel. The RTLs are synthesized in Cadence Genus with a commercial 12nm technology node, and the memory macros are generated and characterized with ARM SRAM compiler. Unfortunately due to the NDAs, we are not able to grant access to the original logic synthesis environment. As a solution, we provide the original logic synthesis reports (power and area) with timestamps for the evaluation. Alternatively, the provided RTLs can be synthesized with other synthesis tools (e.g., Synopsys DC) and open-source PDKs (e.g., freePDKs) at the evaluators' choice, but the exact numbers might change.

A.3.4 Datasets. The experiments use Criteo Kaggle, MovieLens 1 million, and MovieLens 20 million datasets. While the data sets are publicly available, we cannot provide full copies of the datasets in the final archive.

A.3.5 Models. Example pre-trained models are provided with the artifact for evaluation.

A.4 Installation

To install the necessary packages we provide a Docker image (in `docker/`) and a requirements file to be used with `pip`.

A.5 Experiment workflow

For the ease of artifact evaluation, we provide sample bash scripts that automate launching and analyzing experiments. Key steps include:

- Download code: Clone the GitHub repository: https://github.com/harvard-acc/RecPipe
- Build docker image by running `cd docker/; ./build-docker.sh`
- Launch docker image in interactive mode
- Download the pre-trained models from the artifact; these models are trained on the publicly available Criteo Kaggle and provided by academic authors on this work. Download the Criteo Kaggle dataset which is publicly available (see https://www.kaggle.com/c/criteo-display-ad-challenge for reference).
- Update DATA-DIR environment variable in bash scripts in `scripts/` directory
- Update model locations in json configurations files (`dim_4_0.json`, dim_16.json, dim_32_0.json) in `configs/model_configs` directory.
- From the repository's root directory, run test script, `./scripts/recpipe_kaggle.sh`
- From the repository's root directory, run Figure 3 experiment: `./scripts/artifact_eval/figure3.sh`
- From the repository's root directory, run Figure 7 and 8 experiment: `./scripts/artifact_eval/figure7_8.sh`
- From the repository's root directory, run Figure 10 experiments: `./scripts/artifact_eval/figure10a.sh` `./scripts/artifact_eval/figure10c.sh`
- From the repository's root directory, run Figure 12 experiment: `./scripts/artifact_eval/figure12.sh`
- From the repository's root directory, run Figure 13 experiment: `./scripts/artifact_eval/figure13.sh`
- From the repository's root directory, after having run the experiments for Figure 7, 8 and 12, you can generate results for Figure 14 by: `python plotting/figure14/parse_kaggle.sh`

A.6 Evaluation and expected results

Each of the artifact evaluation scripts describe in the "Experiment Workflow" produce results for a specific figure in this paper. The bash scripts include Python scripts to format the experiment outputs for specific bars and lines in each Figure. Please run these scripts from the repository's root directory (e.g., `python plotting/figureX/*.py`)

A.7 Experiment customization

The codebase is organized to facilitate customization and extension. The configuration files in `configs/` allow for easily specifying experiment configurations including number of queries in at-scale simulations, model configurations, number of CPU cores, use of GPU, batch-configurations, and arrival rates.

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

REFERENCES

[1] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. 2020. Understanding Training Efficiency of Deep Learning Recommendation Models at Scale. *arXiv preprint arXiv:2011.05497* (2020).

[2] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J Nair. 2021. High-Performance Training by Exploiting Hot-Embeddings in Recommendation Systems. *arXiv preprint arXiv:2103.00686* (2021).

[3] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung-Kyu Lim, and Hyesoon Kim. 2021. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In *2021 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[4] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. 2016. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[5] Wei Chen, Tie-yan Liu, Yanyan Lan, Zhi-ming Ma, and Hang Li. 2009. Ranking Measures and Loss Functions in Learning to Rank. In *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (Eds.), Vol. 22. Curran Associates, Inc., 315–323. https://proceedings.neurips.cc/paper/2009/file/2f55707d4193dc27118a0f19a1985716-Paper.pdf

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Teman. 2014. DaDianNao: A Machine-Learning Supercomputer. In *MICRO*.

