2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)

Heterogeneous Acceleration Pipeline for Recommendation System Training

Muhammad Adnan[†] Yassaman Ebrahimzadeh Maboud[†] Divya Mahajan[⋆] Prashant J. Nair[†]

[†]The University of British Columbia [⋆]Georgia Institute of Technology

{adnan, yassaman, prashantnair}@ece.ubc.ca divya.mahajan@gatech.edu

***Abstract*—Recommendation models rely on deep learning networks and large embedding tables, resulting in computationally and memory-intensive processes. These models are typically trained using hybrid CPU-GPU or GPU-only configurations. The hybrid mode combines the GPU's neural network acceleration with the CPUs' memory storage and supply for embedding tables but may incur significant CPU-to-GPU transfer time. In contrast, the GPU-only mode utilizes High Bandwidth Memory (HBM) across multiple GPUs for storing embedding tables. However, this approach is expensive and presents scaling concerns.**

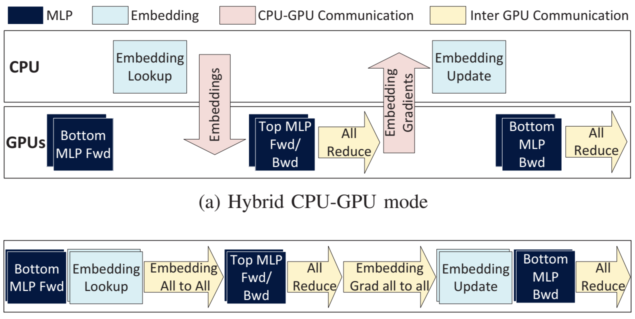
This paper introduces Hotline, a heterogeneous acceleration pipeline that addresses these concerns. Hotline develops a data-aware and model-aware scheduling pipeline by leveraging the insight that only a few embedding entries are frequently accessed (popular). This approach utilizes CPU main memory for non-popular embeddings and GPUs' HBM for popular embeddings. To achieve this, Hotline accelerator fragments a mini-batch into popular and non-popular micro-batches (μ-batches). It gathers the necessary working parameters for non-popular μ-batches from the CPU, while GPUs execute popular μ-batches. The hardware accelerator dynamically coordinates the execution of popular embeddings on GPUs and non-popular embeddings from the CPU's main memory. Real-world datasets and models confirm Hotline's effectiveness, reducing average end-to-end training time by 2.2× compared to Intel-optimized CPU-GPU DLRM baseline.

***Index Terms*—Recommender Systems, Multi-Node Distributed Training, Accelerators.**

I. Introduction

Recommendation models constitute a crucial and widely deployed class of machine learning (ML) workloads [1]. These models employ compute-intensive neural networks and memory-intensive embedding tables to store user and item features [2]. With the increasing number of interactions between users and items, the size of these tables is expected to grow significantly. Production-scale models can already reach several terabytes and contain trillions of parameters [3–5].

The deep Learning Recommendation Model (DLRM) and the Time-Based Sequence Model (TBSM) are popular commercial models. These models are typically trained using either a hybrid CPU-GPU mode or a GPU-only mode [6, 7]. In the hybrid mode (Figure 1a), the CPU provides memory capacity for the embedding entries, while GPUs offer high-throughput data-parallel neural network execution [8]. However, this mode suffers from inefficiencies due to three reasons: (1) GPUs rely on the CPU to provide all embeddings, (2) the low-bandwidth CPU memory acts as a bottleneck, and (3) the CPU's execution of the embedding logic prevents full GPU utilization throughout the training process.



(a) Hybrid CPU-GPU mode

(b) GPU-only mode

Fig. 1: The execution flow of a typical recommendation model in the hybrid CPU-GPU and GPU-only. Due to their large sizes, the embedding tables are stored and processed on CPUs. The GPUs process the neural layers.

Alternatively, the GPU-only mode, illustrated in Figure 1b, employs multiple GPUs to store a single copy of the embeddings and trains in a model-parallel manner [9]. However, this approach necessitates continuous all-to-all communication between GPUs to share their embeddings. Additionally, in this mode, one would need to grow the GPUs to enable the training of larger datasets. For instance, the Terabyte dataset requires at least *four* NVIDIA V100 GPUs to fit its embeddings. Ideally, even larger applications must be enabled using fewer GPUs.

To overcome these limitations of existing training modes, this paper introduces a novel heterogeneous acceleration pipeline called Hotline. The primary goal of Hotline is to fully exploit GPUs' compute throughput and CPU-based memory capacity without encountering any communication or computation bottlenecks. By combining the advantages of both the hybrid and GPU-only modes, Hotline utilizes the GPU-only mode for the entire training process. It leverages the CPU-based main memory to store the majority of embeddings.

This is achieved through an innovative hardware accelerator that pipelines the embedding gathering operation of the CPU together with the compute on GPUs. This approach ensures that GPUs are continuously fed from either the CPU-based memory subsystem or their High Bandwidth Memories

979-8-3503-2658-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ISCA59077.2024.00081

(HBM), avoiding stalls in the process. As a result, Hotline provides a scalable throughput-optimized solution. Broadly, the Hotline acceleration pipeline relies on two key insights.

1. Access-aware Embedding Layout in Memory: Hotline leverages the observation that real-world recommender systems exhibit a high skew in popularity, causing certain embedding entries to be accessed significantly more frequently than others [10–14]. These frequently accessed embeddings, referred to as *frequently-accessed entries*, have a small memory footprint but are computationally significant. To take advantage of this access property, Hotline proposes an optimized *access-aware memory layout for embeddings*. The hardware accelerator in Hotline dynamically classifies frequently-accessed embeddings and places them on the GPU memory, while not-frequently-accessed embeddings are stored in the CPU main memory. The hardware accelerator periodically monitors the access pattern to ensure it captures the most up-to-date trends in the training data.

2. Layout-aware Runtime Scheduling: Hotline employs a dynamic runtime scheduler to achieve optimal compute throughput with the new memory placement. This scheduler divides a mini-batch into two micro-batches (μ-batches), and subsequently, these μ-batches are categorized into two groups. The first category comprises popular inputs that exclusively access frequently-accessed embeddings and are directly scheduled onto the GPUs. Remarkably, we find that approximately 75% of the inputs fall into the first category across a wide range of models. On the other hand, the second category consists of the remaining inputs that may access both frequently-accessed and not-frequently-accessed embeddings. If an input accesses even a single non-frequently-accessed embedding, it is classified as a non-popular input. For the first category of popular inputs, all the required embeddings for the micro-batch are directly scheduled onto the GPUs. In contrast, Hotline gathers the not-frequently-accessed embeddings from the CPU memory for the second category. However, accessing the CPU's main memory can stall the pipeline in such a system. This is because the not-frequently-accessed embeddings are stored in the CPU's main memory and are in the critical path.

