DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference

Udit Gupta[1,2], Samuel Hsia[1], Vikram Saraph[2], Xiaodong Wang[2], Brandon Reagen[2], Gu-Yeon Wei[1], Hsien-Hsin S. Lee[2], David Brooks[1,2], Carole-Jean Wu[2]

[1]Harvard University [2]Facebook Inc.

ugupta@g.harvard.edu carolejeanwu@fb.com

***Abstract*—Neural personalized recommendation is the cornerstone of a wide collection of cloud services and products, constituting significant compute demand of cloud infrastructure. Thus, improving the execution efficiency of recommendation directly translates into infrastructure capacity saving. In this paper, we propose DeepRecSched, a recommendation inference scheduler that maximizes latency-bounded throughput by taking into account characteristics of inference query size and arrival patterns, model architectures, and underlying hardware systems. By carefully optimizing task versus data-level parallelism, DeepRecSched improves system throughput on server class CPUs by 2× across eight industry-representative models. Next, we deploy and evaluate this optimization in an at-scale production datacenter which reduces end-to-end tail latency across a wide variety of recommendation models by 30%. Finally, DeepRecSched demonstrates the role and impact of specialized AI hardware in optimizing system level performance (QPS) and power efficiency (QPS/watt) of recommendation inference.**

In order to enable the design space exploration of customized recommendation systems shown in this paper, we design and validate an end-to-end modeling infrastructure, DeepRecInfra. DeepRecInfra enables studies over a variety of recommendation use cases, taking into account at-scale effects, such as query arrival patterns and recommendation query sizes, observed from a production datacenter, as well as industry-representative models and tail latency targets.

I. Introduction

Recommendation algorithms are used pervasively to improve and personalize user experience across a variety of web-services. Search engines use recommendation algorithms to order results, social networks to suggest posts, e-commerce websites to suggest purchases, and video streaming services to recommend movies. As their sophistication increases with more and better quality data, recommendation algorithms have evolved from simple rule-based or nearest neighbor-based designs [1] to deep learning approaches [2]–[7].

Deep learning-based personalized recommendation algorithms enable a plethora of use cases [8]. For example, Facebook's recommendation use cases require more than 10× the datacenter inference capacity compared to common computer vision and natural language processing tasks [9]. As a result, over 80% of machine learning inference cycles at Facebook's datacenter fleets are devoted to recommendation and ranking inference [10]. Similar capacity demands can be found at Google [11], Amazon [8], [12], and Alibaba [5], [6]. And yet, despite their importance and the significant research on optimizing deep learning based AI workloads [13]–[17] from the systems and architecture community, relatively little attention has been devoted to solutions for recommendation [18]. In fact, deep learning-based recommendation inference poses unique challenges that demand unique solutions.

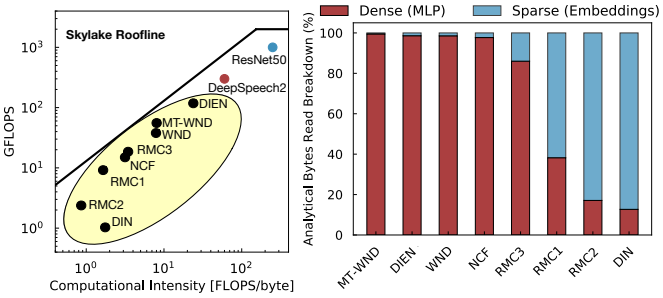

Fig. 1: State-of-the-art recommendation models span diverse performance characteristics compared to CNNs and RNNs. Based on their use case, recommendation models have unique architectures introducing model-level heterogeneity.

First, recommendation models exhibit unique compute, memory, and data reuse characteristics. Figure 1(a) compares the compute intensity of industry-representative recommendation models[1] [2]–[7], [10] to state-of-the-art convolutional (CNN) [19] and recurrent (RNN) neural networks [20]. Compared to CNNs and RNNs, recommendation models, highlighted in the shaded yellow region, tend to be memory intensive as opposed to compute intensive. Furthermore, recommendation models exhibit higher storage requirements (GBs) and irregular memory accesses [10]. This is because recommendation models operate over not only continuous but also categorical input features. Compared to the continuous features (i.e., vectors, matrices, images), categorical features are processed by inherently different operations. This unique characteristic of recommendation models exposes new system design opportunities to enable efficient inference.

Next, depending on the use case, major components of a recommendation model can be sized differently [21]. This

[1]Section III describes the eight recommendation models in detail.

introduces model-level heterogeneity across the state-of-the-art deep learning-based recommendation models. By focusing on memory access breakdown, Figure 1(b) shows diversity among recommendation models themselves. For instance, dense feature processing that incurs *regular* memory accesses dominate for Google's WnD [4], [7], NCF [2], Facebook's DLRM-RMC3 [10], and Alibaba's DIEN [6]. In contrast, categorical, sparse feature processing that incurs *irregular* memory accesses dominate for other recommendation models such as Facebook's DLRM-RMC1/RMC2 [10] and Alibaba's DIN [5]. These diverse characteristics of recommendation models expose system optimization design opportunities.

Finally, recommendation models are deployed across web-services that require solutions to consider effects of executing at-scale in datacenters. For instance, it is commonly known that requests for web-based services follow Poisson and log-normal distributions for arrival and working set size respectively [22]. Similar characteristics are observed for arrival rates of recommendation queries. However, query working set sizes for recommendation follow a distinct distribution with heavier tail effects. This difference in query size distribution leads to varying optimization strategies for at-scale inference. Optimizations based on production query size distributions, compared to log-normal, improve system throughput by up to 1.7× for at-scale recommendation inference.

To enable design optimizations for the diverse collection of industry-relevant recommendation models, this paper presents *DeepRecInfra* – an end-to-end infrastructure that enables researchers to study at-scale effects of query size and arrival patterns. First, we perform an in-depth characterization of eight state-of-the-art recommendation models that cover commercial video recommendation, e-commerce, and social media [2], [4]–[7], [10]. Next, we profile recommendation services in a production datacenter to instrument an inference load generator for modeling recommendation queries.

Built on top of the performance characterization of the recommendation models and dynamic query arrival patterns (rate and size), we propose a hill-climbing based scheduler – *DeepRecSched* – that splits queries into mini-batches based on the query size and arrival pattern, the recommendation model, and the underlying hardware platform. DeepRecSched maximizes system load under a strict tail-latency target by trading off request versus batch-level parallelism. Since it is also important to consider the role of hardware accelerators for at-scale AI infrastructure efficiency, DeepRecSched also evaluates the impact of specialized hardware for neural recommendation by emulating its behavior running on state-of-art GPUs.

The important contributions of this work are:

1) This paper describes a new end-to-end infrastructure, DeepRecInfra, that enables system design and optimization across a diverse set of recommendation models. DeepRecInfra integrates query arrival patterns and size distributions, observed in a production datacenter. We highlight the importance of the unique query arrival and size characteristics for at-scale recommendation inference and identify a new performance optimization opportunity by exploiting request batching (Section III).
2) We propose DeepRecSched– the first batch-scheduler that (a) partitions work across CPUs and accelerators (GPU), (b) trades off batch- (data) and request- (task) parallelism. DeepRecSched is tailor-designed to take into account the dynamic query arrival patterns (rate and size), recommendation model architectures, and service-level latency targets (Section IV). Evaluated with DeepRecInfra, DeepRecSched doubles system throughput of server class CPUs under strict latency targets. In addition, we implement and evaluate the proposed design on a production datacenter with live recommendation query traffic, showing a 1.3× reduction in the tail latency.
3) We demonstrate that GPU accelerators can be appealing for recommendation inference. Given *not all queries are equal* in recommendation inference, this paper shows that the latency and throughput tradeoff between CPU and GPU execution varies across different models, system loads, and latency targets, highlighting the importance of DeepRecSched's dynamism to determine optimal configurations. We also show that, for recommendation inference, power efficiency is not always optimal in the face of GPUs, as compared to CPUs (Section VI).

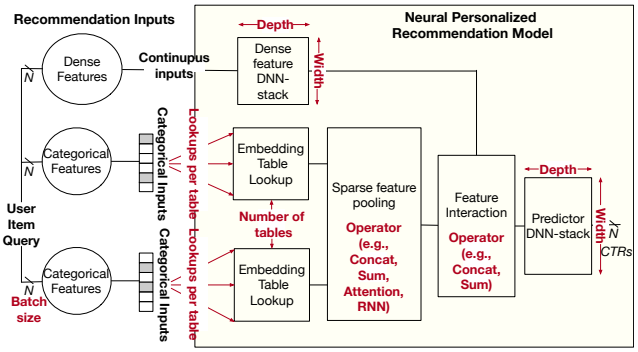

Fig. 2: General architecture of personalized recommendation models. Configuring the key parameters (red) yields different implementations of industry-representative models.