[7] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

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

[9] Marshall Choy. 2020. Accelerating the Modern Machine Learning Workhorse: Recommendation Inference. https://sambanova.ai/blog/accelerating-the-modern-ml-workhorse-recommendation-inference

[10] NEUCHIPS Corp. 2020. NEUCHIPS Recommendation Accelerator RecAccel. https://2ca8d951-4386-4e41-9cab-50c86da5f5a8.filesusr.com/ugd/d79931_9382d53600f54d21a6eabe46d1f0ffa2.pdf

[11] Zhaoxia Summer Deng, Jongsoo Park, Ping Tak Peter Tang, Haixin Liu, Jie Yang, Hector Yuen, Jianyu Huang, Daya S Khudia, Xiaohan Wei, Ellie Wen, Dhruv Choudhary, Raghuraman Krishnamoorthi, Carole-Jean Wu, Nadathur Satish, Changkyu Kim, Maxim Naumov, Sam Naghshineh, and Misha Smelyanskiy. 2021. Low-Precision Hardware Architectures Meet Recommendation Model Inference at Scale. *IEEE Micro* (2021), 1–1. https://doi.org/10.1109/MM.2021.3081981

[12] Xiaocong Du, Bhargav Bhushanam, Jiecao Yu, Dhruv Choudhary, Tianxiang Gao, Sherman Wong, Louis Feng, Jongsoo Park, Yu Cao, and Arun Kejariwal. 2021. Alternate Model Growth and Pruning for Efficient Training of Recommendation Systems. arXiv:cs.IR/2105.01064

[13] 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:cs.LG/1811.05922

[14] Benjamin Ghaemmaghami, Zihao Deng, Benjamin Cho, Leo Orshansky, Ashish Kumar Singh, Mattan Erez, and Michael Orshansky. 2020. Training with Multi-Layer Embeddings for Model Reduction. arXiv:cs.LG/2006.05623

[15] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. 2020. Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 681–697. https://doi.org/10.1109/MICRO50266.2020.00062

[16] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed dimension embeddings with application to memory-efficient recommendation systems. *arXiv preprint arXiv:1909.11810* (2019).

[17] Shuhei Goda, Naomichi Agata, and Yuya Matsumura. 2020. A Stacking Ensemble Model for Prediction of Multi-Type Tweet Engagements. In *Proceedings of the Recommender Systems Challenge 2020 (RecSysChallenge '20)*. Association for Computing Machinery, New York, NY, USA, 6–10. https://doi.org/10.1145/3415959.3415994

[18] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandom 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

[19] Udit Gupta, Brandon Reagen, Lillian Pentecost, Marco Donato, Thierry Tambe, Alexander M Rush, Gu-Yeon Wei, and David Brooks. 2019. Masr: A modular accelerator for sparse rnns. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 1–14.

[20] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of facebook's DNN-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 488–501.

[21] Vipul Gupta, Dhruv Choudhary, Peter Tang, Xiaohan Wei, Xing Wang, Yuzhen Huang, Arun Kejariwal, Kannan Ramchandran, and Michael W. Mahoney. 2021. Training Recommender Systems at Scale: Communication-Efficient Model and Data Parallelism. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2928–2936. https://doi.org/10.1145/3447548.3467080

[22] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 243–254.

[23] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

[24] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 620–629.

[25] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569

[26] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[27] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[28] Samuel Hsia, Udit Gupta, Mark Wilkening, Carole-Jean Wu, Gu-Yeon Wei, and David Brooks. 2020. Cross-Stack Workload Characterization of Deep Recommendation Systems. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*. 157–168. https://doi.org/10.1109/IISWC50251.2020.00024

[29] Yuzhen Huang, Xiaohan Wei, Xing Wang, Jiyan Yang, Bor-Yiing Su, Shivam Bharuka, Dhruv Choudhary, Zewei Jiang, Hai Zheng, and Jack Langman. 2021. Hierarchical Training: Scaling Deep Recommendation Models on Large CPU Clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 3050–3058. https://doi.org/10.1145/3447548.3467084

[30] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: A Chiplet-Based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA '20)*. IEEE Press, 968–981. https:

//doi.org/10.1109/ISCA45697.2020.00083

[31] Taylor Gordon Ivan Medvedev, Haotian Wu. 2019. Powered by AI: Instagram's Explore recommender system. https://ai.facebook.com/blog/powered-by-ai-instagrams-explore-recommender-system/

[32] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[33] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, et al. 2021. MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions. *Proceedings of Machine Learning and Systems* 3 (2021).

