MP-Rec: Hardware-Software Co-Design to Enable Multi-Path Recommendation

Samuel Hsia[1,2], Udit Gupta[1], Bilge Acun[1], Newsha Ardalani[1], Pan Zhong[1],
Gu-Yeon Wei[2], David Brooks[1,2], Carole-Jean Wu[1]

[1]Meta AI, [2]Harvard University

shsia@g.harvard.edu, carolejeanwu@meta.com

Abstract

Deep learning recommendation systems serve personalized content under diverse tail-latency targets and input-query loads. In order to do so, state-of-the-art recommendation models rely on terabyte-scale embedding tables to learn user preferences over large bodies of contents. The reliance on a fixed embedding representation of embedding tables not only imposes significant memory capacity and bandwidth requirements but also limits the scope of compatible system solutions. This paper challenges the assumption of fixed embedding representations by showing how synergies between embedding representations and hardware platforms can lead to improvements in both algorithmic- and system performance. Based on our characterization of various embedding representations, we propose a hybrid embedding representation that achieves higher quality embeddings at the cost of increased memory and compute requirements. To address the system performance challenges of the hybrid representation, we propose MP-Rec — a co-design technique that exploits heterogeneity and dynamic selection of embedding representations and underlying hardware platforms.

On real system hardware, we demonstrate how matching custom accelerators, i.e., GPUs, TPUs, and IPUs, with compatible embedding representations can lead to $16.65\times$ performance speedup. Additionally, in query-serving scenarios, MP-Rec achieves $2.49\times$ and $3.76\times$ higher correct prediction throughput and 0.19% and 0.22% better model quality on a CPU-GPU system for the Kaggle and Terabyte datasets, respectively.

1. Introduction

Deep learning (DL) recommendation models support a wide variety of applications, such as search [4, 6, 27, 60], social media [1, 15, 17, 53], e-commerce [61, 62], and entertainment [18]. Due to its overarching impact, neural recommendation has become a dominant source of compute cycles in large-scale AI infrastructures. In 2019, recommendation use cases contributed to 79% of the overall AI inference cycles at Meta, making it one of the most resource-demanding DL use cases at the data center scale [15, 17].
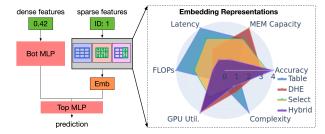

Figure 1: Compared to prior work, MP-Rec explores the embedding access design space through different *embedding representations*.

A critical component of state-of-the-art recommendation models is the embedding table [6, 18, 36, 38, 54, 60, 61, 62]. Information, such as user preferences and content understanding, is represented as individual vectors within these embedding tables. To support increasingly complex applications and user preference models, embedding table sizes have grown super-linearly into the terabyte-scale [51]. As a result, a plethora of system- and hardware-level solutions for neural recommendation have focused on addressing the memory capacity and bandwidth challenges of large-scale embedding tables [30, 34, 49, 24, 36, 3, 7, 33].

However, there is additional room for algorithmic and system performance improvements if we go beyond exclusively using tables as *embedding representation*. Recent proposals examine alternative embedding representations use GEMM-heavy compute stacks to dynamically generate embedding vectors [54, 29]. While these representations significantly reduce memory capacity requirements, the techniques introduce orders of magnitude higher FLOPs.

Based on the detailed design space characterization for embedding representations (Figure 1; Section 3), we identify significant performance improvement potential when utilizing custom accelerators for compatible representations. To demonstrate the impact of representation-hardware compatibility, we perform real-system evaluations on a wide range of hardware, including CPUs and GPUs, as well as custom AI accelerators such as Google Tensor Processing Units (TPUs) [25, 26, 27] and Graph-

core Intelligence Processing Units (IPUs) [10, 22] at core-, chip-, board, and pod-level configurations, demonstrating up to $16.65\times$ performance speedup. Ultimately, *there is no one-size-fits-all static solution* as representation requirements and hardware capabilities vary.

In this work, we propose a new *hybrid* embedding representation for neural recommendation tasks. While prior representations focus exclusively on either memory- or compute-based execution paths, the *hybrid* representation leverages these contrasting execution patterns to increase learning capacity and produce higher quality embeddings. Our evaluation results show that the *hybrid* representation increases model quality by 0.19%, 0.22%, and 0.014% on the open-source Kaggle [28], Terabyte [46], and internal use-cases, respectively. Due to its increased complexity, the *hybrid* representation requires even higher capacity and memory requirements.

Taking a step further, to support the resource-intensive *hybrid* representation and dynamic mapping of heterogeneous accelerators and representations, we propose Multi-Path Recommendation (MP-Rec) — a dynamic representation-hardware co-design technique to maximize throughput of correct recommendations while meeting tail latency requirements. Depending on memory capacities of AI inference systems, MP-Rec first generates accuracy-optimal representation-hardware mappings based on the unique properties of different embedding representations (Figure 1 (right)), forming multiple potential embedding execution paths. At runtime, depending on input query sizes and application-specific performance targets, MP-Rec activates embedding path(s) by scheduling queries onto available representation-hardware configurations to jointly maximize prediction quality and throughput. To further speed up MP-Rec, we introduce MP-Cache to exploit novel caching opportunities introduced by the intermediate results of the new embedding representations. MP-Cache targets both data locality and value similarity of embedding accesses to make computationally-expensive representations viable.

We evaluate the performance of MP-Rec on a real CPU-GPU platform, demonstrating $2.49\times$ and $3.76\times$ higher throughput of correct predictions on Kaggle [28] and Terabyte [46], respectively, over the CPU baseline. In addition, when we integrate IPUs into MP-Rec for query serving, we observe a significant throughput improvement potential of $34.24\times$ that can be unlocked with future software support. For the constant throughput scenarios at strict SLA latency targets, MP-Rec reduces SLA latency target violations by 27.59% compared to exclusively using the baseline embedding table-based recommendation models on CPUs. Overall, MP-Rec showcases the massive potential of algorithmic- and system-performance improvements when we integrate embedding representation design into the system design space for neural recommen-

dation. Finally, these performance improvements can be further enhanced by custom AI accelerators that benefit from representation-level synergies.

The main contributions of this work are as follows:

- We propose a new *hybrid* embedding representation that increases learning capacity and produces higher quality embeddings at the cost of increased compute and memory requirements. The *hybrid* embedding representation demonstrates measurable improvements in model quality.
- We implement and characterize different embedding representations using state-of-the-art custom AI accelerators (i.e., TPUs and IPUs) at core-, chip-, board-, and pod-level configurations. We identify key system challenges of adapting specific representations to accelerators, highlighting distinct accelerator-specific advantages: TPUs for embedding tables, IPUs for compact compute-stacks, and GPUs for energy-efficient execution of large-capacity models (Section 3.4).
- We propose MP-Rec — a dynamic representation-hardware co-design technique for deep learning recommendation inference. MP-Rec mitigates the performance and accuracy degradations from static representation-hardware mappings. We augment MP-Rec with a two-tier cache design (MP-Cache) to exploit unique caching opportunities found in compute-based representations.

2. Background: Embedding Representations

Embeddings are a performance-critical component of neural recommendation models. In order to be processed by recommendation models, sparse feature IDs must first be transformed into dense embedding vectors. This transformation process – *embedding access* – can be realized through different embedding representations.

We start with two distinct classes of embedding representations: *storage* and *generation*. While storing learned embedding vectors as tables is a widely adopted approach, it introduces significant memory capacity and bandwidth requirements [11, 13, 19, 30, 34, 36, 49] (Section 2.1). On the other hand, generating embeddings with compute-heavy encoder-decoder stacks trades off these memory system requirements with compute demand (Section 2.2). To leverage these contrasting qualities, we introduce new embedding representations – *select* and *hybrid* – that leverage complementary system resources to generate embeddings from table and DHE representations (Section 2.3).
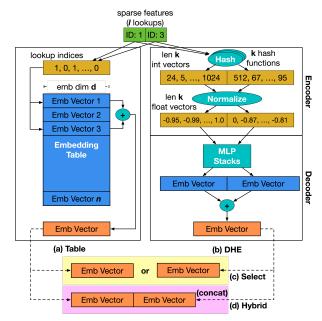
2

Figure 2: (a) *Table* **representation stores learned embeddings while (b)** *DHE* **dynamically generates embeddings via encoder-decoder stacks. We introduce (c)** *select* **representation that selects either (a) or (b) at table-level granularity and (d)** *hybrid* **representation that leverages both (a) and (b) for highly accurate embeddings.**

2.1. Storing Embeddings: Embedding Tables

 Figure 2 (a) depicts the embedding access mechanism of a typical neural recommendation model. Sparse feature lookup IDs are converted into multi-hot encoded vectors, which are then used as indices into embedding tables. While the table representation is used by many state-of-the-art neural recommender systems for its simplicity and relatively high accuracy, it has significant memory bandwidth and capacity requirements.