Systems research for personalized recommendation is still a nascent field. To enable follow-on work, we have **open sourced** the proposed DeepRecInfra infrastructure[2]. This includes the industry-representative neural recommendation models, and at-scale query arrival rates and size distributions presented in this paper.

II. Neural Recommendation Models

Recommendation is the task of personalizing recommending content based on a user's preferences. Recommendation is used across many services including search, video and movie content, e-commerce, and advertisements. However, accurately modeling preferences based on previous interactions can be challenging because users only interact with a small subset of all possible items. As a result, unlike inputs to traditional deep neural networks (DNNs), inputs to recommendation models include both *dense* and *sparse* features.

[2]http://vlsiarch.eecs.harvard.edu/research/recommendation

Model	Company	Domain	Dense-FC	Predict-FC	Embeddings		
					Tables	Lookup	Pooling
NCF [2]	-	Movies	-	256-256-128	4	1	Concat
Wide&Deep [4]	Google	Play Store	-	1024-512-256	Tens	1	Concat
MT-Wide&Deep [7]	Youtube	Video	-	N x (1024-512-256)	Tens	1	Concat
DLRM-RMC1 [10]	Facebook	Social Media	256-128-32	256-64-1	≤ 10	~ 80	Sum
DLRM-RMC2 [10]	Facebook	Social Media	256-128-32	512-128-1	≤ 40	~ 80	Sum
DLRM-RMC3 [10]	Facebook	Social Media	2560-512-32	512-128-1	≤ 10	~ 20	Sum
DIN [5]	Alibaba	E-commerce	-	200-80-2	Tens	Hundreds	Attention+FC
DIEN [6]	Alibaba	E-commerce	-	200-80-2	Tens	Tens	Attention+RNN

TABLE I: Architectural features of state-of-the-art personalized recommendation models.

A. Salient Components in Neural Recommendation Models

To accurately model user preference, state-of-the-art recommendation models use deep learning solutions. Figure 2 depicts a generalized architecture of DNN-based recommendation models with dense and sparse features as inputs.

Features. Dense features describe continuous inputs that are processed with MLP layers i.e., fully-connected layers – similar to classic DNN approaches. On the other hand, sparse features represent categorical inputs, such as the collection of products a user has previously purchased. Since the number of interactions for a categorical feature is often small compared to the feature's cardinality (all available products), the binary vector representing such interactions ends up very sparse.

Embedding Tables. Each sparse feature has a corresponding embedding table that is composed of a collection of latent embedding vectors. The number of vectors, or rows in the table, is determined by the number of categories in the given feature – this can vary from tens to billions. The number of elements in each vector is determined by the number of latent features for the category representation. This latent dimension is typically 16, 32, or 64. In total, embedding tables often require up to tens of GBs of storage.

Embedding Table Access. While embedding tables themselves are dense data structures, embedding operations incur sparse, irregular memory accesses. Each sparse input is encoded either as one-hot or multi-hot encoded vectors, which are used to index specific rows of an embedding table. The resulting embedding vectors are combined with a *sparse feature pooling* operation such as concatenation or sum.

Feature Interaction. The outputs of the dense and sparse features are combined before being processed by subsequent predictor-DNN stacks. Typical operations for feature interaction include concatenation, sum, and averaging.

Product Ranking. The output of the predictor-DNN stacks is the click through rate (CTR) probability for a single user-item pair. To serve relevant content to users, the CTR of thousands of potential items are evaluated for each user. All CTR's are then ranked and the top-N choices are presented to the user. As a result, deploying recommendation models requires running the models with non-unit batch sizes.

III. DeepRecInfra: At-scale Recommendation

To better understand the distinct characteristics of and design system solutions for neural recommendation models, we developed, DeepRecInfra, to model and evaluate at-scale recommendation inference. DeepRecInfra is implemented as a highly extensible framework enabling us to consider a variety of recommendation use cases. In particular, DeepRecInfra consists of three important components: (1) a suite of industry-representative models, (2) industry-representative tail latency targets, and (3) real-time query serving based on arrival rates and working set size distributions profiled from recommendation running in a production datacenter.

A. Industry-scale recommendation models

Recent publications from Google, Facebook, and Alibaba present notable differences across their recommendation models [2], [5]–[7], [10]. The generalized recommendation model architecture shown in Figure 2 can be customized by configuring formative parameters in order to realize these different implementations. To capture the diversity, DeepRecInfra composes a collection of eight state-of-the-art recommendation models. We describe the unique aspects of each model below and summarize their distinguishing parameters in Table I.

- **Neural Collaborative Filtering (NCF)** generalizes matrix factorization (MF) techniques proposed via the Netflix Prize [23] [24] with MLPs and non-linearities. NCF implements one-hot encoded sparse features, four embedding tables, and MF based sparse pooling.
- **Wide and Deep (WnD)** considers both *sparse* and *dense* input features and is deployed in Google's Play Store [4]. Dense input features are directly concatenated with the output of one-hot encoded embedding lookups and a relatively large Predict-FC stack produces output CTRs.
- **Multi-Task Wide and Deep (MT-WnD)** extends WnD by evaluating multiple output objectives including CTR, comment rate, likes, and ratings using a separate Predict-FC stack for each objective. Leveraging multi-objective modeling, MT-WnD enables a finer grained and improved user experience [25].
- **Deep Learning Recommendation Model (DLRM)** is a set of models from Facebook that differs from the aforementioned examples with its large number of embedding lookups [3]. Based on the configurations shown in [10] varying the number of lookups per table and size of FC layers yield three different architectures, DLRM-RMC1, DLRM-RMC2, and DLRM-RMC3.
- **Deep Interest Network (DIN)** uses attention to model user interests. DIN does not consider dense input features but has tens of embedding tables of varying sizes. Smaller embedding tables process one-hot encoded inputs while larger ones (up to 10^9 rows) process multi-hot encoded inputs with hundreds of lookups. Outputs of these embedding operations

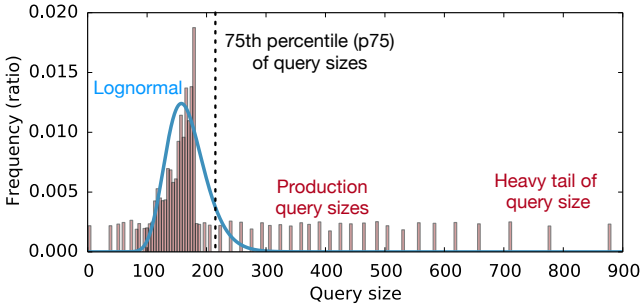

Fig. 3: Queries for personalized recommendation models follow a unique distribution not captured by traditional workload distributions (i.e. normal, log-normal) considered for web-services. The heavy tail of query sizes found in production recommendation services leads to unique design optimizations.

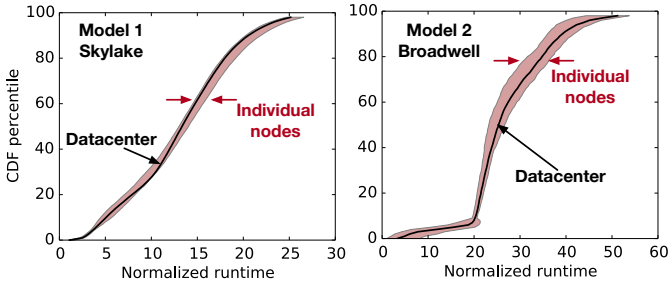

Fig. 4: Performance distribution of recommendation inference at datacenter scale to individual machines. Individual machines follow inference distributions, excluding network and geographic effects, at the datacenter scale to within $\sim 9\%$.

are combined as a weighted sum by a local activation unit (i.e., attention) and then concatenated [5].

- **Deep Interest Evolution Network (DIEN)** captures evolving user interests over time by augmenting DIN with gated recurrent units (GRUs) [6]. Inputs to the model are one-hot encoded sparse features. Embedding vectors are processed by attention-based multi-layer GRUs.

B. Service level requirement on tail latency

Personalized recommendation models are used in many Internet services deployed at a global scale. They must service a large number of requests across the datacenter while meeting strict latency targets set by the Service Level Agreements (SLAs) of various use cases. Thus, recommendation systems are optimized for *latency-bounded throughput* measured as the queries per second (QPS) that can be processed under a *p*95 tail-latency requirement. Across different applications (e.g., search, social-media, e-commerce) we find these latency targets vary significantly, which can result in distinct system design decisions. In this paper, we use the published targets and profiled model runtime to set the tail-latency target. Details on these tail latency targets are available in Section V.

C. Real-Time Query Serving for Recommendation Inference

DeepRecInfra takes into account two important dimensions of real-time query serving: arrival rate and working set sizes.

Query Arrival Pattern: Arrival times for queries for datacenter services are determined by the inter-arrival time between consecutive requests. This inter-arrival time can be modeled using a variety of distributions including uniform, normal or Poisson distributions [22], [26]–[29]. Previous work has shown that, these distributions can lead to different system design optimizations [22], [29]. Following web-services, by profiling the statistical distribution of recommendation services in a production datacenter, we find that query arrival rates follow a Poisson distribution [22], [26]–[28], [30].