To overcome this challenge, Hotline introduces a novel hardware accelerator that schedules the μ-batches in a data-aware and model-aware manner. By doing so, Hotline ensures that GPUs are not starved, allowing the system to gather the not-frequently-accessed embeddings from the CPU memory while executing the frequently-accessed inputs on the GPUs.

Contributions: This work makes three key contributions:

1) Identifies frequently-accessed embeddings *dynamically* at runtime with negligible overhead.
2) Offers a dynamic data and model-aware scheduler to efficiently pipeline the mini-batch dispatch onto GPUs while concurrently obtaining not-frequently-accessed embeddings from the main memory.
3) Offers a runtime framework that increases training throughput by stitching the training process of the recommender model across CPUs and GPUs.

We evaluated Hotline with publicly available deep learning (DLRM) and time-sequence (TBSM) based recommendation models and compared our approach against two baselines - hybrid CPU-GPU and GPU-only baseline.

We compared Hotline against state-of-the-art deep learning frameworks such as XDL [15], FAE [10], and Intel-optimized DLRM [16]. On average, Hotline provides 3.4× speedup over the 4-GPU XDL, 1.4× over FAE, and 2.2× speedup over Intel optimized DLRM. It is noteworthy that Hotline only rearranges inputs in a single mini-batch, but the updates to the model are performed at parity with the baseline. Thus, Hotline does not impact the accuracy or training fidelity of the model. Hotline could train larger models, such as Criteo Terabyte, with a single GPU, whereas the GPU-only baseline required at least 4 GPUs to store its embeddings.

II. Recommendation Systems

Figure 2 illustrates the general structure of deep-learning-based recommendation models, which rely on two types of inputs: dense and sparse. Dense inputs are continuous features, such as the user's age, while sparse inputs represent categorical features, such as the user's location or videos they have liked. The neural network component processes dense inputs using Multi-Layer Perceptron (MLP) techniques, while massive embedding tables handle sparse inputs. Each embedding table represents a categorical feature, with the number of rows corresponding to the possible items associated with that feature. An MLP processes the outputs of both the dense and sparse inputs to generate a prediction, such as the likelihood of clicking or click-through rate (CTR).

A. Training Setup

Large recommender models can be trained using two distributed modes: the hybrid mode and the data-parallel mode. In the hybrid mode, embeddings are stored and gathered on the CPU, while neural networks are executed on GPUs in a data-parallel manner. However, the training throughput of the hybrid mode is often limited due to substantial data transfers and reliance on low-bandwidth CPU main memory.

1) Hybrid CPU-GPU Mode: Figure 3 illustrates the distribution of training time across four real-world models and datasets. The results highlight that embedding operations, such

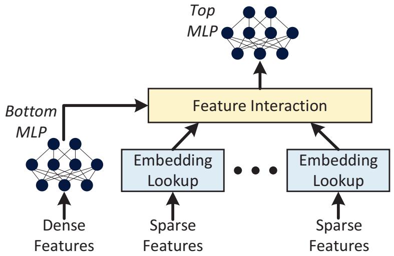


Fig. 2: General structure of a deep-learning-based recommendation model [6, 7, 17]. It consists of compute-bound neural networks and memory-bound embedding tables.

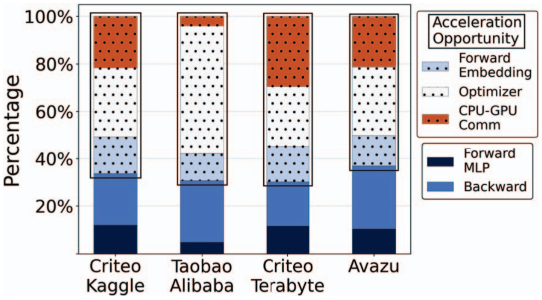


Fig. 3: The breakdown of the training time for an Intel-optimized DLRM with 4-GPU in a hybrid CPU-GPU training setup. The dotted parts of the bar are executed on the CPU and present an opportunity for GPU-based acceleration.

as embedding-lookup in the forward pass, updating embeddings in the optimizer, and CPU-GPU communication, can account for up to 75% of the training time in large datasets, like Criteo Terabyte.

2) Single Node GPU-only Mode: In the GPU-only mode, multiple GPUs are used to store all embeddings and perform data-parallel neural network execution. However, this mode experiences low compute utilization primarily because recommendation models grow with the size of the embedding tables, resulting in a larger memory footprint rather than an increase in neural compute [3–5]. Consequently, the GPU devices must scale with the size of the embedding table rather than the neural network's compute intensity. Figure 4 illustrates the breakdown of training time for four real-world datasets using a single node GPU-only system with NVLink [18] interconnect.

In a single node GPU-only system, transferring embeddings across all devices requires `all-to-all` collectives. For instance, in a 4-GPU system, we observed that this step consumes nearly 12% of the total training time even after employing the fast NVLink interconnect. As the number of nodes increases, the communication time also grows, potentially limiting scalability and becoming the training bottleneck [19].

3) Multi Node GPU-only Mode: Distributed training across multiple nodes exacerbates communication time, particularly with `all-to-all` collectives. In our Dell EMC C4140 nodes, InfiniBand links provide a bandwidth of only 100Gbps, while NVLink offers 2400Gbps for Nvidia-V100 GPUs. This disparity makes communication a major bottleneck. As shown in Figure 5, communication costs now exceed 50% of the multi-node training time.

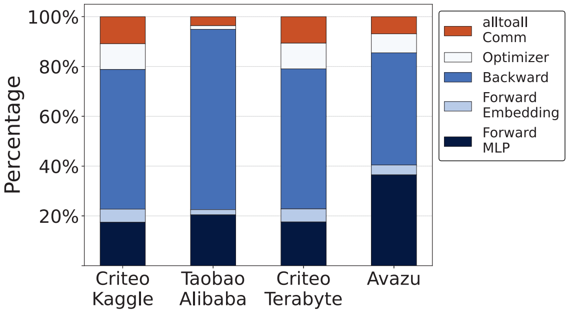


Fig. 4: The breakdown of the training time for DLRM in single node GPU-only training setup. The single-node setup uses NVLink interconnect across four GPUs.

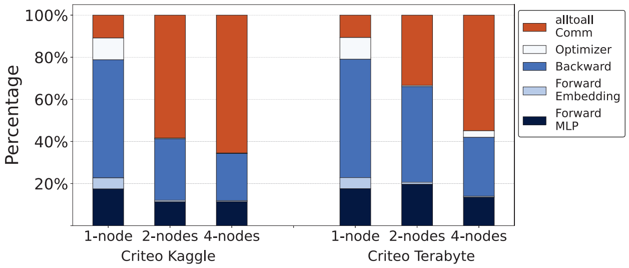


Fig. 5: The training time breakdown for DLRM in a multi-node GPU-only setup with four GPUs per node. The GPUs use NVLink for intra-node GPU connections. The multi-node setup uses 100Gbps InfiniBand for inter-node connectivity.