2.2. Generating Embeddings: DHE

 Alternatively, embedding vectors can be dynamically generated from compute stacks. Examples include Tensor Train Compression (TT-Rec) [54] and Deep Hash Embedding (DHE) [29]. In this work, we focus on DHE (Figure 2(b)) over TT-Rec due to the flexibility in tuning DHE's encoder-decoder stacks. DHE execution is separated into two phases: the encoder stack generates intermediate values from sparse ID inputs (upper block) and the decoder stack generates dense embedding vectors from the intermediate values (lower block). More specifically, the functionalities of the encoder and decoder stacks are as follows:

- **(Encoder Stack):** First apply k parallel, unique encoder hash functions on input sparse IDs. Instead of being used as lookup indices, these updated IDs are then applied with normalization functions to create in-

termediate dense features.
- **(Decoder Stack):** The intermediate dense features then pass through decoder MLP stacks to generate final embedding vectors for downstream stacks.

While embedding tables and DHE both produce dense embedding vectors from sparse IDs, the required algorithmic steps and compute resources are fundamentally different. For embedding tables, individual embedding vectors are stored after training and accessed during inference.

 To learn valuable correlations from the ever-increasing data volume, the number of entries per embedding table have bloated to millions, leading to aggregate memory capacity requirements in the terabytes [36, 37]. In DHE, encoder-decoder stacks are first trained offline. During inference, embedding vectors are dynamically generated running sparse IDs through trained DHE stacks. Encoder hash functions and decoder MLPs contribute higher FLOPs, shifting the system bottleneck from memory capacity to computation.

2.3. Novel Representations: Select & Hybrid

 In Figure 2(c), we present the proposed *select embedding representation*, where we select either embedding table or DHE representation at the feature-level (i.e., table-level) granularity. With the *select* embedding representation, recommendation model designers can make the aforementioned memory-compute tradeoffs for each sparse feature. In Figure 2(d), we capitalize upon the dichotomy of embedding tables and DHE by proposing a *table-compute hybrid representation*. In this proposed *hybrid* representation, sparse IDs are used to *both* access embedding tables and dynamically generate embedding vectors. The resulting embeddings from both mechanisms are then concatenated. The embedding tables and decoder MLP stacks are trained together.

 Our newly proposed *hybrid* representation is based on two key observations. First, embeddings learned from tables and DHE have different semantics. Generated embeddings from DHE can achieve higher model quality for some CTR predictions tasks [28, 42, 46], as demonstrated in Section 3.1. Second, embedding tables and DHE compute stacks stress independent system resources. The *hybrid* representation is unique in its ability to fully utilize both memory and compute resources of an underlying system for higher recommendation quality.

3. Design Space Exploration for Sparse Feature Representation

 In this section, we characterize embedding table, DHE, *select*, and *hybrid* representations across the important dimensions of model accuracy (Section 3.1), model capacity (Section 3.2), execution latency (Section 3.3), and accelerator compatibility (Section 3.4). We show that this

previously unexplored design space offers not only noticeable accuracy improvements but also hardware-specific optimization opportunities.

Figure 3 provides an overview for the design space trade-offs of the four embedding representations along model accuracy (y-axis), capacity, and FLOPs (x-axis for (a) and (b), respectively).

3.1. Accuracy: Tuning DHE Parameters

Hybrid representation (violet points) configurations, which use both table and compute stacks, achieve the highest accuracies. Figure 3 illustrates that the most accurate *hybrid* representation configurations achieve 0.19% and 0.22% accuracy improvements over the embedding table baselines on Kaggle and Terabyte, respectively. Note that for many recommendation use cases, accuracy improvements > 0.1% are considered significant. [4, 48].

We tune the hyperparameters of a DHE stack to realize these improved accuracies. Each point in Figure 3 corresponds to a model with unique hyperparameters. For embedding tables, we vary embedding dimension. For DHE stacks, we vary the number of parallel encoder hash functions k, the decoder MLP width d_{NN}, and decoder MLP height h. We also vary the shape of FC layers for each decoder MLP stack.

Figure 4 depicts the compression ratio – relative to a 12.59 GB embedding table baseline model – (x-axis) and model accuracy (y-axis) for different DHE configurations. The color of each point denotes the number of hash functions used (k). We see that, as k increases from 2 to 2048 (i.e., color progression from red to black), model accuracy increases. Thus, k is an important factor in determining achievable model accuracy. In contrast, for a given k, varying the decoder MLP size and shape had a relatively insignifcant effect on model accuracy. This can be observed from how points with the same color (i.e., same k different (d_{NN}, h)) have relatively similar model accuracies. A similar trend is observed for *select* and *hybrid* representations as well.

3.2. Capacity: Enabling Memory-Constrained Neural Recommendation

In Figure 3 (a), we observe that DHE configurations (red points) have model capacities $10 \sim 1000\times$ smaller than gigabyte-scale baseline embedding table configurations (blue points). Through DHE, we are able to construct a recommendation model that is $334\times$ smaller in model capacity than the MLPerf baseline which uses embedding tables – without any accuracy degradation (i.e., horizontal dotted MLPerf accuracy baseline) [42, 46]. DHE accomplishes this by having a shared set of encoder-decoder parameters for generating embeddings as opposed to storing user- and item-specific embeddings. With these compression ratios, recommendation models can be compressed by orders of magnitude, from GBs to MBs, and deployed on a wider range of hardware platforms and use-cases.

3.3. Latency: Operator Breakdowns

While compute-based representations, such as DHE and *hybrid*, can improve model accuracies, the encoder-decoder stacks introduce FLOPs that lead to execution slowdowns. Figure 3 (b) shows how models that use DHE and *hybrid* have $10 \sim 100\times$ more FLOPs than those relying on embedding tables.

Figure 5 shows operator breakdown of different representations for CPUs and GPUs. For DHE, we see $10.5\times$ and $4.7\times$ slowdown on CPUs and GPUs, respectively. DHE suffers less slowdown on GPUs because its encoder stack is composed of parallel hashing of k encoder hash functions. When $k \sim O(1000)$, GPU outshines CPU for such massively parallel operations. For *select*, we see only $2.1\times$ and $1.5\times$ slowdown on CPUs and GPUs, respectively. For this *select* representation, only the 3 largest embedding tables are replaced with DHE stacks. The rest of the sparse features use table representation, leading to faster execution. For *hybrid*, we observe $11.2\times$ and $5.4\times$ slowdown on CPUs and GPUs, respectively. *Hybrid* results in longest latencies because both embedding tables and DHE stacks are executed to generate highly accurate embedding vectors.

3.4. Accelerator Compatibility: IPU, TPU Evaluation

Next, we explore accelerator-level synergies using real modern AI accelerators: Graphcore IPUs and Google TPUs [10, 22, 25, 26, 27].

CPU and GPU system specifications are detailed in Section 5.1. We evaluate TPUv3 at core-, chip-, and board-level configurations. For chip- and board-level configurations, we employ data-parallelism for increased throughput. We evaluate Graphcore GC200 IPU at chip-, board-, and pod-level configurations (Figure 6). For a single IPU chip, table and *hybrid* configurations require backup Streaming Memory (i.e., DRAM) since the model does not fit within the 900 MB on-chip SRAM. For an IPU-M2000 board, we pipeline the model across the SRAM of the four chips. For IPU-POD16, we employ data-parallelism.

Figure 7 quantifies the design space trade-offs with four key observations:

O1: For embedding tables, TPUs achieve highest speedup due to their custom `TPUEmbedding` layers (Figure 7 (a)). Each TPU core has access to 16 GB of HBM. TPUs utilize this HBM efficiently by: 1) sharding larger tables and replicating smaller tables across TensorCore HBMs and 2) pipelining embedding lookups with TensorCore computations. These optimizations form the basis of TPU custom `TPUEmbedding` layers.
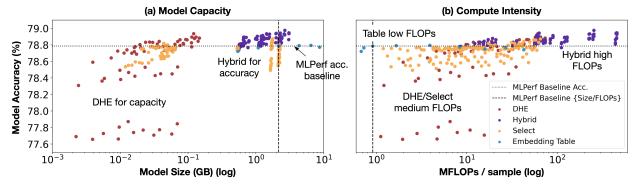
Figure 3: *DHE* representation saves memory capacity, *hybrid* representation enables optimal accuracies, and *table* representation has faster latency from less FLOPs. Evaluation is on the Criteo Kaggle data set.
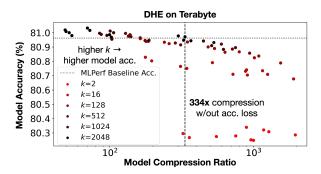

Figure 4: DHE compute stacks can be tuned to improve either model *accuracy* or *compression* ratio.
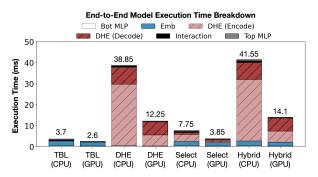

Figure 5: *Operator breakdown* of Table, DHE, *select*, and *hybrid* execution on CPUs and GPUs. *Hybrid* shows worst performance in terms of latency while *select* offers a compromise between Table and DHE.