Query Working Set Size Pattern: *Not all recommendation queries are created equal.* The size of queries for recommendation inference relates to the number of items to be ranked for a given user. This translates to the amount of work per inference. Given the potential number of items to be served depends heavily on users' interaction with the web-service, query sizes vary. While new hardware and software optimizations are typically evaluated across an array of fixed size batches, this is *not* the same as optimizing systems where batch-sizes vary dynamically (i.e., recommendation). To maximize efficiency of systems with dynamic batch-sizes, is it important to optimize for the particular working set size distribution (see Section IV).

Related work on designing system solutions for web services typically assumes working set sizes of queries follow a fixed, normal, or log-normal distribution [22]. However, Figure 3 illustrates that query sizes for recommendation exhibit a heavier tail compared to canonical log-normal distributions. Thus, while DeepRecInfra's load generator supports a variety of distributions, the remainder of this paper uses the query size distribution found in a production datacenter (Figure 3).

D. Subsampling datacenter fleet with single-node servers

To serve potentially billions of users across the world, recommendation models are typically run across thousands of machines. However, it may not always be possible to deploy design optimizations across a production-scale datacenter. We show a handful of machines can be used to study and optimize tail performance of recommendation inference. Figure 4 shows the cumulative distribution of two different recommendation models running on server-class Intel Skylake and Broadwell machines. We find that the datacenter scale performance (black) is within 10% of the distribution measured on a handful of machines (red). Thus, tail-latency trends for recommendation inference across a subset of machines can be representative of larger scale systems.

E. Putting it Altogether

To study at-scale characteristics of recommendation, it is important to use representative infrastructure. This includes representative models, query arrival rates, and query working set size distributions. Thus, we developed DeepRecInfra, shown in Figure 5, by incorporating an extensible load generator to model query arrival rate and size patterns for recommendation use cases. This enables efficient and representative design space exploration catered to at-scale recommendation.

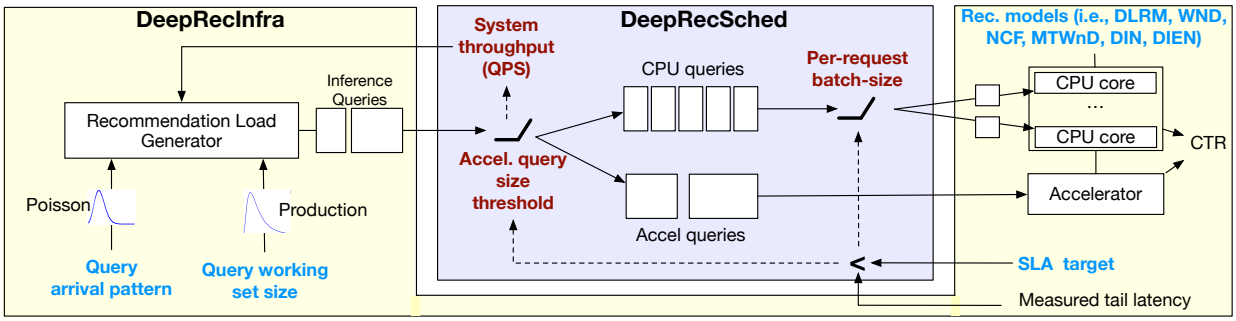

Fig. 5: DeepRecInfra implements an extensible framework that considers industry-representative recommendation models, application level tail latency targets, and real-time query serving (rate and size). Built upon DeepRecInfra, DeepRecSched optimizes system throughput (QPS) under strict latency targets by optimzing per-request batch-size (request versus batch parallelism) and accelerator query size threshold (parallelizing queries across specialized hardware).

DeepRecInfra is designed to enable future research for at-scale recommendation inference. First, model architecture parameters are specified at the command line. This includes parameters such as the width/depth of DNN layers, number of continuous and categorical input features, number of rows and columns in embedding tables, number of sparse IDs per lookup, and interaction operations between continuous and categorical input features. Next, given the example implementations, additional recommendation models can be added to the infrastructure. Finally, to model various at-scale effects, users can configure the degree of model co-location, tail latency targets, and query arrival rates and sizes.

IV. DeepRecSched Design

In this section, we present the design, implementation, and evaluation of the proposed design – DeepRecSched– a recommendation inference scheduler that optimizes latency-bounded throughput for at-scale execution. Central to DeepRecSched is the observation that working set sizes for recommendation queries follow a unique distribution with a heavy tail. Intuitively, large queries limit the throughput (QPS) a system can handle given a strict latency target. DeepRecSched is tailor-designed to address this bottleneck with two design optimizations. First is exploiting batch (data) versus request (task) level parallelism. This is accomplished by splitting large queries into multiple *requests* of smaller batch size; *requests* are processed by parallel cores. This requires carefully balancing batch-level and SIMD-level parallelism, cache contention, and the potential increase in queuing delay from a larger number of smaller-sized requests. Second, large queries are offloaded to specialized AI hardware in order to accelerate at-scale recommendation inference. The decision to offload queries onto specialized AI hardware must carefully balance data communication costs, and parallelism across both server class CPU cores and the accelerator for each model.

DeepRecSched optimizes system throughput across the large and complex design space of recommendation use cases encompassed by DeepRecInfra (i.e., models, latency targets, query serving, hardware platforms). This is accomplished by tuning the per-core batch-size (i.e., balancing batch versus request-level parallelism on CPUs) and accelerator query size threshold (i.e., offloading larger queries to specialized hardware). In order to tune these parameters across the diverse set of recommendation use cases, DeepRecSched implements a hill-climbing based scheduling policy. The scheduler provides an effective solution, given our observation of the convexity of batch-size and accelerator partitioning problem, over more complex control theoretic approaches, such as PID [31], [32]. We motivate the need for automated solutions given the apparent model diversity (Section IV-A), optimize batch versus request parallelism (Section IV-B), and modulate the query offloading degree for specialized hardware (Section IV-C). Figure 5 illustrates the proposed DeepRecSched design in the context of DeepRecInfra.

A. Model diversity demands flexible optimization

The apparent diversity of the industry-representative recommendation models leads to varying, unique performance bottlenecks. Figure 6 compares the performance characteristics of recommendation models running on a server class Intel Broadwell, shown as fractions of time spent on Caffe2 operators for a fixed batch size of 64. As expected, inference runtime for models with high degrees of dense feature processing (i.e., DLRM-RMC3, NCF, WND, MT-WND) is dominated by the MLP layers. On the other hand, inference runtime for models dominated by sparse feature processing (i.e., DLRM-RMC1 and DLRM-RMC2) is dominated by embedding table lookups.

Interestingly, inference runtime for attention based recommendation models is dominated by neither FC nor embedding table operations. For instance, inference run time for DIN is split between concatenation, embedding table, sum, and FC operations. This is a result of the attention units, which (1) concatenate user and item embedding vectors, (2) perform a small FC operation, and (3) use the output of the FC operation to weight the original user embedding vector. Similarly, the execution time of DIEN is dominated by recurrent layers. This is a result of fewer embedding table lookups whose outputs are processed by a series of relatively large attention layers.

This design space is further expanded considering the heterogeneity of CPUs found in production datacenters [9]. Recent work shows recommendation models are run on a

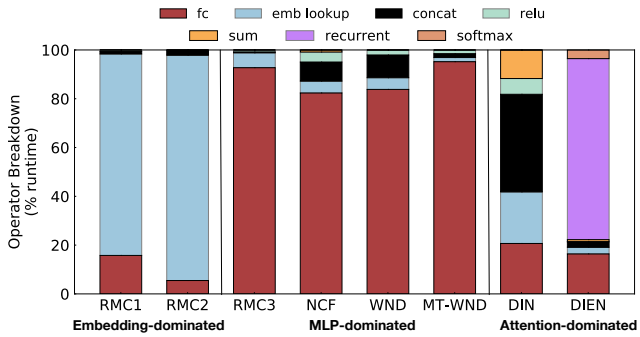

Fig. 6: Operator breakdown of state-of-the-art personalized recommendation models with a batch-size of 64. The large diversity in bottlenecks leads to varying design optimizations.