B. Popularity in Training Inputs

The Hotline framework leverages a fundamental characteristic of recommendation models, where specific users and items exhibit significantly higher popularity than others. This phenomenon leads to certain embeddings being accessed far more frequently than others, as illustrated in Figure 6 across various real-world datasets. Typically, a small number of frequently-accessed embeddings can receive over 100× more access than others, catering to over 75% of inputs with only approximately 512 MB of embeddings. Nevertheless, adapting to changes in input popularity poses a complex challenge.

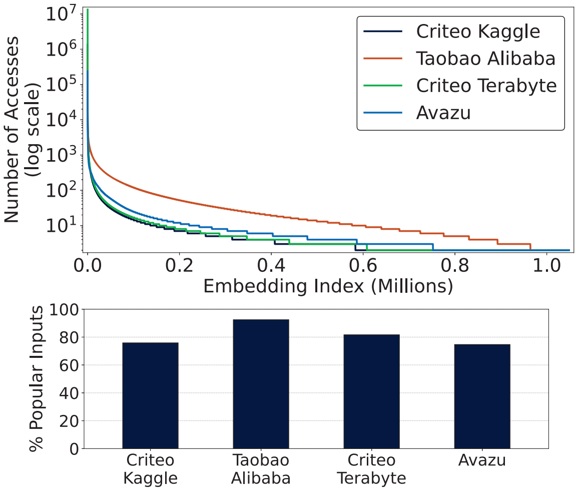


Fig. 6: Number of accesses per embedding entry per one training epoch. Inputs that account for at least 1-in-every-100000 embedding accesses are labelled as *popular*.

III. Challenges and Insights

The frequently-accessed embeddings have a small memory footprint and serve the majority of inputs, making it feasible to place these frequently-accessed embeddings locally across GPUs. This approach could eliminate the need to involve CPUs for most inputs, leading to significant performance benefits. However, this approach faces three key challenges.

Challenge 1 – Embeddings in CPUs and GPUs:

During the training process, mini-batches of input data access a mix of frequently-accessed and non-frequently-accessed embeddings, which are stored across both the CPU main memory and the HBM of GPUs. Consequently, some embeddings from the CPU's main memory must be collected and transmitted to the GPU for embedding computations.

> ***Our Approach****: Hotline partitions each mini-batch into two micro-batches (μ-batches). The inputs in a μ-batch either access* only *frequently-accessed embeddings or any arbitrary embeddings. First, Hotline schedules the μ-batches that access* only *frequently-accessed embeddings on the GPU(s) for execution. Concurrently, it collects the parameters for the μ-batches that access embeddings from the CPU memory.*

Challenge 2 – Segregation and Scheduling with CPU: Achieving efficient mini-batch segregation and parameter gathering can be accomplished using CPUs and GPUs instead of hardware accelerators. However, GPUs are not optimized for fine-grained mini-batch segregation. To address this, CPU-based multi-processing can be employed for mini-batch segregation, parameter gathering, and scheduling.

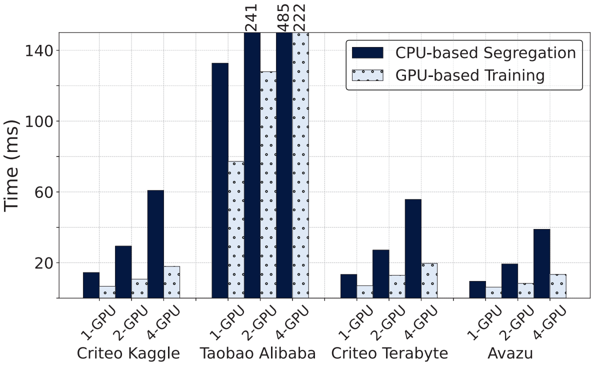


Fig. 7: CPU segregation and scheduling time for a mini-batch using Intel Xeon CPU while the V100 GPU(s) training on a mini-batch. We use mini-batches of 1K, 2K, and 4K inputs for 1, 2, and 4-GPU execution, respectively. Each mini-batch contains two μ-batches (popular and non-popular).

Our study, as illustrated in Figure 7, revealed that even when utilizing all CPU cores, an Intel Xeon CPU exhibits a mini-batch segregation latency up to 2.5× higher than that of NVIDIA-V100 GPU(s) single mini-batch GPU-based training. This is because CPU-based segregation necessitates numerous memory look-ups to determine if an input is high access.

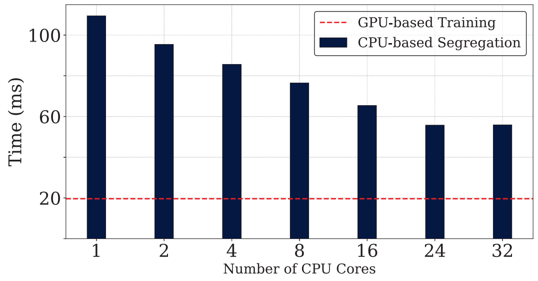


Fig. 8: The wall-clock time by varying CPU cores for segregating a 4K input mini-batch of the Criteo Terabyte dataset. The time overhead of CPU-based segregation plateaus with an increasing number of cores.

We investigated the bottleneck in CPU-based segregation by varying the number of CPU cores for segregating a Criteo Terabyte dataset mini-batch. Figure 8 illustrates that adding cores initially decreases segregation time slightly, but beyond 24 cores, segregation time plateaus. This indicates that the issue lies with parallel memory accesses from the CPU cores rather than CPU compute throughput. Therefore, segregation is memory bound, and even hardware like the Data Streaming Accelerator (DSA) (within Intel Sapphire Rapids), designed for data copying and transformation, would not alleviate the issue due to its inability to handle parallel memory lookups.

As the mini-batch size increases, the processing overhead and latency for CPU-based segregation also increase proportionately. CPUs cannot actively segregate and schedule popular and non-popular μ-batches *before* the GPUs finish their execution. Consequently, our experiments demonstrate that GPUs remain idle for over 50% of the training time.

> ***Our Approach****: Hotline introduces a novel accelerator that utilizes parallel lookup engines capable of performing fine-grained tasks. These tasks include determining whether an input is a high-access or low-access value, fragmenting the mini-batches to form new μ-batches, and enabling efficient parameter gathering for the μ-batches. With the help of this accelerator, the acceleration pipeline can segregate and schedule the non-popular μ-batch as soon as the GPUs finish executing the popular μ-batch.*

Challenge 3 – Evolving Access Skews: Prior studies have used an *offline* profiler [10, 11] to identify frequently-accessed embeddings. Some of these studies do not account for this overhead, up to 15%, in their training times [10]. They also assume that the training data is available before training and that the set of frequently-accessed embeddings does not change over time. However, training data in the recommender models is mostly structured as user activity across some finite time. As user behavior changes rapidly every few hours or days, static profiling may not quickly identify the corresponding shift embedding accesses [14]. Figure 9 shows the change in user behavior for the Criteo Terabyte dataset across days.