O2: For DHE stacks, IPUs perform well primarily when all model parameters and inputs fit into the IPU scratchpad memory (Figure 7 (b)). Each Graphcore IPU has access to 900 MB of scratchpad SRAM. While this SRAM has no default caching abilities, when all model parameters and activations fit within this SRAM budget, IPUs rely on virtually only compute and on-chip memory accesses, leading to significant speedups. Furthermore, the lack of off-chip DRAM access also contributes to high energy efficiency for DHE use cases. IPU-16 achieves 16.65× performance speedup over embedding table execution on CPUs.
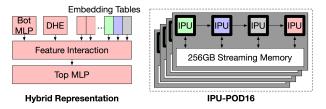

Figure 6: *Hybrid* representation deployment strategies for pod-scale IPUs. Single chip execution requires offloading three largest embedding tables (green, blue, and gray shade) to DRAM while pod-level execution duplicates board-level parallelism strategy four times for data parallelism.

O3: GPUs offer higher energy efficiency for large embedding table-based models (Figure 7 (bottom). For large embedding table execution, GPU is most energy-efficient. This is because while TPU shows 3.12×, 11.13× performance speedup for its chip- and board-level configurations, its single chip TDP is 1.8× higher than that of V100's. Additionally, the V100 is also more energy efficient than IPU for this specific use-case. This is because a single IPU chip's SRAM scratchpad cannot hold the entire model, leading to frequent off-chip DRAM access.

O4: No single hardware platform is optimal for all representations and optimization objectives, motivating the need for a dynamic representation-hardware co-design solution. Figure 7 shows that there is no one size fits all hardware solution across all possible embedding representations. While TPUs accelerate embedding lookups well, when the models are small enough to fit on-chip, IPUs perform better because of more efficient on-chip memory accesses. On the other hand, GPUs offer a competitive option from per-chip and ease-of-use standpoints (both TPUs and IPUs require lengthy one-time compilations).

The in-depth characterization based on real AI systems demonstrates the potential for performance improvement by considering heterogeneous representations and accelerators. Thus, to best exploit these algorithmic and system
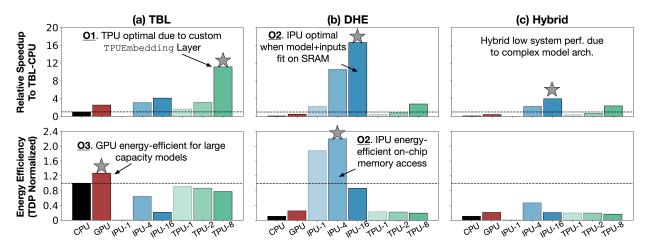
Figure 7: *Table, DHE, and hybrid embedding representations* evaluated across different custom accelerators. TPUv3s see speedups from TPU-optimized `TPUEmbeddings` while Graphcore IPUs offer optimal performance when the model and activations fit within its 900 MB SRAM per-chip scratchpad. CPU: Broadwell Xeon; GPU: V100; IPU-1: 1-chip GC200; TPU-1: 1-core TPUv3 (TPU has 2 cores/chip).
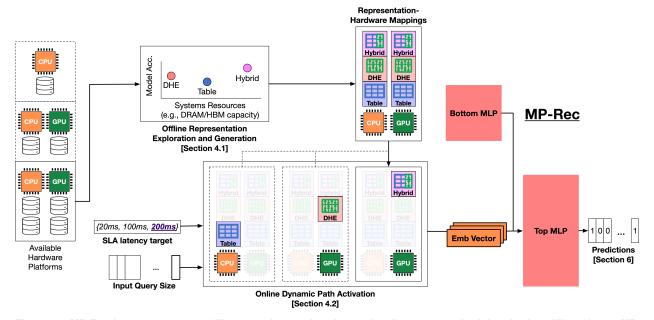

Figure 8: *MP-Rec* has two stages: offline mapping exploration and online query scheduler. In the offline phase, MP-Rec considers algorithmic and systems-level exploration constraints to generate optimal representation-hardware configurations. In the online phase, MP-Rec dynamically schedules queries onto available representation-HW execution paths based on query-level information to maximize for amount of high quality recommendation.

level trade-offs, representation and hardware pairing must be a dynamic rather than static decision.

4. MP-Rec: Representation-Hardware Co-Design

Built upon the real system characterization results, we propose a **M**ulti-**P**ath embedding representation co-design technique for **Rec**ommendation inference, **MP-Rec**. MP-Rec maximizes throughput of correct predictions in two stages:

Offline Stage [Section 4.1]. MP-Rec determines *which embedding representation(s)* will be used and their corresponding *hardware mapping strategies* (Algorithm 1). Embedding representation and mapping decisions are based on system memory capacities.

Online Stage [Section 4.2]. MP-Rec considers service-level agreements (SLA), such as model accuracy and tail latency targets, of the application and runtime factors, such as input query sizes (Algorithm 2). MP-Rec produces embedding vectors from sparse features
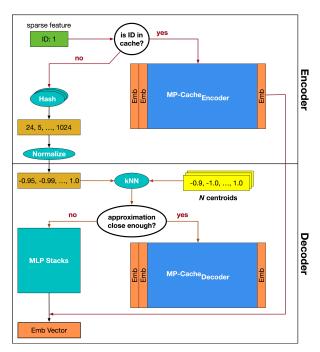
6

Figure 9: *MP-Cache* **is comprised of two cascading stages. MP-Cache$_{encoder}$ exploits access frequency of sparse IDs while MP-Cache$_{decoder}$ exploits value similarity between intermediate results.**

by dynamically activating either Table, DHE, or *hybrid* execution paths on available hardware platforms.

To accelerate the encoder-decoder stack in the DHE and *hybrid* paths, we introduce **MP-Cache$_{encoder}$** for the encoder stack and **MP-Cache$_{decoder}$** for the decoder stack. MP-Cache$_{encoder}$ exploits the power law distribution of embedding access frequencies [13], whereas MP-Cache$_{decoder}$ considers value similarity of intermediate encoder stack results. Figures 8 and 9 provide the design overviews for MP-Rec and MP-Cache, respectively. Next, we present the major components of MP-Rec in detail.

4.1. Offline HW-Specific Representation Generation

MP-Rec factors in representation-level insights and exploration constraints to generate representation-hardware configurations (Algorithm 1).

Heterogeneous Hardware Platforms. MP-Rec considers available hardware platforms and their memory capacities for embedding access. Traditionally, neural recommender systems are deployed exclusively on server-class CPUs and GPUs due to their large memory requirements [15, 17, 37]. However, with DHE's potential for compression, MP-Rec is able to target a wider range of hardware platforms. For MP-Rec, we consider each individual hardware component by their memory capacity budget and map representation(s) accordingly to maximize memory capacity utilization.

Representation Exploration. Algorithm 1 details the

Algorithm 1 MP-Rec Offline Stage
___
Input: Embedding Representation Space $R = \{r_i\}$, Hardware Platforms $H = \{h_i\}$
Output: Optimal representation-hardware mapping strategies $S^\star \in \{(r_i, h_i)\} \,|\, r_i^\star$ is accuracy optimal.
1: $S^\star \leftarrow \{\}$
2: **for all** hardware $h_i \in H$ **do**
3:     **if** $\exists r_{j,hybrid}^\star$ that fits on h_i **then**
4:         $S^\star \cup (r_{j,hybrid}^\star, h_i)$
5:     **end if**
6:     **if** $\exists r_{j,table}$ that *still* fits on h_i **then**
7:         $S^\star \cup (r_{j,table}, h_i)$
8:     **end if**
9:     **if** $\exists r_{j,DHE}^\star$ that *still* fits on **then**
10:         $S^\star \cup (r_{j,DHE}^\star, h_i)$
11:     **end if**
12:     **if** h_i has $\leq$ one r_j mapping in $S^\star$ **then**
13:         $S^\star \cup (r_{j,DHE(compact)}, h_i)$
14:     **end if**
15: **end for**
16: train all representations r_i found within $S^\star$
___

steps for finding optimal representation-hardware mappings $S^\star$. For each hardware component h_i, we first see if there exists a *hybrid* embedding representation $r_{j,hybrid}^\star$ that is 1) under capacity budget, 2) has large # of encoder hash functions k and 3) has a decoder MLP (i.e., d_{NN}, h) as small as reasonably possible. As discussed in Section 3, we want high k for better model accuracy and small decoder MLP to minimize memory footprint and FLOPs.

With a *hybrid* representation that provides high accuracy, MP-Rec then searches for an embedding table representation $r_{j,table}$ that can be later activated to handle latency-critical situations (i.e., tight SLA latency targets). If there is still capacity available on h_i, we search for a DHE representation $r_{j,DHE}^\star$ that has accuracy-latency trade-offs in-between the *hybrid* and table configurations. We then repeat this process for all available hardware platforms. On memory-constrained devices, we search for compact representation $r_{j,DHE(compact)}$. Selected representations are then profiled against the expected workload at different query sizes.