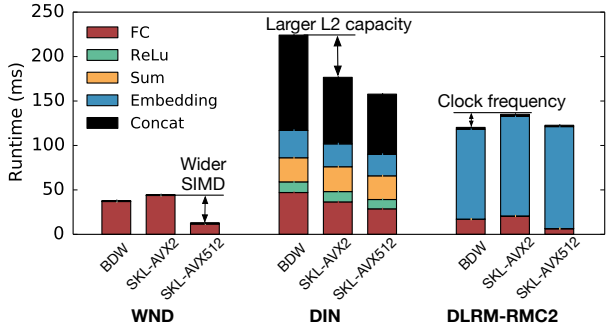

Fig. 7: Performance of WnD, DIN, and DLRM-RMC3 on Broadwell and Skylake, using AVX-2 and AVX-256 support. Performance variation across hardware platforms is due to difference in micro-architectural features such as SIMD-width, cache capacity, and clock frequency.

variety of server class CPUs such as Intel Broadwell and Skylake [10]. While, Broadwell implements CPUs running at 2.4GHz with AVX-256 SIMD units and inclusive L2/L3 cache hierarchies, Skylake cores run at 2.0GHz with AVX-512 units and exclusive caches with a larger effective cache capacity. Figure 7 shows the impact of CPU micro-architecture on neural recommendation inference performance. We show the performance of WnD, DIN, and DLRM-RMC2 on Broadwell (BDW), as well as Skylake using both AVX-256 (SKL-AVX2) and AVX-512 (SKL-AVX512) instructions. Given the fixed operating frequency and cache hierarchy between SKL-AVX2 and SKL-AVX512, the 3.0× performance difference for WnD can be attributed to the better utilization of the SIMD units. Similarly, given the fixed SIMD width, the 1.3× performance difference between BDW and SKL-AVX2 is a result of the larger L2 caches that help accelerate the Concat operator with highly regular memory access pattern. Finally, the performance difference between BDW and SKL-AVX2 instructions on DLRM-RMC2 is attributed to a 20% difference in core frequency accelerating the embedding table operations.

Given the variety of operator and system bottlenecks, an important design feature of DeepRecSched is to automatically optimize request- versus batch-level parallelism and leverage parallelism with specialized hardware.

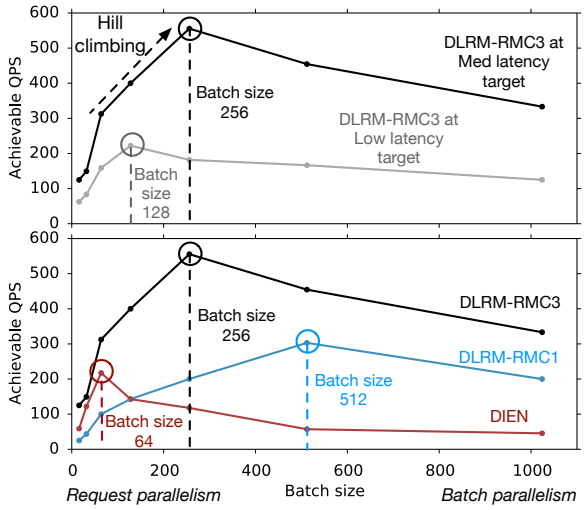

Fig. 8: Optimal request vs. batch parallelism varies based on the use case. Optimal batch-size varies across latency targets for DLRM-RMC (top) and models (bottom) i.e., i.e., DLRM-RMC2 (embedding-dominated), DLRM-RMC3 (MLP-dominated), DIEN (attention-dominated).

B. Optimal batch size varies

While all queries can be processed by a single core, splitting queries across cores to exploit hardware parallelism, is often advantageous. Thus, DeepRecSched splits queries into individual *requests*. However, this sacrifices parallelism within a request with a decreased *batch size*.

The optimal batch size that maximizes the system QPS throughput varies based on (1) tail latency targets and (2) recommendation models. Figure8 shows the achievable system throughput (QPS) as we vary the per-core batch-size. Recall that small batch-sizes (request parallelism) parallelizes a single query across multiple cores while larger batch-sizes (batch parallelism) processes a query on a single core. Figure 8 (top) illustrates that, for DLRM-RMC3, the optimal batch size increases from 128 to 256 as the tail latency target is relaxed from 66*ms* (low) to 100*ms* (medium). (See Section V for more details on tail-latency targets.) Furthermore, Figure 8(bottom) shows that the optimal batch size for DIEN (attention-based), DLRM-RMC3 (FC heavy), and DLRM-RMC1 (embedding table heavy) is 64, 128, and 256, respectively.

Note that the design space is further expanded when optimizing across the heterogeneous hardware platforms [9]. Following Figure 7, micro-architectural features across these servers can impact the optimum tradeoff between request- and batch-level parallelism. For example, higher batch sizes are typically required to exploit the benefits of the wider SIMD units in Intel Skylake [10]. Next, while inclusive cache (i.e., Broadwell) hierarchies simplify cache coherence protocols, they are more susceptible to cache contention and performance degradation from parallel cores [33], [34]. In the context of recommendation, this can be achieved by trading off request for batch parallelism. Section VI provides a more detailed analysis into the implication of hardware heterogeneity on

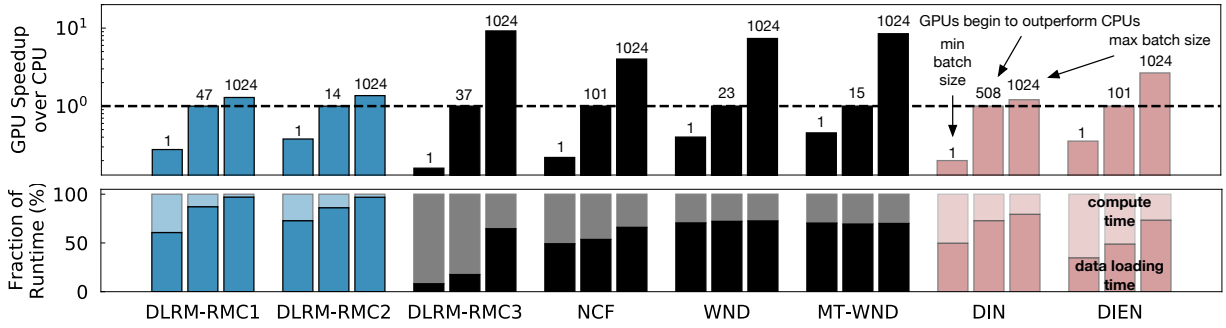

Fig. 9: GPU speedup over CPU for representative recommendation models. The batch-size at which GPUs start to outperform CPUs and their speedup at large batch-sizes varies across models.

trading off request- versus batch-level parallelism.

C. Leverage parallelism with specialized hardware

In addition to balancing request- versus batch-level parallelism on general purpose CPUs, in the presence of specialized AI hardware, DeepRecSched improves system throughput by offloading queries that can best leverage parallelism in the available specialized hardware. We evaluate the role of accelerators with state-of-the-art GPUs.

Figure 9(top) illustrates the speedup of GPUs over a CPU (single threaded) across the recommendation models at various batch sizes. For each model, we illustrate the relative performance of GPUs over CPUs at a unit batch-size, the batch-size required to outperform CPU-only hardware platforms, and a large batch-size of 1024. Given the higher compute intensity and memory bandwidth, GPUs provide significant performance benefits at higher batch sizes — especially for compute intensive models. However, across the different classes of recommendation models, there is large variation in (1) speedup at large batch sizes (i.e. 1024) and (2) batch size required to outperform CPU-only hardware platforms vary widely. This is due to the overhead of transferring inputs from the CPU to the GPU, which consumes a significant fraction of time. For instance, as shown in Figure 9(bottom), across all batch sizes, data loading time consumes on average 60∼80% of the end-to-end inference time on the GPU for all models.

In addition to considering performance characteristics of recommendation models on standalone systems, it is important to analyze the impact of the dynamic query working set size distributions. Figure 10 illustrates the execution time breakdown for queries smaller than the p75th size versus larger queries. Despite the long tail, the collection of small queries constitute over half the CPU execution time. *25% of large queries contribute to nearly 50% of total execution time.* This unique query size distribution with a long tail makes GPUs an interesting accelerator target. Figure 10 shows that, across all models, GPU can effectively accelerate the execution time of large queries. While offloading the large queries can reduce execution time, the amount of speedup varies based on the model architecture. The optimal threshold for offloading varies across models, motivating a design that can automatically tune the offloading decision for recommendation inference.

Trading off processing queries on CPUs versus GPUs requires careful optimization. Intuitively, offloading queries to the GPU incurs significant data transfer overheads. To amortize this cost, GPUs often require larger batch sizes to exhibit speedup over CPUs, as shown in Figure 9 [35]. Consequently, DeepRecSched improves system throughput by offloading the largest queries for recommendation inference to the GPU. This can be accomplished by tuning the *query-size* threshold. Queries larger than this threshold are offloaded to the GPU while smaller ones are processed on CPU cores.

Figure 11 illustrates the impact of query-size threshold (x-axis) on the achievable QPS (y-axis) across a variety of recommendation models. The optimal threshold varies across the three recommendation models, DLRM-RMC3, DLRM-RMC1, and DIEN. In fact, we find that the threshold not only varies across model architectures, but also across tail latency targets. Note, the optimal query size thresholds take into account the dynamic working set distribution found in Figure 3. Compared to the batch-size at which the GPU demonstrates speedup over the CPU (Figure 9), the optimal query-size threshold for both DLRM-RMC1 and DIEN are higher — 47 vs. 320 for DLRM-RMC1 and 101 vs. 512 for DIEN. Thus, systems that optimize recommendation inference by offloading work to specialized AI accelerators must consider the dynamic working set sizes — a salient feature of DeepRecSched.