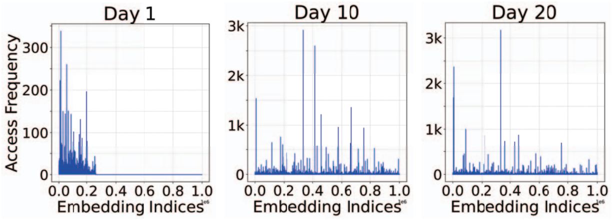


Fig. 9: Evolving skew in training data across days for Terabyte dataset (Embedding Table 20). Thus, popular embeddings vary sometimes as frequently as a few hours.

> ***Our Approach**: To address the abovementioned issues, Hotline adopts a dynamic approach that* samples *a small fraction of inputs (usually 5%) to identify frequently-accessed embeddings. This minimizes the profiling overheads to be ⩽5% while enabling efficient tracking of frequently-accessed embeddings across mini-batches. Furthermore, the accelerator continuously re-calibrates the frequently-accessed embeddings to adapt to changes in training data.*

IV. The Hotline System

The system comprises an InfiniBand-based multi-core server. Each node has multiple GPU devices with inter-GPU communication achieved via NVLink [18]. The Hotline system places the accelerator on the *low profile PCIe slot* that GPUs do not use, enabling it to access the DMA engine through the PCIe switch and communicate directly with the CPU memory as shown in Figure 10. Notably, the Hotline system requires no modifications to the CPU and GPU devices. This system operates in two phases:

1. The Learning Phase: The Hotline accelerator actively determines the frequently-accessed embeddings at runtime. To achieve this, the accelerator performs mini-batch sampling in the first epoch. Our experiments demonstrate that sampling just 5% of the mini-batches is sufficient to identify over 90% of the frequently-accessed embeddings. Based on the sampled mini-batches, the accelerator progressively classifies the accessed embeddings as either frequently-accessed or non-frequently-accessed. Subsequently, the contents of the frequently-accessed embeddings are replicated across all GPUs, and the accelerator memory stores only the *indices* of the frequently-accessed embeddings. In a multi-node setup, the learning phase occurs on a single node's Hotline accelerator, after which the indices of frequently-accessed embeddings are copied to the Hotline accelerators of all nodes.

2. The Acceleration Phase: The Hotline system's *acceleration* phase commences once the frequently-accessed embeddings are replicated on each GPU. During this phase, the Hotline accelerator actively classifies a mini-batch into two $\mu-batches$ based on input popularity. The system employs GPUs to accelerate both $\mu-batches$ through pipelined execution, as depicted in Figure 12. Notably, Hotline operates on a finer scale, updating non-frequently-accessed embeddings on the CPU and frequently-accessed ones on the GPU[1]. As a result, Hotline updates embeddings at distinct locations on CPUs or GPUs, thereby avoiding any coherence requests.

The frequently-accessed embeddings of the popular μ-batch are synchronized across all GPUs with dense parameters via an `all-reduce` collective. On the other hand, for the non-popular μ-batch, frequently-accessed embeddings are updated on GPUs, while the remainder is updated on the CPU's main memory using DMA.

Sources of benefits: The benefits of Hotline arise from (1) overlapping embedding lookup and communication required for non-popular $\mu-batches$ with GPU-based execution of popular $\mu-batches$, (2) executing all operations, including embedding lookup and update, on GPU HBM using a data and model-aware pipeline scheduler, and (3) using a novel Hotline-accelerator to pipeline segregation and parameter gathering for a single mini-batch without stalling the GPU devices. Roofline analysis showed a theoretical 3× gain from GPU HBM for embedding lookups over Intel's Optimized Embedding Bag operator [16] for DDR4 memory. In practice, Hotline achieves nearly a 2.2× improvement over Intel Optimized DLRM.

A. Model Updates with Hotline

Click-through rate (CTR) is modelled as a binary classification problem with binary cross-entropy (BCE) loss. A mini-batch (M) with n inputs is $= \{m_1, m_2, \ldots, m_n\}$, where m_i is a single input. Across mini-batch M, the BCE loss (L) for each input $f(m_i)$ is represented as Equation 1, where y_i is the target and p_i is the predicted probability for the i^{th} input.

$$f(m_i) = y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \tag{1}$$

Thus the BCE loss (L) for M is represented as Equation 2.

$$L_{baseline} = \sum_{i=1}^{n} f(m_i) = \sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \tag{2}$$

As shown in Figure 12, Hotline splits mini-batch M into two μ-batches: popular and non-popular. The popular μ-batch is represented as $\mathcal{O} = \{o_1, o_2, \ldots, o_l\}$. Similarly, the non-popular μ-batch is represented as $\mathcal{X} = \{x_1, x_2, \ldots, x_k\}$. Often $l > k$ because the popular inputs are a larger portion of the dataset. The two μ-batches are mutually exclusive, i.e., without overlapping inputs. We express this using Equation 3.

$$\mathcal{O} \cup \mathcal{X} = M \qquad \mathcal{O} \cap \mathcal{X} = \emptyset \tag{3}$$

The BCE loss for $\mathcal{O}$ and $\mathcal{X}$ is denoted by Equation 4:

$$\begin{aligned} L_{popular} &= \sum_{i=1}^{l} f(o_i) \\ L_{non-popular} &= \sum_{i=1}^{k} f(x_i) \end{aligned} \tag{4}$$

[1] Prior work, such as FAE [10], have coherence overheads. These overheads stem from the requirement of synchronizing embeddings between CPUs and GPUs, as illustrated in Figure 20. These synchronization processes happen at each transition between popular and non-popular mini-batches.

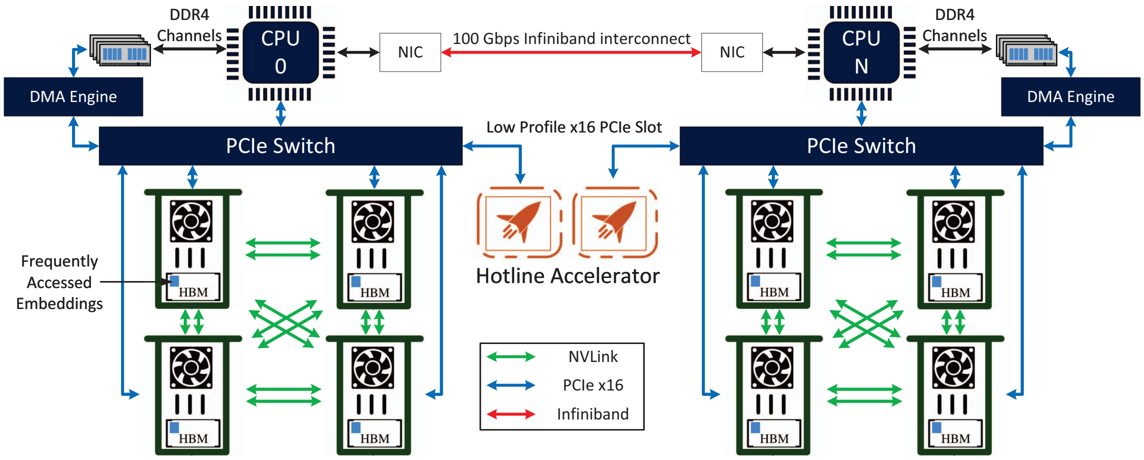


Fig. 10: The Hotline system features an accelerator situated between the CPU and GPU(s), responsible for accessing the main memory to retrieve training inputs and embeddings, which are then efficiently relayed to the GPU(s). In multi-node distributed training, each node utilizes its own Hotline accelerator to oversee and execute parameter aggregation for its mini-batch.