With a set of representations on each hardware platform, we can selectively activate mappings during the online stage to maximize recommendation quality while hitting SLA targets and maintaining high throughput to the best of our abilities.

4.2. Online Dynamic Multi-Path Activation

During online phase, MP-Rec dynamically activates representation-hardware execution paths to handle incoming queries based on query-level information (Algorithm 2). Currently, scheduling is dependent on query sizes and SLA latency targets.

Table 1: Systems Configurations.

Machines	Intel Broadwell CPU	NVIDIA V100 GPU	Graphcore IPU-M2000 (4 IPUs)	Graphcore IPU-POD16 (16 IPUs)
Frequency	2.2 GHz	1.2 GHz	1.35 GHz	1.35 GHz
Cores	12	5120	5888	23552
Cache Sizes	0.3-3-30 MB	3 MB	3.6 GB	14.4 GB
DRAM Capacity	264 GB	32 GB HBM2	256 GB	1024 GB
DRAM Bandwidth	76.8 GB/s	900 GB/s	20 GB/s	80 GB/s
TDP	105 W	250 W	600 W	2400 W

Algorithm 2 MP-Rec Online Stage

Input: Representation-hardware mappings $S \in \{(r_i, h_i)\}$, Input Query q

Output: Selected query execution path (r_i, h_i)

1: $n, t_{SLA} \leftarrow$ query size, SLA latency target
2: **if** $(r_{j,hybrid}, h_i)$ can process query size n under t_{SLA} **then**
3:      return $(r_{j,hybrid}, h_i)$
4: **else if** $(r_{j,DHE}, h_i)$ can process query size n under t_{SLA} **then**
5:      return $(r_{j,DHE}, h_i)$
6: **else**
7:      return $(r_{j,table}, h_i)$
8: **end if**

Varying Query Sizes and SLA Tail Latency Targets.
In real-world production environments, incoming queries arrive at different sizes and have to be served within application-specific SLA latency targets. Based on prior works, recommendation workloads can have query sizes between $1 - 4K$ and SLA latency targets from $1 - 100s$ ms [13].

Maximizing Throughput of Correct Predictions.
We activate representation execution paths based on incoming query size n and SLA latency target t_{SLA} (Algorithm 2). If there exists a *hybrid* configuration that can finish a query of size n within t_{SLA} without throughput degradation, that representation execution path is activated to achieve highest possible accuracy. If no *hybrid* execution path exists, we see if there is a DHE representation path for moderately improved accuracy. If n and t_{SLA} are too strict, MP-Rec then defaults to activating the Table representation path to satisfy SLA conditions. Ultimately, MP-Rec dynamically activates the path of highest recommendation quality while ensuring table-level system throughput for different SLA conditions and varying query sizes, thus maximizing the throughput of correct predictions.

4.3. MP-Cache: Mitigating Latency of the Compute-Stack Path

While DHE and *hybrid* paths offer accuracy and capacity improvements, activating either path comes with significant latency overheads. In large-scale inference serving experiments (Section 6), we see that these latency degradations lead to higher tail latencies. In order to close this performance gap, we devise **MP-Cache**, a two-part caching optimization that exploits both access frequency and value similarity of embedding accesses (Figure 9).

MP-Cache$_{encoder}$: Exploiting Access Frequency.
In recommendation workloads, the access counts of *power*power users and items make up a sizable portion of total accesses [50]. We exploit this observation by caching pre-computed embeddings of such hot, frequently accessed IDs in a cache within the encoder stage. If we encounter a hot ID, we can directly look up the ID's pre-calculated embedding vector and skip the entire encoder-decoder stack.

MP-Cache$_{decoder}$: Exploiting Value Similarity. If a sparse feature ID does not hit in MP-Cache$_{encoder}$, it goes through the encoder stack to generate an intermediate dense vector. This dense vector then becomes an input to the decoder MLP stack. As mentioned in Section 3, this decoder MLP stack can be costly in terms of latency. To mitigate this latency overhead, we propose MP-Cache$_{decoder}$ to exploit value similarity between intermediate dense vectors. We profile the intermediate dense vectors generated from a recommendation workload's sparse IDs and construct N centroids that best represent the overall distribution of possible intermediate vectors. With these centroids, our compute becomes k-nearest neighbors (kNN) search between the target intermediate dense vector and N centroids. In implementation, if the vectors are normalized, finding the nearest centroid can be simplified to parallelizable dot product followed by an argmax function, thus providing speedup over computation-heavy MLP stacks. The number of centroids N is an adjustable parameter: larger N gives better approximations at the cost of more compute.

5. Methodology

We present the experimental methodology used for evaluating MP-Rec on a design space spanning hardware platforms (Section 5.1), recommendation use cases (Section 5.2), inference runtime characteristics (Section 5.3), and evaluation metrics (Section 5.4).

8

5.1. Hardware Systems

One of MP-Rec's core features is its ability to generate optimal representation-hardware mappings for heterogeneous systems with different memory capacities. To demonstrate this flexibility, we evaluate MP-Rec at three different configurations:

1. **HW-1:** Single CPU-GPU node with 32 GB CPU DRAM and 32 GB GPU HBM2. Unless otherwise specified, we evaluate this configuration in Section 6.
2. **HW-2:** Resource-constrained case-study with 1 GB CPU DRAM and 200 MB GPU HBM2.
3. **HW-3:** Custom-accelerator case-study with 32 GB CPU DRAM and board-, pod-level IPU platforms.

CPU, GPU, and IPU performance data is collected on real commodity hardware platforms (Table 1). For query serving experiments involving IPU platforms (Section 6.3), we exclude model compilation overheads. We profile IPU platforms across all possible query configurations and use this profiled information to get estimated performance.

5.2. Datasets and Models

We evaluate MP-Rec with open source recommendation datasets Criteo Kaggle [28] and Terabyte [46] trained on Meta's Deep Learning Recommendation Model (DLRM) [38]. The MLPerf baseline model for Terabyte is $5.8\times$ larger than the baseline model for Kaggle (12.59 GB and 2.16 GB, respectively). For each of the embedding representations covered in Section 2, we replace the embedding tables of DLRM with our implementation of the target representation. The encoder-decoder stack implementation is based on [29] and in PyTorch [41]. Respective characterization baselines (i.e., accuracy, capacity, FLOPs) in Section 3 are from the default MLPerf DLRM-Kaggle and Terabyte configurations [42]. For IPU query serving use-cases (Section 6.3), we reduce the embedding dimension of the Terabyte model's tables to fit the model onto IPU-POD16 (Table 1).

5.3. Inference Runtime Characteristics

As shown in Figure 8, MP-Rec dynamically serves inference queries across different runtime conditions:

Query Sizes and Distribution. Representation-HW mappings perform differently based on query sizes. Unless otherwise specified, we evaluate a generated query set of size 10K, following a lognormal distribution with an average query size of 128 [13, 14].

SLA Latency Targets. Inference query requests have to be finished under application-specific SLA latency targets. For recommendation workloads, latency targets can range from $1-100s$ milliseconds [13]. We overview results for a strict SLA scenario of 10ms (found in e-commerce platforms such as [6, 27, 40, 61, 62]) then demonstrate MP-Rec benefits at targets up to 200ms.

High Throughput Inference. Inference engines ideally maintain high throughput. However, large query execution may lead to QPS degradations. Unless otherwise specified, we target 1000 QPS.

5.4. Evaluation Metrics

In addition to model quality in click-through rate prediction accuracy, and model capacity in bytes, we evaluate:

- $throughput_{correct\_predictions}$: **Throughput of Correct Predictions**. For production use-cases, we care about not only the quality of individual recommendations but also how efficiently the models can be served at-scale. We evaluate $\frac{Correct\,Samples}{Second}$ with:

$$\frac{Queries}{Second} \times \frac{Samples}{Query} \times \frac{Correct\,Samples}{Samples}$$
$$= QPS \times Query\,Size \times Model\,Accuracy$$

- **SLA Latency Violations.** Meeting SLA is crucial for recommendation use cases. Thus, we also evaluate the effectiveness of MP-Rec in reducing SLA violations.

6. Evaluation Results and Analysis

In this section, we show how MP-Rec improves upon various static representation-hardware deployment choices – in both throughput of correct predictions and accuracy – by dynamically switching between representations on heterogeneous hardware. We first evaluate MP-Rec – with MP-Cache enabled – on the HW-1 design point. Then, we cover production systems evaluation and consider resource-constrained (HW-2) and custom-accelerator (HW-3) case studies. Next, we perform sensitivity studies on both query size distributions and SLA latency targets and explore query-splitting across heterogeneous hardware as an additional optimization. After that, we explore MP-Rec's dynamic switching mechanism and MP-Cache's two-stage structure. Finally, we quantify how MP-Rec reduces SLA violations for constant throughput use-cases and present analytical scaling implications on large-scale training systems.