D. DeepRecSched Design Summary

Recall that production datacenters run a variety of recommendation models that evolve over time across heterogeneous hardware platforms with varying SLA targets [9]. Thus, automated solutions are needed to optimize batch- and request-level parallelism, and offloading inference queries to specialized hardware. We tailor-design DeepRecSched for at-scale recommendation inference. For example, given the convexity of the batch-size and accelerator offloading optimization problem seen in Figure 8 and Figure 11, DeepRecSched implements a hill-climbing based algorithm. Compared to more complex control-theoretic approaches, such as PID controllers, hill-climbing offers a simple, scalable, and effective solution. In fact, we demonstrate the efficacy of DeepRecSched, by not only evaluating it using DeepRecInfra but also deploying it in a real production datacenter fleet (see Section VI for details).

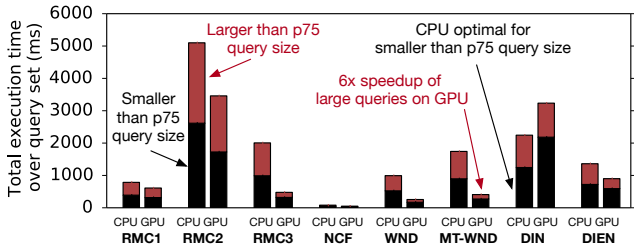

Fig. 10: Aggregated execution time over the query set based on the size distribution for CPU and GPU. GPUs readily accelerate larger queries; however, the optimal inflection point and speedup differ across models.

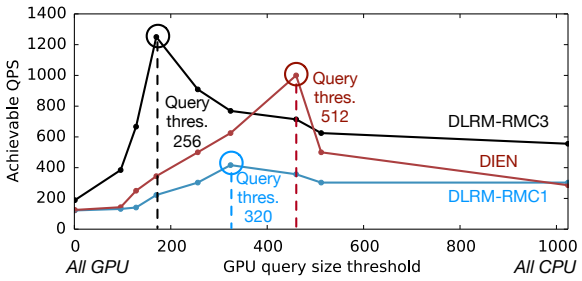

Fig. 11: The optimal query size threshold, and thus fraction of queries processed by the GPU, varies across recommendation models i.e., DLRM-RMC2 (embedding-dominated), DLRM-RMC3 (MLP-dominated), DIEN (attention-dominated).

Given a particular recommendation model, hardware platform, and tail latency target, DeepRecSched first tunes the tradeoff between batch- versus request-level parallelism. Starting with a unit batch-size, DeepRecSched gradually increases the per-core batch-size in order to optimize system throughput. Note that DeepRecSched is designed to perform the hill-climbing based optimization during the initial warm up period of launching a service. DeepRecSched then tunes the query-size threshold for offloading queries to specialized hardware. Starting with a unit query-size threshold (i.e., all queries are processed on the accelerator), DeepRecSched again applies hill-climbing to gradually increase the threshold until the achievable QPS degrades. As what Section VI later shows, by automatically tuning the per-request batch size and GPU query-size threshold, DeepRecSched optimizes infrastructure efficiency of at-scale recommendation across a variety of different model architectures, tail latency targets, query-size distributions, and the underlying hardware.

V. Methodology

We implement and evaluate DeepRecSched with DeepRecInfra across a variety of different hardware systems and platforms. We then compare the performance and power efficiency results with a production-scale baseline.

DeepRecInfra comprises three notable components:

- Model Implementation: We implement all the recommendation models (Table I) in Caffe2 with Intel MKL as the backend library for CPUs [36] and CUDA/cuDNN 10.1 for GPUs [37]. All CPU experiments are conducted with a single Caffe2 worker and Intel MKL thread.
- SLA Latency Targets: Table II presents the tail latency targets for each of the recommendation models [4]–[7], [10]. For instance, the Google Play store imposes an SLA target of tens of milliseconds on WnD [4], [11]. On the other hand, Facebook's social media platform requires DLRM-RMC1, DLRM-RMC2, and DLRM-RMC3 have an SLA target of hundreds of milliseconds [10]. Alibaba's e-commerce platform requires DIN and DIEN have an SLA target of tens of milliseconds [5], [6]. To explore the design tradeoffs over a range of latency targets, we consider three latency targets for each model — *Low*, *Medium*, and *High* — where Low and High tail latency targets are set to be 50% lower and 50% higher than that of Medium, respectively.
- Real-Time Query Patterns: Query patterns in DeepRecInfra are configurable on two axes: arrival rate and size. The arrival pattern is fit to a Poisson distribution whereas sizes are drawn from the production distribution (Figure 3).

Model	Runtime Bottleneck	SLA target
DLRM-RMC1	Embedding dominated	100*ms*
DLRM-RMC2	Embedding dominated	400*ms*
DLRM-RMC3	MLP dominated	100*ms*
NCF	MLP dominated	5*ms*
WND	MLP dominated	25*ms*
MT-WND	MLP dominated	25*ms*
DIN	Embedding + Attention dominated	100*ms*
DIEN	Attention-based GRU dominated	35*ms*

TABLE II: Summarizing performance implications of different personalized recommendation and latency targets used to illustrate design space tradeoffs for DeepRecSched.

Experimental System Setup. To consider the implications of hardware heterogeneity found in datacenter [9], [10], [38], we evaluate DeepRecSched with two generations of dual-socket server-class Intel CPUs: Broadwell and Skylake. Broadwell comprises 28 cores running at 2.4GHz with AVX-2 SIMD units and implements an inclusive L2/L3 cache hierarchy. Its TDP is of 120W. Skylake comprises of 40 cores running at 2.0GHz with AVX-512 SIMD units and implements an exclusive L2/L3 cache hierarchy. Its TDP is of 125W.

To consider the implications of AI hardware accelerators, we extend the design space to take into account a GPU accelerator model based on real empirical characterization. The accelerator performance model is constructed with the performance profiles of each recommendation model across the range of query sizes over a real-hardware GPU — server-class NVIDIA GTX 1080Ti with 3584 CUDA cores, 11GB of DDR5 memory, and optimized cuDNN backend library (see Figure 9). This includes both data loading and model computation [39]–[44], capturing the performance-critical components of the end-to-end recommendation inference.

Production-scale baseline. We compare DeepRecSched to the baseline that implements a fixed batch size configuration. This fixed batch size configuration is typically set by splitting the largest query *evenly* across all available cores on the underlying hardware platform. Given the maximum query

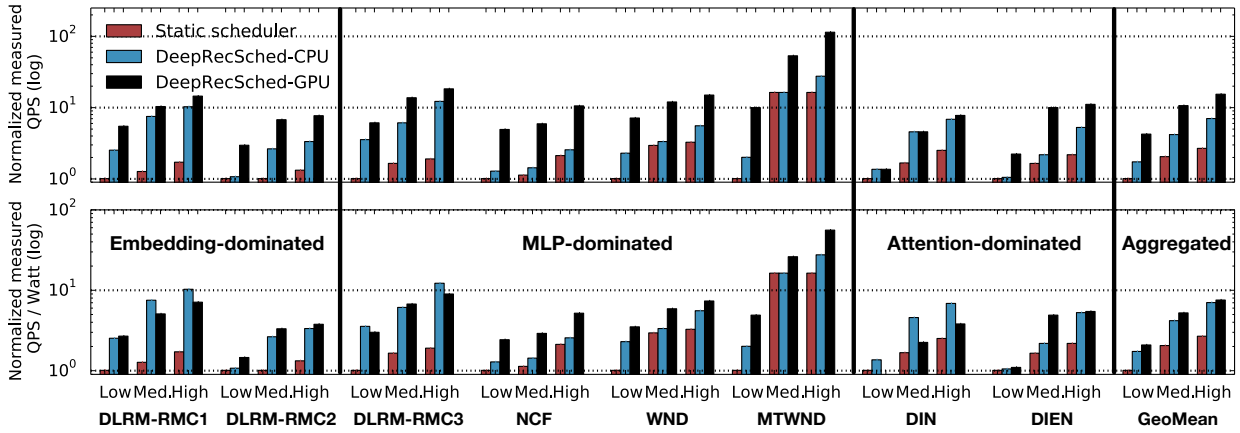

Fig. 12: Compared to a static scheduler based on production recommendation services, the top figure shows performance, measured in system throughout (QPS) across a range of latency targets, while the bottom shows power efficiency (QPS/Watt), for DeepRecSched-CPU and DeepRecSched-GPU.

size of 1000 (Figure 3), the static batch size configuration is determined as 25 for a server-class 40-core Intel Skylake.