The BCE loss of Hotline is denoted as $L_{hotline} = L_{popular} + L_{non-popular}$. Now, using Equation 3, and Equation 4, we can rewrite Equation 2 as Equation 5.

$$\begin{aligned} L_{baseline} &= \sum_{i=1}^{l} f(o_i) + \sum_{i=1}^{k} f(x_i) = \sum_{i=1}^{n} f(m_i) \\ &= L_{popular} + L_{non-popular} \\ &= L_{hotline} \end{aligned} \tag{5}$$

Therefore, the BCE loss calculated for baseline and Hotline is the same. Consequently, their gradients during backpropagation are also identical. Thus, compared to baseline, Hotline depicts *no loss* in training or testing accuracy.

V. The Hotline Accelerator

Figure 11 shows the block diagram of the Hotline accelerator. We will now describe the micro-architectural details of each component within the Hotline hardware accelerator.

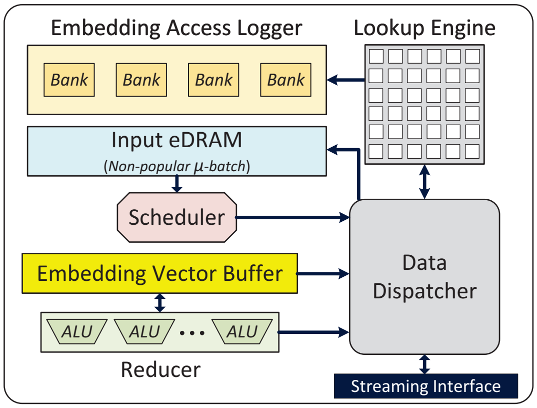


Fig. 11: The components within the Hotline hardware accelerator block. The Hotline accelerator is connected to the low-profile PCIe slot via a Streaming Interface.

A. Data Dispatcher

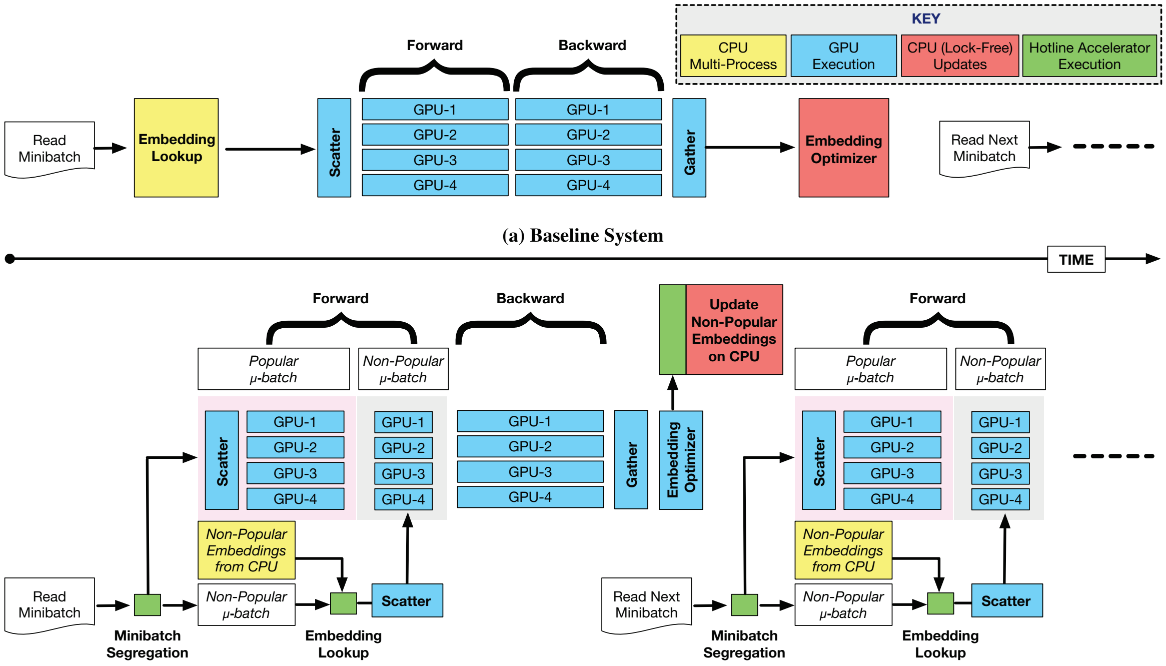
Figure 13 shows the Data Dispatcher block, which includes the *Address Registers* containing the base address of each embedding table in CPU and GPU memory. The *Memory Controller* uses these Address Registers to generate embedding addresses. The *Input Classifier* segregates incoming inputs based on the Embedding Access Logger (EAL) information, distinguishing them as popular or non-popular. While the popular μ-batch executes, the dispatcher sends the non-popular μ-batch from the input eDRAM to the Lookup Engine. Our design shows that a small 2.5 MB of eDRAM can store mini-batches with up to 16K inputs.

The non-popular μ-batch accesses arbitrary embeddings. The memory controller sends a direct memory access (DMA) request to the DMA engine for not-frequently-accessed embeddings and initiates a GPU read memory request for frequently-accessed embeddings. The Reducer block processes working parameters from CPU and GPU memory, reducing multiple embedding rows into a single embedding vector, and then stores it in the *Embedding Vector Buffer*.

B. Embedding Access Logger (EAL)

The Embedding Access Logger (EAL) actively utilizes a counter to track the frequency of access to embedding entries. EAL stores only the indices of embedding entries with valid bits and access counts. Figure 14 depicts the components of EAL, which include a multi-banked Static Random Access Memory (SRAM), a controller, and a queue.

A. Naive Embedding Tracking: Due to a large number of sparse parameters in recommender models, per-entry frequency counters would require gigabytes of on-chip storage. Alternatively, storing the frequencies in CPU/GPU memory



(a) Baseline System

(b) Hotline Acceleration

Fig. 12: The execution pipeline of Hotline involves the accelerator actively classifying a mini-batch into popular and non-popular μ-batches, then scheduling the popular μ-batch onto the GPU(s). Simultaneously, the accelerator gathers the working parameters for the non-popular μ-batch to schedule onto the GPU(s).