6.1. MP-Rec Performance Overview

MP-Rec achieves the highest model accuracy among all the embedding representations on both Kaggle and Terabyte datasets by using more accurate representations like DHE and *hybrid* (Section 6.2 – **Insight 1**). For Kaggle and Terabyte use-cases, MP-Rec conditionally improves achievable model accuracy by 0.19% and 0.22%, respectively (Table 2).

Despite their accuracy benefits, DHE and *hybrid* execution paths exhibit long latencies from their orders of magnitude higher FLOPs. Thus, statically deploying these compute-based representations on fixed-hardware platforms leads to throughput degradations. MP-Rec avoids
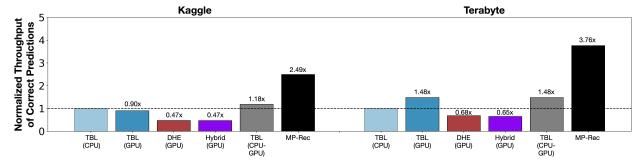
Figure 10: *Throughput of Correct Predictions* for serving 10K queries in *Kaggle* and *Terabyte* use-cases, respectively. With MP-Cache, MP-Rec improves upon throughput of correct predictions by activating high accuracy execution paths. Statically deploying DHE, *hybrid* representations leads to throughput degradations compared to executing embedding tables on CPUs and/or GPUs.

Table 2: *Achievable model accuracies* of optimal representation-hardware mappings for *Kaggle* and *Terabyte*, respectively. By using DHE or Hybrid, MP-Rec increases achievable model accuracies over table baselines.

	Table (Baseline)	DHE	Hybrid	MP-Rec
Kaggle	78.79%	78.94%	78.98%	78.98%
Terabyte	80.81%	80.99%	81.03%	81.03%

Table 3: *Memory footprints* for HW-1 on *Kaggle* and *Terabyte*, respectively. MP-Rec incurs greater memory footprint than static deployment choices since it stores multiple representations on each HW platform.

	Table (Baseline)	DHE	Hybrid	MP-Rec
Kaggle	2.16 GB	126 MB	2.29 GB	4.58 GB
Terabyte	12.58 GB	123 MB	12.70 GB	25.41 GB

these performance degradations by dynamically switching execution paths at the representation- and HW-level granularities (Section 6.2 – **Insights 2, 3**). Furthermore, MP-Cache reduces the latency of DHE and *hybrid* encoder-decoder stacks, making these representations more viable for activation (Section 6.2 – **Insight 4**). MP-Rec optimizes $throughput_{correct\_predictions}$ by using these factors to improve accuracy while maintaining performance. MP-Rec improves $throughput_{correct\_predictions}$ by 2.49× and 3.76× on Kaggle and Terabyte, respectively (Figure 10). We further break down improvements in $throughput_{correct\_predictions}$ in Figure 11.

To achieve these benefits, MP-Rec stores multiple representation execution paths on each hardware platform. This leads to increased memory footprint compared to statically using a single representation (Table 3). We demonstrate MP-Rec's ability to target memory-constrained and accelerator-based design points in Table 4 (**Insight 5**) and Figure 12 (**Insight 6**), respectively.

Production Use-Case Evaluation. We implement and evaluate MP-Rec using recommendation tasks in a production setting where our baseline is an internal table-based recommendation model. We either replace the embedding tables of this baseline model with DHE stacks or augment the tables for *hybrid* representations. First, we observe a noticeable model compression ratio when replacing embedding tables with DHE stacks. Next, *hybrid* configurations achieve 0.014% improvement in model

accuracy. Finally, utilizing DHE stacks introduces flops that incur a throughput degradation of 23.59%.

6.2. Evaluation Result Insights for MP-Rec

We begin by evaluating MP-Rec at **HW-1** (as specified in Section 5.1) on both the Kaggle and Terabyte use-cases. Evaluation is done on 10K queries with targets of 1000 QPS and 10ms SLA latency target. We highlight the key MP-Rec features from evaluation results:

Insight 1: MP-Rec improves achievable recommendation quality by including carefully-tuned DHE and *hybrid* execution paths. During its online phase, MP-Rec dynamically activates DHE and *hybrid* paths when there is no expected latency-bounded throughput degradation. Thus, during the execution of an entire query set, MP-Rec conditionally matches higher model accuracies of DHE, *hybrid* representations (Table 2).

Insight 2: MP-Rec mitigates the throughput degradation of DHE and *hybrid* representations by conditionally activating more accurate representation(s). Throughput of table-only configurations on either CPUs or GPUs (blue) is higher than that of DHE-, *hybrid*-only configurations (crimson, violet) (Figure 10). On Kaggle, using exclusively DHE or *hybrid* on GPUs for their accuracy benefits degrades $throughput_{correct\_predictions}$ by 62.8% and 63.3%, respectively – compared to exclusively using embedding tables on CPUs. This throughput degradation comes from the increased FLOPs of DHE encoder-decoder stacks. So, even though DHE and *hybrid* execu-
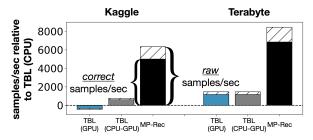
10

Figure 11: Changes in raw throughput (hatched white bars) and *throughput of correct predictions* (colored bars) for *Kaggle* and *Terabyte* use-cases, respectively.

Table 4: Achievable model accuracy, normalized throughput, and memory footprint for design point *HW-2*.

	Achievable Accuracy	Normalized Throughput of Correct Predictions	Memory Capacity
TBL (CPU)	78.721%	1.00×	542 MB
DHE (GPU)	78.936%	0.43×	123 MB
MP-Rec	78.936%	2.26×	CPU: 665 MB GPU: 123 MB

tion paths offer higher prediction accuracies for each *individual query*, $throughput_{correct\_predictions}$ still drops significantly from worse system performance. MP-Rec is able to recover these $throughput_{correct\_predictions}$ degradations by selectively activating DHE, *hybrid* paths based on incoming query characteristics. Namely, MP-Rec schedules queries onto DHE-, *hybrid*-paths when there are no expected latency-bounded throughput degradations.

Insight 3: MP-Rec improves the throughput performance of embedding table baselines by dynamically switching at the hardware platform granularity. Depending on use-case (i.e., base model, query statistics, and latency constraints), optimal execution paths vary by hardware platform for a particular representation. We demonstrate this with an additional baseline where CPU-GPU switching is enabled for table-only representation (gray bars in Figure 10). For example, for Kaggle, purely switching at the CPU-GPU granularity achieves 18% performance improvement over CPU-only execution (Figure 10 (left)). However, for Terabyte, CPU-GPU switching does not enable further speedups since throughput of CPU execution, at best, matches that of GPU execution (Figure 10 (right)). In either case, enabling CPU-GPU switching has a lower-bound performance of optimal static deployment configuration. The reason behind this is that, for throughput, CPU execution is favored when queries are small and model complexity is relatively low (i.e., Kaggle base model). In these scenarios, overheads for GPU-based model execution (e.g., data loading) are less amortized.

Insight 4: MP-Cache increases throughput of correct predictions by decreasing the latency of highly ac-
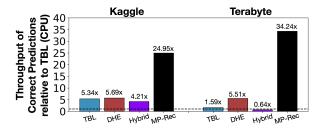

Figure 12: *IPU Query Serving*: If model fits on IPUs and IPUs are able to handle dynamic query sizes, there are potential speedups across different representations.
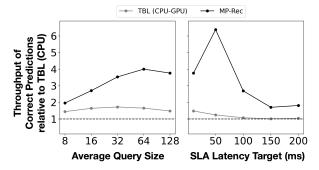

Figure 13: Sensitivity studies for query size and SLA latency target. Default settings assume average query size 128 and SLA target 10 ms. Results shown for Terabyte use-case.

curate representations, namely, DHE and *hybrid*. MP-Cache reduces the long latency of executing DHE, *hybrid* encoder-decoder stacks. This allows these compute-based representations to be viable for more query serving opportunities. Without MP-Cache, MP-Rec will only switch onto long-latency execution paths when there are no expected throughput degradations. Thus, for scenarios like large queries – where table-based execution would have completed under strict latency targets – MP-Cache enables switching onto DHE and/or *hybrid* execution paths. MP-Cache enables MP-Rec to improve system throughput and quality of recommendations served hand in hand. We provide further breakdown in Figure 11.

6.3. Additional Heterogeneous Hardware Case Studies

In addition to evaluating MP-Rec on a large-capacity CPU-GPU system (i.e., HW-1), we expand our analysis to memory-constrained and accelerator-enabled case-studies.

Insight 5: MP-Rec finds optimal representation-HW mappings on constrained HW design points. We introduce design point **HW-2** with constrained memory capacities (Section 5.1). As seen in Table 4, MP-Rec utilizes HW-2's memory capacity budgets with both DHE and TBL execution paths. By doing so, MP-Rec matches optimal accuracy given by DHE (Table 2)) and, at the
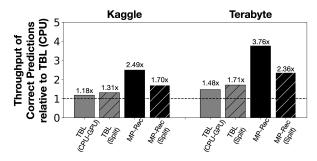
Figure 14: When incorporating DHE and *hybrid*, query splitting is sub-optimal without careful tuning of split ratios. Baseline is embedding table-CPU execution.
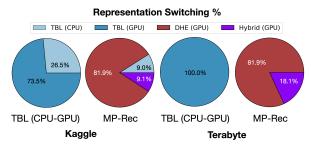

Figure 15: Switching Breakdown. MP-Rec enables execution of compute-based representations.