VI. DeepRecSched Evaluation

This section presents the performance and power efficiency improvements of DeepRecSched running on CPUs (DeepRecSched-CPU) and GPUs (DeepRecSched-GPU) over the baseline across a vast and complex design space, including all eight state-of-the-art recommendation models using DeepRecInfra. Next, we detail the benefits of DeepRecSched by diving into (1) request- versus batch-level parallelism, (2) a case study of demonstrating the optimizations in a real production datacenter, and (3) parallelization opportunities of offloading requests to specialized hardware.

Performance. Figure 12(top) compares the throughput performance of DeepRecSched-CPU and DeepRecSched-GPU versus a baseline static scheduler across the three tail latency configurations, all normalized to the measured QPS at the *low* tail latency case of the baseline. Overall, DeepRecSched-CPU achieves 1.7×, 2.1×, and 2.7× higher QPS across all models for the *low*, *medium*, and *high* tail latency targets, respectively. DeepRecSched-CPU is able to increase the overall system throughput by optimizing batch size configuration. Furthermore, DeepRecSched-GPU increases performance improvement to 4.0×, 5.1×, and 5.8× at the *low*, *medium*, and *high* tail latency targets, respectively. Thus, parallelizing requests across general-purpose CPUs and specialized hardware provides additional performance improvement for recommendation.

Power efficiency. Figure 12(bottom) compares the QPS-per-watt power efficiency of DeepRecSched-CPU and DeepRecSched-GPU by again normalizing the measured QPS/Watt to the *low* tail latency case of the baseline static scheduler. Given higher performance under the TDP power budget as the baseline, DeepRecSched-CPU achieves 1.7×, 2.1×, and 2.7× higher QPS/Watt for all models under the *low*, *medium*, and *high* tail latency targets, respectively. Aggregated across all models, DeepRecSched-GPU improves the power efficiency improvement to 2×, 2.6×, and 2.9× for each latency target. Compared to the performance improvement, DeepRecSched-GPU provides marginal improvement in power efficiency due to the overhead of GPU acceleration. In fact, while DeepRecSched-GPU improves system QPS across all recommendation models and latency targets, compared to DeepRecSched-CPU, it does not globally improve QPS/Watt. In particular, the power efficiency improvement of DeepRecSched-GPU is more pronounced for compute intensive models (i.e., WND, MT-WND, NCF). On the other hand, for memory intensive models (i.e., DLRM-RMC1, DIN), the power overhead for offloading recommendation inference to GPUs outweighs the performance gain, degrading the overall power efficiency. Thus, judicious optimization of offloading queries across CPUs and specialized AI hardware can improve infrastructure efficiency for recommendation at-scale.

A. Balance of Request and Batch Parallelism

Here we take a deep into how DeepRecSched-CPU improves QPS by balancing request- versus batch-level parallelism across varying (1) latency targets, (2) query size distributions, (3) models, and (4) hardware platforms.

Optimizing across SLA targets. Figure 13(a) illustrates the tradeoff between request- and batch-level parallelism across varying tail latency targets for DLRM-RMC1. Under lower, stricter tail latency targets, QPS is optimized at lower batch sizes — favoring request level parallelism. On the other hand, at more relaxed tail latency targets, DeepRecSched-CPU finds the optimal configuration to be at a higher batch size — favoring batch-level parallelism. For instance, using the production working set size distribution, the optimal batch-size at target tail latencies of 60*ms* and 120*ms* are 128 and 1024 respectively. Intuitively, this is a result of achieving overall higher system throughput with larger batch-sizes at more relaxed latency targets. As shown in Figure 12(top), optimizing this per-request batch size yields DeepRecSched-CPU's QPS improvements over the static baseline across latency targets.

Optimizing across query size distributions Figure 13(a) also shows the optimal batch size, for DLRM-RMC1, varies

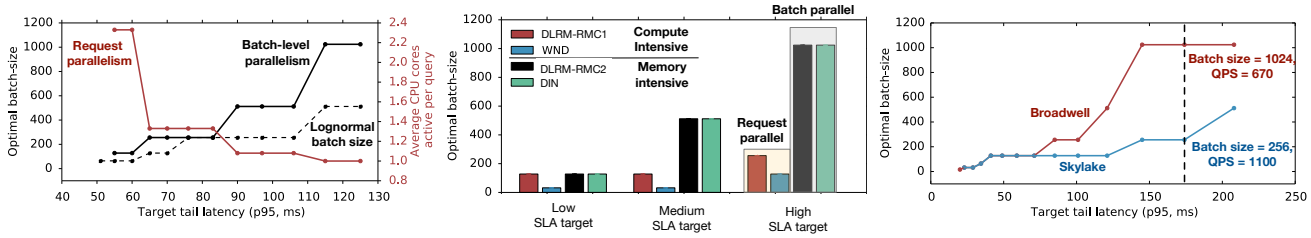

Fig. 13: Exploiting the unique characteristics of at-scale recommendation yields efficiency improvements given the optimal batch size varies across SLA targets and query size distributions (left), models (middle), and hardware platforms (right).

across query working set size distributions (lognormal and the production distribution). The optimal batch-size across all tail latency targets is strictly lower for lognormal than the query size distribution found in production recommendation use cases. This is a result of, as shown in Figure 3, query sizes in production recommendation use cases following a distribution with a heavier tail. In fact, applying optimal batch-size configuration based on the lognormal query size distribution to the production distribution degrades the performance of DeepRecSched-CPU by 1.2×, 1.4×, and 1.7× at low, medium, and high tail-latencies for DLRM-RMC1. Thus, built ontop of DeepRecInfra, DeepRecSched-CPU carefully optimizes request verus batch-level parallelism for recommendation inference in production datacenters.

Optimizing across recommendation models. Figure 13(b) illustrates that the optimal batch size varies across recommendation models with distinct compute and memory characteristics. Here, we consider two compute intensive models (e.g., DLRM-RMC3, WnD) and two memory intensive models (e.g., DLRM-RMC1, DIN). We find that compute intensive models are typically optimized with lower batch-sizes as compared to memory intensive models. For example, at the high SLA targets, DLRM-RMC3 and WnD have an optimal batch size of 256 and 128, respectively. On the other hand, DLRM-RMC1 and DIN are optimized at a larger batch size of 1024. This is a result of the compute intensive models being accelerated by the data-parallel SIMD units (i.e., AVX-512 in Intel Skylake, AVX-256 in Intel Broadwell). Thus, throughput for compute intensive models is maximized by fully utilizing the data-parallel SIMD units and parallelizing queries into multiple requests across the chip-multiprocessor cores.

While higher throughput is achieved at smaller batch-sizes for compute intensive models, memory intensive models require larger batch sizes. This is because the primary performance bottleneck of models with heavy embedding table accesses lies in the DRAM bandwidth utilization. In order to saturate, and fully utilize, the per-core memory bandwidth, memory intensive recommendation models must be run with higher batch-sizes. Thus, in addition to request level parallelism, memory bandwidth utilization can be improved significantly by running recommendation inference at a higher batch size. By exploiting characteristics of the models to optimize the per-request batch size, DeepRecSched-CPU achieves higher QPS across the various recommendation models.

Optimizing across hardware platforms. Figure 13(c) shows the optimal batch size, for DLRM-RMC3, varies across server architectures (Intel Broadwell and Skylake machines). The optimal batch size, across all tail-latency targets, is strictly higher on Intel Broadwell compared to Skylake. For example, at a latency target of 175*ms*, the optimal batch-size on Intel Broadwell and Skylake is 1024 and 256, respectively. This is a result of the varying cache hierarchies on the two platforms. In particular, Intel Broadwell implements an inclusive L2/L3 cache hierarchy while Intel Skylake implements an exclusive L2/L3 cache hierarchy. As a result, Intel Broadwell suffers from higher cache contention with more active cores leading to performance degradation. For example, at a latency target of 175*ms* and per-request batch sizes of 16 (request-parallel) and 1024 (batch-parallel), Intel Broadwell has an L2 cache miss rate of 55% and 40% respectively. To compensate for this performance penalty, DeepRecSched-CPU runs recommendation models with higher batch-sizes — fewer request and active cores per query — on Intel Broadwell.

Overall, DeepRecSched enables a fine balance between request vs. batch-level parallelism across not only varying tail latency targets, query size distributions, and recommendation models, but also the underlying hardware platforms.