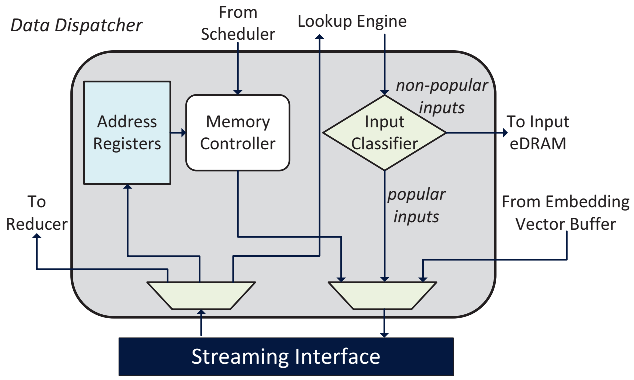


Fig. 13: The Address Registers, Memory Controller, and Input Classifier constitute the Data Dispatcher block.

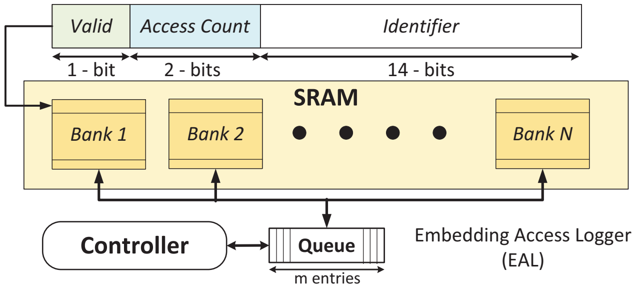


Fig. 14: The Embedding Access Logger (EAL) block consists of Multi-Banked SRAM, the Controller, and the Queue sub-blocks. It tracks frequently accessed embeddings.

would require three accesses: one to obtain the embeddings, one to read the frequency, and another to update it.

B. Efficient Embedding Tracking: The EAL design is motivated by two observations: the size of frequently-accessed embeddings is $\leq$512 MB and their access skews are extremely high. Thus, EAL is designed as a cache-like structure that tracks frequently-accessed embedding indices using a 4 MB SRAM cache with 2 million blocks. EAL uses the Static Re-reference Interval Predictor (SRRIP) replacement policy with a 2-bit Re-reference Predictor Value (RRPV) counter to reduce area overheads. As frequently-accessed embeddings have $>$100× more frequent accesses, a 2-bit RRPV counter (access counter or AC) with insertions at RRPV-1 value captures $>$99% of the frequently-accessed embeddings with 70% tracking capability. Even if a non-frequently-accessed embedding is misclassified as frequently-accessed or vice versa, it has no impact on model fidelity.

To evaluate EAL with one-hot encoded inputs versus multi-hot encoded inputs, we compared the hit rate of one-hot encoded real-world datasets with multi-hot encoded synthetic datasets (Section VII-F4). The hit rate of EAL for the multi-hot encoded datasets decreases by only a maximum of 5%.

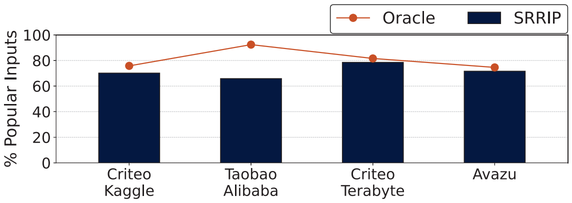


Fig. 15: SRRIP-based tracker as compared to the Oracle LFU scheme. On average, the SRRIP-based tracker can track 90% of the frequently-accessed embeddings.

Figure 15 compares the SRRIP logger to an Oracle logger [2].

C. Multi-Banked SRAM for Parallel Lookup: Hotline enables parallel lookups by dividing the EAL into multiple banks. Figure 16 shows our empirical design space exploration, which reveals the average number of requests issued as the number of banks (n) and input queue size (m) vary. On average, a 512-sized queue with 64 banks allows for 60 parallel requests per iteration without collisions. A controller schedules requests from the lookup engine block to the 512-entry queue. Periodically, the EAL switches to the 'learning' phase to capture changes in popular embeddings.

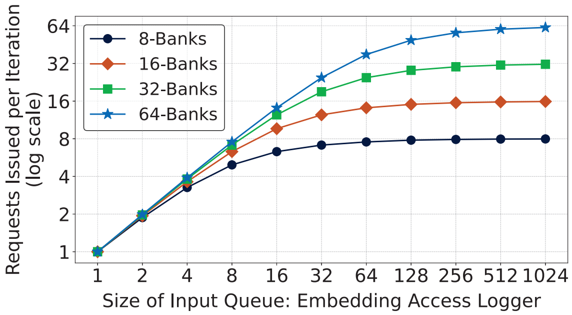


Fig. 16: Impact of Queue Size and Banks on the number of parallel requests per iteration. Embedding Access Logger uses a 512 queue with 64 banks to enable 60 parallel requests.

C. The Lookup Engine

The Lookup Engine is a parallel 2D lookup network that extracts embedding entries from every training input. It can achieve 26× throughput per input if it requires 26 distinct embedding tables. Additionally, the 2D lookup network allows for exploiting parallelism within the mini-batch. During the learning phase, the Lookup engine provides EAL with the indices accessed by each input. The Lookup Engine classifies inputs as popular during the acceleration phase if all embedding indices are within EAL.

A single lookup engine, as shown in Figure 17, contains registers for embedding table numbers, hot embedding index, and a randomizer. The randomizer hashes the (Embedding Index, Embedding Table) tuple to scatter embedding index values across EAL and prevent trashing. A low-latency Fiestal Network implements the randomizer [20].

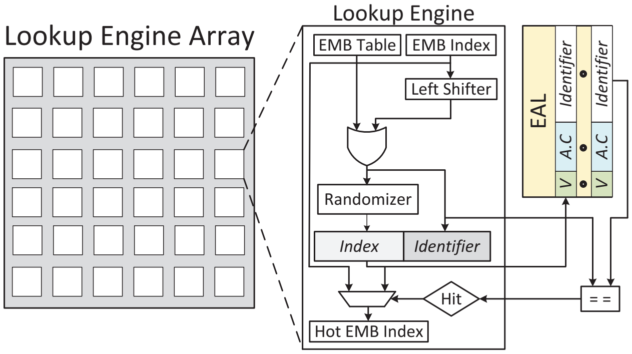


Fig. 17: The Lookup Engine. The lookup engine determines the embedding entries required by an input.

D. The Reducer

The Reducer performs a sparse-length element-wise sum operation using a simple arithmetic unit array. It reduces multiple embedding rows into a single vector through a pooling operation and saves the result in the Embedding Vector Buffer.

E. Instruction Set Architecture

The Hotline accelerator relies on a driver to communicate with the CPU's main memory and GPU devices. The driver interacts with the DMA engine to access not-frequently-accessed embeddings on CPU main memory and GPU devices via a PCIe link. It uses instructions, as listed in Table I, to read/write the necessary data into these devices.