Figure 16: In recommendation workloads, ID access frequencies follow power law distributions. With both MP-Cache$_{encoder}$ and MP-Cache$_{decoder}$, MP-Cache closes the performance gap between encoder-decoder stacks and embedding tables.

increases. This is because the larger the query sizes, the more GPU/accelerator-offloading opportunities. Figure 13 (right) shows that, as SLA latency target increases, speedup reduces. This is because when latency target budget for query execution is so high (e.g., 200ms), even CPU-embedding table baseline can achieve high throughput execution.

6.5. Additional Optimization: Query Splitting

In order to better exploit heterogeneous hardware platforms, one potential optimization that can be applied on top of this study is query splitting. In theory, splitting a query for a given representation across both CPU and GPU can better utilize available resources and result in lower query load per hardware platform. In Figure 14, we explore this by evenly splitting each query for a representation across both CPU and GPU. We observe that for embedding table configurations, query splitting is better than the CPU-GPU switching baseline. However, for MP-Rec, where compute-intensive representations like DHE and *hybrid* are available, even query splitting is detrimental to performance. This is because for embedding table execution, query splitting results in smaller queries that CPUs are effective for. However, query splitting for compute-intensive representations forces CPU execution, which is extremely ineffective (see Figure 5).

6.6. Dynamic Multi-Path Activation

During the offline phase, each representation – Table, DHE, and *hybrid* – are mapped onto both CPU and/or GPU platforms. During the online phase, MP-Rec switches between representation execution paths given different input query sizes and application SLA target. For example, when input query is small (i.e., $\mathcal{O}(10)$), we activate table-CPU execution for tight SLA targets and *hybrid*/DHE-GPU for medium SLA targets. For large query sizes (i.e., $\mathcal{O}(100)$), table execution swaps onto GPU platform and the *hybrid*/DHE-GPU path is only activated if there are no expected throughput degradations. *For each query, we activate the path of highest*

same time, achieves higher throughput of CPU embedding table execution (Table 4 (left)).

Insight 6: IPUs can offer potential speedups in heterogeneous hardware platforms – given sufficiently large multi-node configurations and further software support. We evaluate an IPU-POD16 in our query serving experiment for both Kaggle and Terabyte (Figure 12). We choose a pod-level configuration – as opposed to chip- and board-level configurations – since the Terabyte model is on the order of 10 GBs and accessing backup DRAM significantly hampers performance (see Section 3.4). IPU's ability to first fit entire DHE stacks on-chip then leverage data parallelism across multiple nodes enables large potential speedups on DHE and MP-Rec configurations. Table and *hybrid* configurations for Terabyte offer less speedup from the lack of data parallelism – each of the 16 IPU nodes dedicate its on-chip SRAM to storing unique shards of the model's parameters. We assume that the IPU is able to handle incoming queries of different sizes. In practice, changes in input shapes require lengthy (i.e., ∼ 30 minutes) re-compilations.

6.4. Sensitivity Studies

Figure 13 showcases sensitivity studies over the dimensions of query size and SLA latency target. When varying average query size, we maintain a log-normal distribution for the query set (Section 5.3). Figure 13 (left) shows that both table CPU-GPU switching and MP-Rec show more improvement as the average query size
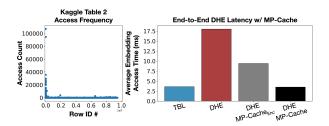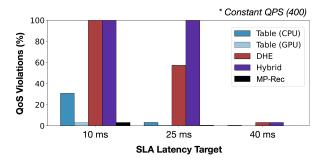
Figure 17: At constant throughput, MP-Rec reduces SLA latency target violations by dynamically switching to suitable representation-hardware execution paths.

Figure 18: Through model *compression*, DHE allows multi-node recommendation models to be run on a single node, reducing communication overheads.

recommendation accuracy if it won't lead to performance degradation. Otherwise, we activate embedding table paths to ensure throughput and latency targets are met.

Figure 15 presents representation switching breakdown of table (CPU-GPU switching) and MP-Rec for both Kaggle and Terabyte. For Kaggle, we see that TBL (CPU) is always present since the execution time of small queries on Kaggle is too fast for the GPU offloading overhead be effectively amortized. For Terabyte, we see that TBL (GPU) is always preferable compared to TBL (CPU). This contributes to the equal performance of the TBL(GPU) and TBL(CPU-GPU) configurations, as what Figure 10 shows.

6.7. MP-Cache Result Analysis

Figure 16 shows how MP-Cache exploits (a) access frequency and (b) value similarity. The access frequency opportunities come from power law distribution of recommendation workloads. Figure 16 (a) depicts the access distribution for Criteo Kaggle. When we analyze the access counts of the largest sparse feature (Embedding table 2 comes with 10M entries and 3M total accesses), we confirm that hot row IDs have 10K+ access counts while others are barely accessed more than once, if at all.

MP-Cache$_{encoder}$ statically exploits the access frequency locality pattern to speed up the encoder-decoder stack. With only 2KB dedicated to MP-Cache$_{encoder}$, we see $1.57\times$ performance improvement over using the entire encoder-decoder stack. With a 2MB cache, the performance improvement becomes $1.92\times$. To further close the latency gap between encoder-decoder stacks and embedding tables ($\sim 5\times$ difference), MP-Cache$_{decoder}$ converts the compute-heavy MLP in decoder stacks to kNN search. When all vectors are normalized, kNN search becomes parallelizable dot product, leading to further speedup. Figure 16 (right) shows that MP-Cache achieves comparable performance level for DHE as embedding table access.
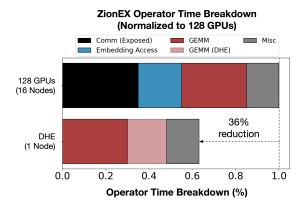
6.8. Reducing SLA Violations

MP-Rec reduces SLA target latency violations at constant throughput scenario (Figure 17). When we statically deploy a representation, SLA latency violations occur at constant throughput when the input query is too large to finish under latency target. At an SLA latency target of 10 ms, statically deploying embedding tables on CPUs will lead to 30.73% of queries violating SLA latency target. Without MP-Rec or MP-Cache, statically deploying DHE or *hybrid* at 400 QPS will lead to 100% SLA violation.

With MP-Rec, we dynamically switch to representations that help us meet target SLA latency target. Figure 17 shows that across a whole range of SLA latency targets, MP-Rec reduces percentage of queries violating SLA latency targets. Compared to embedding table-CPU execution, MP-Rec observes 3.14% SLA latency violations (27.59% improvement) at 10ms latency target.

6.9. Scaling Analysis for Multi-Node Systems

Production recommendation models have terabyte-scale embedding tables, leading to the requirement of multi-node hardware systems for both inference and training tasks [2, 16, 20, 36, 57, 59]. To distribute such model across multiple nodes, embedding tables have to be sharded. During execution, communication collectives, such as `All-to-All` and `AllReduce`, are used to gather embedding-table lookup and data-parallel MLP results from different compute nodes. These collectives contribute to inter-node communication time [43], which can be costly from a system performance perspective. For large-scale recommendation training systems, such as ZionEX [37], exposed inter-node communication contributes to nearly 40% of the total model training time.

DHE can reduce the memory capacity requirement of the Terabyte benchmark by $334\times$ (Figure 4). With this compression, MP-Rec can potentially enable larger-size, industry-scale recommendation models for single-node

13

systems, mitigating the exposed multi-node communication time at the cost of additional DHE-specific computation (Figure 18). Based on our analytical model, for a 128-GPU ZionEX system, the total execution time can be reduced by 36% by replacing embedding tables with DHE, thus eliminating inter-node communication [37].

7. Related Work

State-of-the-art neural recommender systems use embedding tables for their embedding representation, resulting in substantial memory capacity requirements [6, 38, 60, 61, 62]. Recently proposed compute-based representations reduce these memory capacity constraints at the cost of increasing FLOPs [29, 54]. These works have explored point solutions for different embedding representations. In this paper, we explore the interaction between embedding representations and other algorithmic and systems metrics of interest.

Earlier works overview breakdown of recommendation workloads across server-class CPU and GPU systems [1, 13, 15, 19, 58]. Additionally, given the importance of neural recommendation, there are many proposals for custom hardware accelerators [3, 14, 21, 23, 32, 35]. In particular, many of these accelerators are DRAM- [30, 34, 39] and SSD-level [47, 49] modifications aimed at improving embedding access time.