[34] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 3097–3105. https://doi.org/10.1145/3447548.3467139

[35] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–12.

[36] Criteo Kaggle. 2014. Display Advertising Challenge: Predict click-through rates on display ads. https://www.kaggle.com/c/criteo-display-ad-challenge

[37] Wang-Cheng Kang and Julian McAuley. 2019. Candidate Generation with Binary Codes for Large-Scale Top-N Recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1523–1532.

[38] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 790–803.

[39] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, Yeongon Cho, Jin Hyun Kim, Yongsuk Kwon, Kyungsoo Kim, Jin Jung, Ilkwon Yun, Sung Joo Park, Hyunsun Park, Joonho Song, Jeonghyeon Cho, Kyomin Sohn, Nam Sung Kim, and Hsien-Hsin Sean Lee. 2021. Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM. *IEEE Micro* (2021), 1–1. https://doi.org/10.1109/MM.2021.3097700

[40] Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Tensor Reduction in Memory. *IEEE Computer Architecture Letters* 20, 1 (2021), 5–8. https://doi.org/10.1109/LCA.2020.3042805

[41] Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, and Vikas Chandra. 2019. Herald: Optimizing heterogeneous dnn accelerators for edge devices. *arXiv preprint arXiv:1909.07437* (2019).

[42] 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*. 740–753.

[43] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2020. Tensor Casting: Co-Designing Algorithm-Architecture for Personalized Recommendation Training. arXiv:cs.AR/2010.13100

[44] Shih-Hsiang Lin, Pei-Yin Chen, and Yu-Ning Lin. 2017. Hardware Design of Low-Power High-Throughput Sorting Unit. *IEEE Trans. Comput.* 66, 8 (2017), 1383–1395. https://doi.org/10.1109/TC.2017.2672966

[45] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2020. Understanding Capacity-Driven Scale-Out Neural Recommendation Inference. *arXiv preprint arXiv:2011.02084* (2020).

[46] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, et al. 2020. Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems. *arXiv preprint arXiv:2003.09518* (2020).

[47] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[48] Lillian Pentecost, Marco Donato, Brandon Reagen, Udit Gupta, Siming Ma, Gu-Yeon Wei, and David Brooks. 2019. Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 769–781.

[49] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 267–278.

[50] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[51] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[52] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 165–175.

[53] Franyell Silfa, Gem Dot, Jose-Maria Arnau, and Antonio Gonzalez. 2017. E-PUR: An Energy-Efficient Processing Unit for Recurrent Neural Networks. (2017). arXiv:1711.07480 https://arxiv.org/pdf/1711.07480.pdf

[54] Linghao Song, Jiachen Mao, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. 2019. Hypar: Towards hybrid parallelism for deep learning accelerator array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 56–68.

[55] Hu Wan, Xuan Sun, Yufei Cui, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. 2021. FlashEmbedding: Storing Embedding Tables in SSD for Large-Scale Recommender Systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '21)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/3476886.3477511

[56] 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 *26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[57] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. 2020. Kraken: Memory-Efficient Continual Learning for Large-Scale Real-Time Recommendations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*. IEEE Press, Article 21, 17 pages.

[58] Xinyang Yi, Yi-Fan Chen, Sukriti Ramesh, Vinu Rajashekhar, Lichan Hong, Noah Fiedel, Nandini Seshadri, Lukasz Heldt, Xiang Wu, and Ed H. Chi. 2018. Factorized Deep Retrieval and Distributed TensorFlow Serving *(SysML'18)*.

[59] Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. 2021. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models *(MLSys'21)*.

[60] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-switching: Dealing with fluctuating workloads in machine-learning-as-a-service systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.

[61] Jeff (Jun) Zhang, Parul Raj, Shuayb Zarar, Amol Ambardekar, and Siddharth Garg. 2019. CompAct: On-Chip ComPression of ActIvations for Low Power Systolic Array Based CNN Acceleration. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 47 (Oct. 2019), 24 pages.

[62] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783723

[63] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. arXiv:cs.DC/2003.05622

[64] 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 (CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 319–328.

[65] 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 (RecSys '19)*. ACM, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997

[66] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5941–5948.

[67] 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*. ACM, 1059–1068.