TABLE I: Hotline's Instruction Set

Instruction	Operand 1	Operand 2	Description
dma_rd(op1, op2)	mem start idx	# bytes	DMA read request
dma_wr(op1, op2)	mem start idx	# bytes	DMA write request
v_add(op1, op2)	input vector	emb vec buff	element wise addition
v_mul(op1, op2)	input vector	emb vec buff	element wise dot product
s_wr(op1, op2)	reg idx	base addr	write emb base addr
gpu_rd(op1, op2)	gpu device id	sparse idx	read emb idx from GPU device

VI. Evaluation Methodology

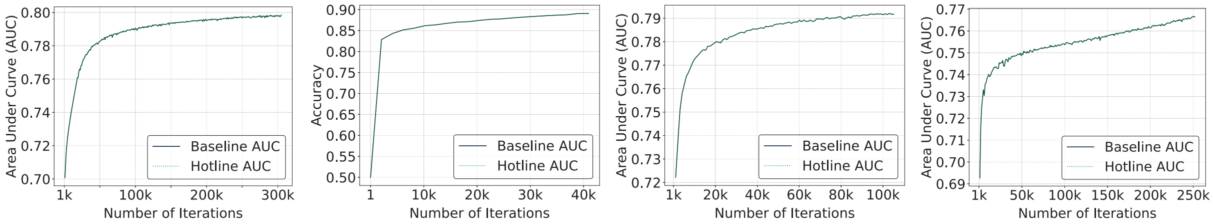
A. Models

Table II presents the specifications of four open-sourced recommender models that were evaluated using Hotline. The models have varying numbers of sparse parameters, ranging from 5.1M for RM1 to 266M for RM3. These models consist of a top and bottom multi-layer perceptron (MLP) with a deep learning attention layer for RM1. On the other hand, RM2 and RM3 have more sparse features and larger embedding tables, making them embedding-dominated models. RM4 has an average-sized dense neural network and sparse embedding tables. Benchmarks such as Deep Learning (DLRM) [6] and Time-based Sequence Models (TBSM) [7] were used to train these models, with TBSM training the RM1 model and DLRM training the RM2, RM3, and RM4 models.

[2] We could use the Least Frequently Used (LFU) replacement policy to understand the access frequency. However, this incurs significant area overheads, as each cache block would require a 24-bit counter (Figure 6 shows embeddings can have up to 10 million accesses).

TABLE II: Recommender Model Architecture and Parameters

Model	Dataset	Time Series	Features		Parameters			Neural Network Configuration			Size (GB)
			Dense	Sparse	Dense	Sparse	Sparse Dim	Bottom MLP	Top MLP	DNN	
RM1	Taobao Alibaba [21]	21	1	3	7.3 k	5.1 M	16	1-16	30-60-1	Attn. Layer	0.3
RM2	Criteo Kaggle [22]	1	13	26	287.5 k	33.8 M	16	13-512-256-64-16	512-256-1	-	2
RM3	Criteo Terabyte [23]	1	13	26	549.1 k	266 M	64	13-512-256-64	512-512-256-1	-	63
RM4	Avazu [24]	1	1	21	281.4 k	9.3 M	16	1-512-256-64-16	512-256-1	-	0.55



(a) Criteo Kaggle

(b) Taobao Alibaba

(c) Criteo Terabyte

(d) Avazu

Fig. 18: The accuracy of Hotline with full-precision training. Hotline maintains exactly identical training fidelity as the baseline.

B. Datasets

We train on four real-world datasets, listed in Table II. Taobao Alibaba [21] is a user behavior dataset for recommendation problems with implicit feedback. Criteo Kaggle [22] dataset contains advertising data and is obtained from the Display Advertising Challenge to capture user preferences by predicting CTR. Criteo Terabyte [23] is the largest publicly available dataset for user click logs. The Avazu [24] dataset is taken from a CTR prediction competition by Kaggle.

C. Software libraries and setup

We configured DLRM and TBSM using Pytorch-1.9 with the *torch.distributed* backend to support scalable distributed training and performance optimizations [25]. To achieve GPU-to-GPU communication for collective operations like gather, scatter, and all-reduce, we used NVIDIA Collective Communication Library (NCCL) [26] on NVLink-2.0. We compared our results with other implementations such as XDL [15], Intel-optimized DLRM [16], and FAE [10]. The XDL-based implementation uses Tensorflow-1.2 [27].

TABLE III: System Specifications

Device	Architecture	Memory	Storage
CPU	Intel Xeon Silver 4116 (2.1 GHz)	192 GB DDR4 (76.8 GB/s)	1.9 TB NVMe SSD
GPU	Nvidia Tesla V100 (1.2 GHz)	16 GB HBM-2.0 (900 GB/s)	-

D. Server Specifications

Table III provides information about the server used for the experiments. The server employs a 24-core Intel Xeon Silver 4116 (2.1 GHz) processor based on Skylake architecture and is equipped with 4 NVIDIA Tesla-V100 GPUs. The communication between the GPUs, Hotline accelerator, and the rest of the system is facilitated via a 16x PCIe Gen3 bus. All experiments are conducted on a single server.

E. Measurements

We measure the model's convergence time using wall clock time. The Verilog RTL architecture of the Hotline accelerator is validated using Synopsys DC at 350 MHz with 45nm technology. Cacti is used to estimate the area/energy of memory components and their access time. The accelerator details can be found in Table IV. The runtime of the accelerator is calculated using the compute and access cycles obtained from Synopsys DC and Cacti through RTL simulation. Additionally, we estimate the time it takes to gather the working parameters using real-system DMA and HBM latencies and incorporate this latency in the pipeline. The end-to-end training time includes the latency for executing the non-popular μ-batch with parameters already available on the GPU. To mitigate HBM contention, we fetch frequently-accessed embeddings from different GPUs in a round-robin fashion, ensuring a balanced memory load on each device.

TABLE IV: Accelerator Specifications

Parameters	Settings	Parameters	Settings
Frequency	350 MHz	EAL size	4 MB
No of Reducer ALU Units	16	No of Lookup Engines	64
Input eDRAM size	2.5 MB	Embedding Vector Buffer	0.5 kB
Total Area	7.01 mm^2	Average Energy	132 mJ

VII. Results and Analysis

A. Training Accuracy

We evaluated the accuracy of Hotline using full-precision DLRM and TBSM model implementations. Figure 18 illustrates the Area Under Curve (AUC) accuracy metric for Kaggle, Terabyte, and Avazu established by MLPerf [28, 29]. We observed that Hotline followed the baseline test and train accuracy and had *no accuracy implications*. This is because Hotline fragments a mini-batch into two μ-batches that continue to update the same embeddings. The baseline implementation and Hotline update these embeddings with

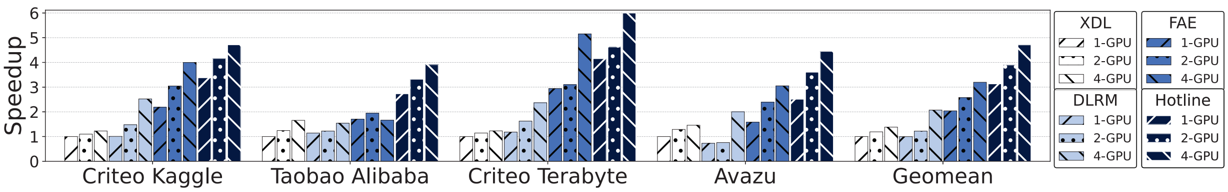


Fig. 19: The performance comparison of Hotline with XDL, Intel optimized DLRM, and FAE implementations (normalized to a 1-GPU XDL). On average, even a 1-GPU Hotline provides 3.1× higher performance than the baseline.