Other works that take advantage of heterogeneous hardware platforms include FleetRec [24] and Hercules [31]. FleetRec demonstrates that, for a fixed recommendation model, operator-level splitting is faster than CPU-only execution. In contrast, we switch between heterogeneous embedding representations, resulting in accuracy improvements that do not exist in fixed-model approaches. If we apply both operator-level (i.e., FleetRec) and representation-level (i.e., MP-Rec) parallelisms for a recommendation workload, we could see improvements stemming from improved embedding table execution efficiency. For appropriate query sizes, *hybrid* execution will be activated more often due to speedups in its table stack. However, speedup would ultimately be limited as *hybrid* execution bottlenecks lie in its DHE stacks (Figure 5). On the other hand, Hercules identifies optimal heterogeneous server architectures and system resource mappings – especially in the context of diurnal cycles of recommendation workloads.

Other embedding table optimizations usually fall under one of two objectives: *compression* or *faster access*. TT-Rec [54], ROBE [8], and [12, 45] leverage matrix factorization and other related weight sharing/reduction techniques to reduce overall memory footprint of embedding tables. Other works such as cDLRM [5], Bandana [9], RecShard [43], DreamShard [56], AutoShard [55], FlexShard [44], and Kraken [52] leverage access frequency information to make embedding access more hardware-efficient.

8. Conclusion

We explore alternative embedding representations for deep learning recommendation, where embedding vectors can be generated from embedding table lookups and/or encoder-decoder stacks. This is fundamentally different from prior work, where user-item relationships are learned using a single feature representation. To maximize throughput of correct predictions while meeting tail latency requirements, we propose a new representation-system co-design approach for real-time inference, MP-Rec. Depending on memory capacities of AI inference systems, MP-Rec selects unique embedding representations, forming multiple embedding execution paths for recommendation inference. At runtime, depending on input query sizes and application-dependent performance targets, MP-Rec activates embedding path(s) to jointly maximize model quality and throughput. Using the open-source MLPerf-DLRM with Kaggle and Terabyte datasets, MP-Rec achieves higher throughput of correct predictions and model quality at the same time while meeting application-specific tail latency requirements.

9. Acknowledgements

References

[1] B. Acun, M. Murphy, X. Wang, J. Nie, C. Wu, and K. Hazelwood. Understanding training efficiency of deep learning recommendation models at scale. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 802–814, Los Alamitos, CA, USA, mar 2021. IEEE Computer Society.

[2] Muhammad Adnan, Yassaman Maboud, Divya Mahajan, and Prashant J. Nair. High-performance training by exploiting hot-embeddings in recommendation systems. In *Proceedings of the 48th International Conference on Very Large Data Bases (VLDB)*, volume 15. VLDB Endowment, 2021.

[3] Michael Anderson, Benny Chen, Stephen Chen, Summer Deng, Jordan Fix, Michael Gschwind, Aravind Kalaiah, Changkyu Kim, Jaewon Lee, Jason Liang, Haixin Liu, Yinghai Lu, Jack Montgomery, Arun Moorthy, Satish Nadathur, Sam Naghshineh, Avinash Nayak, Jongsoo Park, Chris Petersen, Martin Schatz, Narayanan Sundaram, Bangsheng Tang, Peter Tang, Amy Yang, Jiecao Yu, Hector Yuen, Ying Zhang, Aravind Anbudurai, Vandana Balan, Harsha Bojja, Joe Boyd, Matthew Breitbach, Claudio Caldato, Anna Calvo, Garret Catron, Sneh Chandwani, Panos Christeas, Brad Cottel, Brian Coutinho, Arun Dalli, Abhishek Dhanotia, Oniel Duncan, Roman Dzhabarov, Simon Elmir, Chunli Fu, Wenyin Fu, Michael Fulthorp, Adi Gangidi, Nick Gibson, Sean Gordon, Beatriz Padilla Hernandez, Daniel Ho, Yu-Cheng Huang, Olof Johansson, Shishir Juluri, Shobhit Kanaujia, Manali Kesarkar, Jonathan Killinger, Ben Kim, Rohan Kulkarni, Meghan Lele, Huayu Li, Huamin Li, Yueming Li, Cynthia Liu, Jerry Liu, Bert Maher, Chandra Mallipedi, Seema Mangla, Kiran Kumar Matam, Jubin Mehta, Shobhit Mehta, Christopher Mitchell, Bharath Muthiah, Nitin Nagarkatte, Ashwin Narasimha,

Bernard Nguyen, Thiara Ortiz, Soumya Padmanabha, Deng Pan, Ashwin Poojary, Ye (Charlotte) Qi, Olivier Raginel, Dwarak Rajagopal, Tristan Rice, Craig Ross, Nadav Rotem, Scott Russ, Kushal Shah, Baohua Shan, Hao Shen, Pavan Shetty, Krish Skandakumaran, Kutta Srinivasan, Roshan Sumbaly, Michael Tauberg, Mor Tzur, Sidharth Verma, Hao Wang, Man Wang, Ben Wei, Alex Xia, Chenyu Xu, Martin Yang, Kai Zhang, Ruoxi Zhang, Ming Zhao, Whitney Zhao, Rui Zhu, Ajit Mathews, Lin Qiao, Misha Smelyanskiy, Bill Jia, and Vijay Rao. First-generation inference accelerator deployment at facebook, 2021.

[4] Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I. Shamir, Rakesh Shivanna, and Qiqi Yan. On the factory floor: Ml engineering for industrial-scale ads recommendation models, 2022.

[5] Keshav Balasubramanian, Abdulla Alshabanah, Joshua D Choe, and Murali Annavaram. *CDLRM: Look Ahead Caching for Scalable Training of Recommendation Models*, page 263–272. Association for Computing Machinery, New York, NY, USA, 2021.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Zhaoxia Deng, Jongsoo Park, Ping Tak Peter Tang, Haixin Liu, Jie Yang, Hector Yuen, Jianyu Huang, Daya Khudia, Xiaohan Wei, Ellie Wen, Dhruv Choudhary, Raghuraman Krishnamoorthi, Carole-Jean Wu, Satish Nadathur, Changkyu Kim, Maxim Naumov, Sam Naghshineh, and Mikhail Smelyanskiy. Low-precision hardware architectures meet recommendation model inference at scale. *IEEE Micro*, 41(5):93–100, 2021.

[8] Aditya Desai, Li Chou, and Anshumali Shrivastava. Random offset block embedding (robe) for compressed embedding tables in deep learning recommendation systems. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 762–778, 2022.

[9] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. Bandana: Using non-volatile memory for storing deep learning models, 2018.

[10] Karl Freund and Patrick Moorhead. The graphcore second generation ipu, 2020.

[11] Amir Gholami, Zhewi Yao, Sehoon Kim, Michael W. Mahoney, and Kurt Keutzer. Ai and memory wall, 2021.

[12] A.A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory (ISIT)*, page 2786–2791. IEEE Press, 2021.

[13] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 982–995. IEEE Press, 2020.

[14] Udit Gupta, Samuel Hsia, Jeff Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin Sean Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. Recpipe: Co-designing models and hardware to jointly optimize recommendation quality and performance. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 870–884, New York, NY, USA, 2021. Association for Computing Machinery.

[15] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501, 2020.

[16] Vipul Gupta, Dhruv Choudhary, Peter Tang, Xiaohan Wei, Xing Wang, Yuzhen Huang, Arun Kejariwal, Kannan Ramchandran, and Michael W. Mahoney. 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, page 2928–2936, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629, 2018.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 173–182, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[19] S. Hsia, U. Gupta, M. Wilkening, C. Wu, G. Wei, and D. Brooks. Cross-stack workload characterization of deep recommendation systems. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 157–168, Los Alamitos, CA, USA, oct 2020. IEEE Computer Society.

[20] Yuzhen Huang, Xiaohan Wei, Xing Wang, Jiyan Yang, Bor-Yiing Su, Shivam Bharuka, Dhruv Choudhary, Zewei Jiang, Hai Zheng, and Jack Langman. 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, page 3050–3058, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 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, page 968–981. IEEE Press, 2020.

[22] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking, 2019.

[23] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B. Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. Microrec: Efficient recommendation inference by hardware and data structure solutions. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 845–859, 2021.

[24] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. Fleetrec: Large-scale recommendation inference on hybrid gpu-fpga clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 3097–3105, New York, NY, USA, 2021. Association for Computing Machinery.

[25] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten lessons from three generations shaped google's tpuv4i : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14, 2021.

[26] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. *Commun. ACM*, 63(7):67–78, jun 2020.

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

[28] Criteo Kaggle. Display advertising challenge: Predict click-through rates on display ads, 2014.

[29] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery &and Data Mining*, KDD '21, page 840–850, New York, NY, USA, 2021. Association for Computing Machinery.

[30] 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. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 790–803. IEEE, 2020.

[31] Liu Ke, Udit Gupta, Mark Hempstead, Carole-Jean Wu, Hsien-Hsin S. Lee, and Xuan Zhang. Hercules: Heterogeneity-aware inference serving for at-scale personalized recommendation. In *2022 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2022.

[32] 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 S. Lee. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro*, 42(1):116–127, 2022.