B. Tail Latency Reduction for At-Scale Production Execution

Following the evaluations using DeepRecInfra, we deploy the proposed design and demonstrate that the optimizations translate to higher performance in a real production datacenter. Figure 14 illustrates the impact of varying the batch-size on the measured tail latency of recommendation models running in a production datacenter. Experiments are conducted using production A/B tests with a portion of real-time datacenter traffic to consider end-to-end system effects including load-balancing and networking. The A/B tests run on a cluster of hundreds of server-class Intel CPUs running a wide collection of recommendation models used in the production datacenter fleet. The baseline configuration is a fixed batch-size, deployed in a real production datacenter fleet, set coarsely optimizing for a large collection of models. Optimizing the batch- versus request-parallelism at a finer granularity, by taking into account particular model architectures and hardware platforms, enables further performance gains. To enable this finer granularity optimization and account for the diurnal production traffic asf well as intra-day query variability, we deploy and evaluate DeepRecSched over the course of 24 hours. Compared to the baseline configuration, the optimal

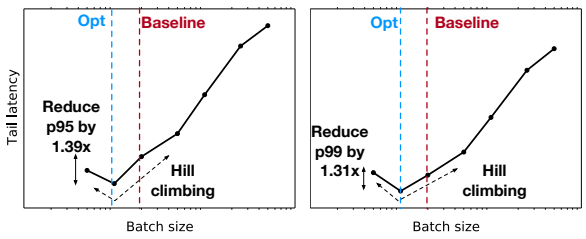

Fig. 14: Exploiting the request vs. batch-level parallelism optimization demonstrated by DeepRecSched in a real production datacenter improves performance of at-scale recommendation services. Across models and servers, optimizing batch size reduces p95 and p99 latency by 1.39× (left) and 1.31× (right).

batch size provides a 1.39× and 1.31× reduction in p95 and p99 tail latencies, respectively. This reduction in the tail latency can be used to increase system throughput (QPS) of the cluster of machines.

C. Leverage Parallelism with Specialized Hardware

In addition to trading off request vs. batch-level parallelism, DeepRecSched-GPU leverages additional parallelism by offloading recommendation inference queries to GPUs.

Performance improvements. GPUs are often treated as throughput-oriented accelerators. However, in the context of personalized recommendation, we find that GPUs can unlock lower tail latency targets unacheivable by CPUs. Figure 15(a) illustrates the performance impact of scheduling requests across both CPUs and GPUs. While the lowest achievable tail-latency targets for DLRM-RMC1 on CPUs is 57*ms*, GPUs can achieve a tail-latency target of as low as 41*ms* (1.4× reduction). This is a result of recommendation models exhibiting high compute and memory intensity, as well as the heavy tail of query sizes in production use cases (Figure 3).

Next, in addition to achieving lower tail latencies, parallelization across both the CPU and the specialized hardware increases system throughput. For instance, Figure 15(a) shows that across all tail-latency targets, DeepRecSched-GPU achieves higher QPS than DeepRecSched-CPU. This is as a result of the execution of the larger queries on GPUs, enabling higher system throughput. Interestingly, the percent of work processed by the GPU decreases with higher tail latency targets. This is due that, at a low latency target, DeepRecSched-GPU optimizes system throughput by setting a low query size threshold and offloads a large fraction of queries to the GPU. Under a more relaxed tail-latency constraint, more inference queries can be processed by the CMPs. This leads to a higher query size threshold for DeepRecSched-GPU. At a tail latency target of 120*ms*, the optimal query size threshold is 324 and the percent of work processed by the GPU falls to 18%. As shown in Figure 12(top), optimizing the query size threshold yields DeepRecSched-GPU's system throughput improvements over the static baseline and DeepRecSched-CPU across the different tail latency targets and recommendation models.

Infrastructure efficiency implications. While GPUs can enable lower latency and higher QPS, power efficiency is not always optimized with GPUs as the specialized AI accelerator. For instance, Figure 15(b) shows the QPS/Watt of both DeepRecSched-CPU and DeepRecSched-GPU for DLRM-RMC1, across a variety of tail latency targets. At low tail latency targets, QPS/Watt is maximized by DeepRecSched-GPU — parallelizing queries across both CPUs and GPUs. However, under more relaxed tail-latency targets, we find QPS/Watt is optimized by processing queries on CPUs only. Despite the additional power overhead of the GPU, DeepRecSched-GPU does not provide commensurate system throughput benefits over DeepRecSched-CPU at higher tail latencies.

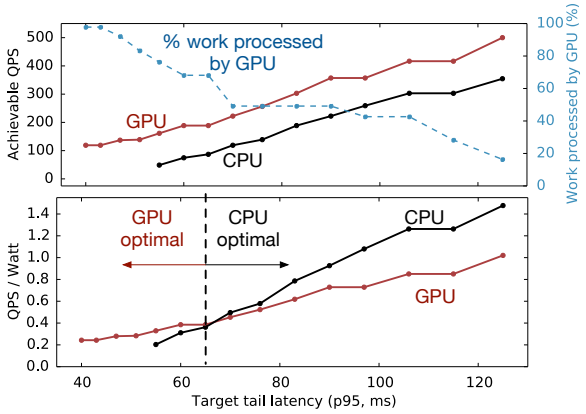

Fig. 15: (Top) System throughput increases by scheduling queries across both CPUs and GPUs. The percent of work processed by the GPU decreases at higher tail latency targets. (Bottom) While QPS strictly improves, the optimal configuration based on QPS/Watt, varies based on tail latency targets.

More generally, power efficiency is co-optimized by considering both the tail latency target and the recommendation model. For instance, Figure 12(b) illustrates the power efficiency for the collection of recommendation models across different tail latency targets. We find that DeepRecSched-GPU achieves higher QPS/Watt across all latency targets for compute-intensive models (i.e., NCF, WnD, MT-WnD) — the performance improvement of specialized hardware outweighs the increase in power footprint. Similarly, for DLRM-RMC2 and DIEN, DeepRecSched-GPU provides marginal power efficiency improvements. On the other hand, the optimal configuration for maximizing power efficiency of DLRM-RMC1 and DLRM-RMC3 varies based on the tail latency target. As a result, Figure 12(b) shows that in order to maximize infrastructure efficiency, it is important to consider model architecture and tail latency targets.

D. Datacenter provisioning implications.

In addition to the scheduling optimizations offered by DeepRecSched, the analysis can be applied to study provisioning strategies for datacenters running the wide collection of recommendation models. Figure 16 considers the ratio of CPU to GPUs, in order to minimize total power consumption, as we vary tail latency targets (left) and GPU power efficiency (right). Here, all models serve an equal amount of traffic (QPS); the tradeoffs will vary based on the distribution of

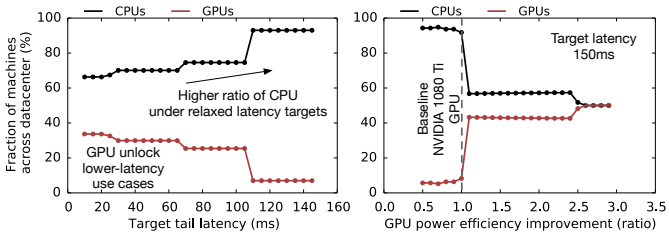

Fig. 16: Evaluating datacenter provisioning strategies for recommendation in order optimize overall power footprint. (Left) Increasing tail-latency reduces the fraction of GPUs. (Right) Improving GPU TDP, such as with more power efficient accelerators, increases the fraction of GPUs.

models deployed. Figure 16 shows higher ratios of GPUs are optimal under lower latency targets. Intuitively, this follows Figure 15 as GPUs enable lower latency recommendation use cases by accelerating the large queries. However, under more relaxed tail latency (i.e., SLA) targets it is optimal deploy higher ratios of CPUs for recommendation inference. Note, tail latency targets vary across applications as shown in Table II.

In addition to the impact of varying SLA targets, accelerator power efficiency also impacts datacenter provisioning. Figure 16 (right) considers the impact of varying the power efficiency of the NVIDIA GTX 1080 Ti GPU. Intuitively, improving power efficiency makes accelerators more appealing for recommendation inference. Thus, designing efficient GPUs and accelerators may enable specialized hardware for recommendation inference at the datacenter scale.

VII. Related Work

While the system and computer architecture community has devoted significant efforts to characterize and optimize deep neural network (DNN) inference efficiency, relatively little work has explored running recommendation at-scale.

DNN accelerator designs. Currently-available benchmarks for DNNs primarily focus on FC, CNNs, and RNNs [35], [45]–[47]. Building upon the performance bottlenecks, a variety of software and hardware solutions have been proposed to optimize traditional DNNs [13]–[15], [17], [48]–[68]. While the benchmarks and accelerator designs consider a variety of DNN use cases and systems, prior solutions do not apply to the wide collection of state-of-the-art recommendation models presented in this paper. For example, recent characterization of Facebook's DLRM implementation demonstrates that DNNs for recommendation have unique compute and memory characteristics [3], [10]. These implementations are included, as DLRM-RMC 1-3, within DeepRecInfra. In addition, MLPerf, an industry-academic benchmark suite for machine learning, provides NCF as a training benchmark [28]. (MLPerf is developing a more representative recommendation benchmark for the next submission round [69], [70]). In addition, an important aspect of the end-to-end infrastructure presented in the paper is accounting for at-scale request characteristics (arrival rate and size), particularly important for recommendation.

Optimizations for personalized recommendation. A few recent works explore design optimization opportunities for recommendation models. For instance, TensorDimm proposes a near memory processing solution for recommendation models similar to DLRM-RMC 1-3 and NCF [18]. Ginart et al. and Shi et al. [71], [72] compresses embedding tables in recommendation models while maintaining the model accuracy. In contrast, this paper optimizes the at-scale inference performance of a wider collection of recommendation models by considering the effect of inference query characteristics.

Machine learning at-scale. Finally, prior work has examined the performance characteristics and optimizations for ML running on at-scale, warehouse scale machines. Sirius and DjiNN-and-Tonic explore the implications of ML in warehouse-scale computers [26], [73]. However, the unique properties of recommendation inference and query patterns have not been the focus of the prior work. Li et al. [22] exploit task and data-level parallelism to meet SLA targets of latency critical applications i.e., Microsoft's Bing search. Furthermore, recent work has open-sourced benchmarks for studying the performance implication of at-scale execution of latency critical datacenter workloads and cloud micro-services [27], [30]. This paper provides an end-to-end infrastructure (DeepRecInfra) and design solutions (DeepRecSched) specialized for recommendation inference. DeepRecInfra models real-time query patterns, representative of the distinct working set size distribution. The unique characteristics provide significant performance improvement for at-scale recommendation.

VIII. Conclusion

Given the growing ubiquity of web-based services that use recommendation algorithms, such as search, social-media, e-commerce, and video streaming, neural personalized recommendation comprises the majority of AI inference capacity and cycles in production datacenter. We propose DeepRecInfra, an extensible infrastructure to study at-scale recommendation inference comprising eight state-of-the-art recommendation models, SLA targets, and query patterns. Built upon this framework, DeepRecSched exploits the unique characteristics of at-scale recommendation inference in order to optimize system throughput, under strict tail latency targets, by 2×. In a real production datacenter, DeepRecSched achieves similar performance benefits. Finally, through judicious optimizations, DeepRecSched can leverage additional parallelism by offloading queries across CPUs and specialized AI hardware to achieve higher system throughput and infrastructure efficiency.

IX. Acknowledgements

We would like to thank Cong Chen and Ashish Shenoy for the valuable feedback and numerous discussions on the at-scale execution of personalized recommendation systems in Facebook's datacenter fleets. The collaboration led to insights which we used to refine the proposed design presented in this paper. It also resulted in design implementation, testing, and evaluation of the proposed idea for production use cases. This work is also supported in part by NSF XPS-1533737 and an NSF Graduate Research Fellowship.

REFERENCES

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pp. 285–295, ACM, 2001.

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

[3] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019.

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

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

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

[7] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and E. Chi, "Recommending what video to watch next: A multitask ranking system," in *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, (New York, NY, USA), pp. 43–51, ACM, 2019.

[8] C. Underwood, "Use cases of recommendation systems in business – current applications and methods," 2019.

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

[10] U. Gupta, X. Wang, M. Naumov, C.-J. Wu, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, B. Jia, H.-H. S. Lee, *et al.*, "The architectural implications of facebook's dnn-based personalized recommendation," *arXiv preprint arXiv:1906.03109*, 2019.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, IEEE, 2017.

[12] M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung, P. Nel, and S. Malhotra, "Notes from the ai frontier insights from hundreds of use cases," 2018.

[13] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," in *ISCA*, 2016.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," *CoRR*, vol. abs/1602.01528, 2016.

[15] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[16] V. Volkov and J. W. Demmel, "Benchmarking gpus to tune dense linear algebra," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, 2008.

[17] Udit Gupta, Brandon Reagen, Lillian Pentecost, Marco Donato, Thierry Tambe, Alexander M. Rush, Gu-Yeon Wei, David Brooks, "MASR: A modular accelerator for sparse rnns," in *International Conference on Parallel Architectures and Compilation Techniques*, 2019.

[18] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 740–753, ACM, 2019.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[20] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[21] C.-J. Wu, R. Burke, E. H. Chi, J. Konstan, J. McAuley, Y. Raimond, and H. Zhang, "Developing a recommendation benchmark for mlperf training and inference," 2020.

[22] J. Li, K. Agrawal, S. Elnikety, Y. He, I. Lee, C. Lu, K. S. McKinley, *et al.*, "Work stealing for interactive services to meet target latency," in *ACM SIGPLAN Notices*, vol. 51, p. 14, ACM, 2016.

[23] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, pp. 30–37, Aug. 2009.

[24] "Netflix update: Try this at home." https://sifter.org/~simon/journal/20061211.html.

[25] J. Baxter, "A bayesian/information theoretic model of learning to learn via multiple task sampling," *Machine Learning*, vol. 28, pp. 7–39, Jul 1997.

[26] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, *et al.*, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *ACM SIGPLAN Notices*, vol. 50, pp. 223–238, ACM, 2015.

[27] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pp. 3–18, ACM, 2019.

[28] "A broad ml benchmark suite for measuring performance of ml software frameworks, ml hardware accelerators, and ml cloud platforms." https://mlperf.org/, 2019.

[29] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 202–208, ACM, 2009.

[30] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–10, IEEE, 2016.

[31] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, (New York, NY, USA), p. 248–259, Association for Computing Machinery, 2004.

[32] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Divergence-aware warp scheduling," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 99–110, 2013.

[33] A. Jaleel, J. Nuzman, A. Moga, S. C. Steely, and J. Emer, "High performing cache hierarchies for server workloads: Relaxing inclusion to capture the latency benefits of exclusive caches," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 343–353, IEEE, 2015.

[34] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr, and J. Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 151–162, IEEE Computer Society, 2010.

[35] E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.

[36] "Intel math kernel library." https://software.intel.com/en-us/mkl, 2018.

[37] "Nvidia cuda deep neural network library." https://developer.nvidia.com/cudnn, 2019.

[38] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, (Washington, DC, USA), pp. 119–130, IEEE Computer Society, 2012.

[39] C. Gregg and K. Hazelwood, "Where is the data? why you cannot debate cpu vs. gpu performance without the answer," in *(IEEE ISPASS)*

IEEE International Symposium on Performance Analysis of Systems and Software, pp. 134–144, 2011.

[40] D. Lustig and M. Martonosi, "Reducing gpu offload latency via fine-grained cpu-gpu synchronization," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 354–365, 2013.

[41] S. Pai, M. J. Thazhuthaveetil, and R. Govindarajan, "Improving gpgpu concurrency with elastic kernels," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, (New York, NY, USA), p. 407–418, Association for Computing Machinery, 2013.

[42] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. Mccardwell, A. Villegas, and D. Kaeli, "Hetero-mark, a benchmark suite for cpu-gpu collaborative computing," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–10, 2016.

[43] S.-Y. Lee and C.-J. Wu, "Performance characterization, prediction, and optimization for heterogeneous systems with multi-level memory interference," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 43–53, 2017.

[44] M. E. Belviranli, F. Khorasani, L. N. Bhuyan, and R. Gupta, "Cumas: Data transfer aware multi-application scheduling for shared gpus," in *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[45] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks, "Fathom: Reference workloads for modern deep learning methods," IISWC'16, 2016.

[46] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pp. 88–100, IEEE, 2018.

[47] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition,"

[48] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, "SCNN: an accelerator for compressed-sparse convolutional neural networks," *CoRR*, vol. abs/1708.04485, 2017.

[49] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzàlez, "E-pur: An energy-efficient processing unit for recurrent neural networks," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, PACT '18, pp. 18:1–18:12, ACM, 2018.

[50] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pp. 319–333, ACM, 2019.

[51] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible acceleration for 3d cnn-based video understanding," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 933–946, Oct 2018.

[52] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Teman, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[53] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, Oct 2016.

[54] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration,"

[55] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 18, IEEE Press, 2016.

[56] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, 2014.

[57] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 92–104, ACM, 2015.

[58] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 369–381, ACM, 2015.

[59] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[60] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[61] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 380–392, IEEE, 2016.

[62] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 255–266, IEEE Press, 2016.

[63] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "Tabla: A unified template-based framework for accelerating statistical machine learning," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 14–26, IEEE, 2016.

[64] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 751–764, ACM, 2017.

[65] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G.-Y. Wei, and D. Brooks, "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*, MICRO '52, (New York, NY, USA), pp. 769–781, ACM, 2019.

[66] Y. Kwon and M. Rhu, "Beyond the memory wall: A case for memory-centric hpc system for deep learning," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 148–161, IEEE, 2018.

[67] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," *arXiv preprint arXiv:1909.04548*, 2019.

[68] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 382–394, ACM, 2017.

[69] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, C.-J. Wu, L. Xu, C. Young, and M. Zaharia, "Mlperf training benchmark," *arXiv preprint arXiv:1910.01500*, 2019.

[70] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *arXiv preprint arXiv:1911.02549*, 2019.

[71] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," *arXiv preprint arXiv:1909.11810*, 2019.

[72] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, "Compositional embeddings using complementary partitions for memory-efficient recommendation systems," *arXiv preprint arXiv:1909.02107*, 2019.

[73] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and tonic: DNN as a service and its implications for future warehouse scale computers," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–40, June 2015.