[33] Daya Khudia, Jianyu Huang, Protonu Basu, Summer Deng, Haixin Liu, Jongsoo Park, and Mikhail Smelyanskiy. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

[34] Youngeun Kwon, Yunjae Lee, and Minsoo 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*, pages 740–753, 2019.

[35] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. Tensor casting: Co-designing algorithm-architecture for personalized recommendation training. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 235–248, 2021.

[36] M. Lui, Y. Yetim, O. Ozkan, Z. Zhao, S. Tsai, C. Wu, and M. Hempstead. Understanding capacity-driven scale-out neural recommendation inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 162–171, Los Alamitos, CA, USA, mar 2021. IEEE Computer Society.

[37] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie (Amy) Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 993–1011, New York, NY, USA, 2022. Association for Computing Machinery.

[38] 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. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[39] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. Trim: Enhancing processor-memory interfaces with scalable tensor reduction in memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 268–281, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, Juan Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications, 2018.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[42] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 446–459. IEEE Press, 2020.

[43] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. Recshard: Statistical feature-based memory optimization for industry-scale neural recommendation. In *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[44] Geet Sethi, Pallab Bhattacharya, Dhruv Choudhary, Carole-Jean Wu, and Christos Kozyrakis. Flexshard: Flexible sharding for industry-scale sequence recommendation models. 2023.

[45] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. *CoRR*, abs/1909.02107, 2019.

[46] Criteo Terabyte. Criteo terabyte click logs, 2013.

[47] Hu Wan, Xuan Sun, Yufei Cui, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. *FlashEmbedding: Storing Embedding Tables in SSD for Large-Scale Recommender Systems*, page 9–16. Association for Computing Machinery, New York, NY, USA, 2021.

[48] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 839–848, New York, NY, USA, 2018. Association for Computing Machinery.

[49] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 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)*, 2021.

[50] Carole-Jean Wu, Robin Burke, Ed Chi, Joseph A. Konstan, Julian J. McAuley, Yves Raimond, and Hao Zhang. Developing a recommendation benchmark for mlperf training and inference. *CoRR*, abs/2003.07336, 2020.

[51] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. Sustainable ai: Environmental implications, challenges and opportunities. In *Proceedings of Machine Learning and Systems*, volume 4, pages 795–813, 2022.

[52] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. Kraken: Memory-efficient continual learning for large-scale real-time recommendations. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17, 2020.

[53] Xinyang Yi, Yi-Fan Chen, Sukriti Ramesh, Vinu Rajashekhar, Lichan Hong, Noah Fiedel, Nandini Seshadri, Lukasz Heldt, Xiang Wu, and Ed H. Chi. Factorized deep retrieval and distributed tensorflow serving. In *Proceedings of Machine Learning and Systems*, SysML'18, 2018.

[54] Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. Tt-rec: Tensor train compression for deep learning recommendation models. In *Proceedings of Machine Learning and Systems*, volume 3, pages 448–462, 2021.

[55] Daochen Zha, Louis Feng, Bhargav Bhushanam, Dhruv Choudhary, Jade Nie, Yuandong Tian, Jay Chae, Yinbin Ma, Arun Kejariwal, and Xia Hu. Autoshard: Automated embedding table sharding for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 4461–4471, New York, NY, USA, 2022. Association for Computing Machinery.

[56] Daochen Zha, Louis Feng, Qiaoyu Tan, Zirui Liu, Kwei-Herng Lai, Bhargav Bhushanam, Yuandong Tian, Arun Kejariwal, and Xia Hu. Dreamshard: Generalizable embedding table placement for recommender systems. 2022.

[57] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. Distributed hierarchical gpu parameter server for massive scale deep learning ads systems, 2020.

[58] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. Distributed hierarchical gpu parameter server for massive scale deep learning ads systems. In *MLSys*, 2020.

[59] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. Aibox: Ctr prediction model training on a single node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 319–328, New York, NY, USA, 2019. Association for Computing Machinery.

[60] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: A multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, pages 43–51, New York, NY, USA, 2019. ACM.

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

[62] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068. ACM, 2018.

A. Artifact Appendix

A.1. Abstract

This artifact package includes CPU- and GPU-compatible PyTorch-based implementations of proposed deep learning recommendation architectures (i.e., embedding representations), range of relevant hyperparameters, and custom TPU-, IPU-compatible implementations. The base implementation is compatible with PyTorch-compliant CPUs and GPUs while TPU-, IPU-implementations require access to cloud-hosted TPUs/IPUs and their associated PyTorch branches (i.e., PyTorch/XLA and poptorch). Inference experiments and characterization require at least a single CPU/GPU node (TPU-IPU benchmarking scripts support single- and multi-node execution) while design space exploration involving accuracy evaluation is best executed with large-scale GPU clusters due to the large number of GPU training jobs. Open-sourcing these implementations of proposed embedding representations will allow other researchers to not only characterize and deploy these algorithmic innovations on their own choice of systems but also develop novel embedding processing techniques of their own.

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

- **Algorithm:** embedding representations; Deep Learning Recommendation Model (DLRM); recommendation systems
- **Compilation:** (for TPU) XLA; (for IPU) PopART
- **Model:** Deep Learning Recommendation Model (DLRM) – publicly available at `https://github.com/facebookresearch/dlrm` ($2 \sim 12$GB, depending on model architecture configuration); other model architecture modifications in artifact packages
- **Data set:** Criteo Kaggle/Terabyte benchmarks – publicly available via MLCommons MLPerf benchmarks; artifact also provides instructions on how to synthetically generate Kaggle/Terabyte-like input data for characterization purposes
- **Run-time environment:** Linux
- **Hardware:** (base benchmarks) CPU, GPU; (additional benchmarks) TPU ($1\sim8$ core configurations), IPU ($1\sim16$ chip configurations)
- **Run-time state:** yes – embedding access patterns may affect measured system performance
- **Execution:** sole-user; profiling
- **Metrics:** execution time; (commented out) operator breakdown; model accuracy
- **Output:** profiled timing information; (commented out) operator breakdowns; training accuracy
- **Experiments:** base experiments print out overall timing information (operator breakdown commented out)
- **How much disk space required (approximately)?:** (without Criteo Kaggle/Terabyte datasets) 50 GB
- **How much time is needed to prepare workflow (approximately)?:** (without setting up Criteo Kaggle/Terabyte datasets and on CPU-GPUs) 20 min; (TPU-IPU benchmarks) associated software stack setups (see official PyTorch-TPU/IPU tutorials) require 1+ hr
- **How much time is needed to complete experiments (approximately)?:** inference runs take minutes to reach steady state; training runs take $18\sim24$ hours per run – fully exploring design space requires 100s of training runs.
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT License
- **Data licenses (if publicly available)?:** Criteo Terabyte License (`https://ailab.criteo.com/criteo-1tb-click-logs-dataset/`)
- **Workflow framework used?:** (training runs) slurm
- **Archived (provide DOI)?:** Yes; `https://github.com/samhsia/MP-Rec-AE`

A.3. Description

A.3.1. How to access Clone repository from `https://github.com/samhsia/MP-Rec-AE`

A.3.2. Hardware dependencies Any CPU/GPU should be able to run the reference recommendation workloads. As described in the paper, we experimented on server-class (Xeon) Intel CPUs and NVIDIA (CUDA-enabled) GPUs. Any CPU/GPU platform that is compatible with

PyTorch (see **Software dependencies** subsection – also duplicated in `dlrm_mprec/requirements.txt`) will work. TPUs and IPUs available via cloud are compatible with the custom TPU and IPU implementations that leverage modules within their associated software packages.

A.3.3. Software dependencies PyTorch; PyTorch/XLA; poptorch. Required PyTorch packages:

- future
- numpy
- onnx
- pydot
- torch
- torchviz
- scikit-learn
- tqdm
- torchrec
- torchx
- primesieve

A.3.4. Data sets Criteo Kaggle/Terabyte. For instructions on how to generate data in the shape of Criteo Kaggle and Terabyte datasets, see `dlrm_mprec/configurations.txt`. We also provide a script to download Criteo Kaggle in `dlrm_mprec/download_kaggle.sh`.

A.3.5. Models Base DLRM and variations: DHE, *hybrid*, and *select*. See paper for more description and chracterization on these variations and `dlrm_mprec/configurations.txt` for sample commands for running each of these variations on both CPUs and GPUs.

A.4. Installation

Use `pip`, `conda`, or your choice of Python package manager to install requirements listed above in **Software dependencies** subsection (also in `dlrm_mprec/requirements.txt`).

A.5. Evaluation and expected results

`python dlrm_s_pytorch.py –mini-batch-size=2 –data-size=6` should be your first command to ensure within installation works. If the aforementioned command works, you should try running the other embedding representations with commands listed in: `dlrm_mprec/configurations.txt`. You should see profiled timing information.

A.6. Experiment customization

Listed in `dlrm_mprec/configurations.txt` are DHE stack hyperparameters relevant to the characterization and discussion in the paper. We list ranges for number of encoder hash functions and decoder MLP shapes.