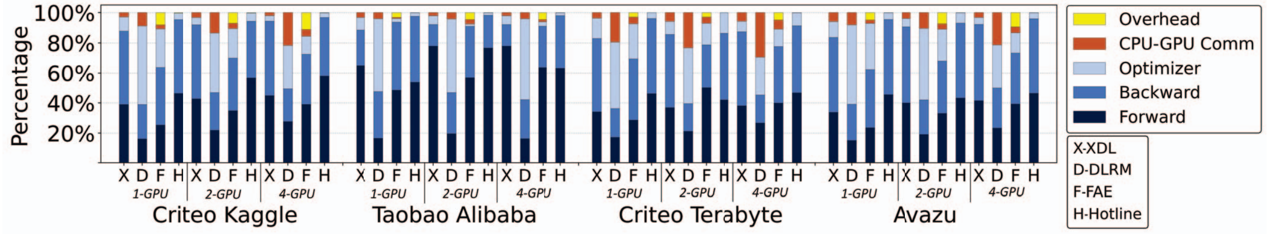


Fig. 20: Latency breakdown of 1, 2, and 4 GPU implementations of software frameworks and Hotline. The overhead of CPU-GPU communication time increases as the number of GPUs scale because of inter-GPU communication.

identical gradients in each mini-batch. Table V also compares the testing accuracy, AUC, and cross-entropy loss across datasets for DLRM (baseline) and Hotline.

TABLE V: Comparison of Accuracy Metrics

Dataset	DLRM			Hotline		
	Accuracy (%)	AUC	Logloss	Accuracy (%)	AUC	Logloss
Criteo Kaggle	78.64	0.798	0.456	78.64	0.798	0.456
Taobao Alibaba	89.11	92.61	0.270	89.11	92.61	0.270
Criteo Terabyte	81.20	0.792	0.421	81.20	0.792	0.421
Avazu	83.61	0.766	0.387	83.61	0.766	0.387

B. Comparison with Hybrid Baseline

1) Performance comparisons: Figure 19 compares Hotline with three state-of-the-art software implementations while varying the number of GPUs. XDL [15] uses a parameter server approach, Intel-Optimized DLRM [16] executes embeddings on the CPU with lock-free updates, and FAE [10] utilizes input popularity with *offline pre-processing* and *CPU-based scheduling without pipelining*. While Hotline is designed to adapt to changing trends in user behaviour (Section III (Challenge 3) and Figure 9), FAE cannot capture such changing trends as it employs a static offline profiler while also incurring a 15% overhead. In contrast, Hotline periodically updates the frequently accessed embeddings with minimal overhead. We use weak scaling to scale mini-batch size with GPUs, and all numbers are normalized to XDL's 1-GPU setup.

Figure 19 shows that Hotline reduces training time as the recommender model is executed on GPU(s). Hotline has a speedup of 3.1× and 3.2× for 1-GPU and 2-GPU setups, and 3.4× for 4-GPU setups, on average across all models and datasets, over XDL. Compared to optimized DLRM, Hotline has a speedup of 3.1×, 3.1×, and 2.2× for 1-GPU, 2-GPU, and 4-GPU implementations, respectively. Hotline also outperforms FAE with a speedup of 1.5×, 1.5×, and 1.4× for the 1-GPU, 2-GPU, and 4-GPU setups, respectively. This is due to efficient runtime scheduling on a massively-parallel Hotline accelerator instead of the CPU.

2) Latency breakdown: Figure 20 demonstrates the latency breakdown of Hotline and three hybrid baselines. The Criteo Kaggle and Terabyte datasets, which are more embedding and memory intensive, comprise high CPU–GPU communication time. Hotline eliminates the CPU-GPU communication time for popular μ-batch being completely executed on GPU. In contrast, for non-popular μ-batch, it hides the parameter gathering under popular μ-batch execution. In the case of the Taobao dataset, which is dominated by the neural network, deep learning execution surpasses the communication time. Overall overhead is shown in Figure 20. This overhead for Hotline includes online profiling and is minimal, primarily because online profiling done at the start of training is not hidden under GPU execution. Still, all subsequent profiling is hidden under GPU execution, significantly reducing overhead. Also, the lookup engine parallelizes the input accesses from EAL for embedding indices. This results in fast online profiling. In our evaluation, we transitioned to the access learning phase twice within a single epoch. Users can specify the frequency at which the learning phase is invoked.

In contrast, our experiments reveal that offline profilers have a 15% additional overhead in training time [10]. Prior work often overlooks this overhead of the static offline profiler in the overall training time [10, 11]. Additionally, FAE incurs coherence overhead from embedding synchronization when switching between popular and non-popular data.

3) Throughput improvements: Figure 21 shows that for a 4-GPU system, Hotline achieves higher throughput than the optimized DLRM baseline, averaging 2.6× more epochs/hour. The throughput of Hotline increases rapidly for larger mini-

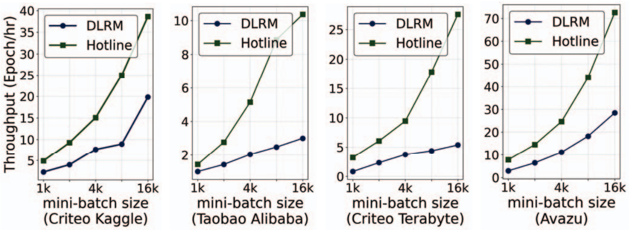


Fig. 21: Training throughput (Epochs/hour) with 4-GPU executions. Hotline achieves, on average, 2.6× more throughput than the optimized DLRM baseline.

batches due to its ability to utilize a larger popular μ-batch, which can be fully executed on GPU, hiding parameter gathering and communication latency for non-popular μ-batches.

C. Comparison: GPU-only Baseline

We compare Hotline against Nvidia's GPU-only baseline, HugeCTR [9]. HugeCTR scales the number of GPUs to fit the entire model using model-parallel training for embeddings and data-parallel training for the neural network. Figure 22 compares Criteo Kaggle and Criteo Terabyte datasets.

HugeCTR can train small models like Criteo Kaggle on a single GPU, so its results are normalized to 1-GPU HugeCTR. However, for large models like Criteo Terabyte, HugeCTR throws an Out of Memory (OOM) error and cannot fit the model within 1 or 2 GPUs, so its results are normalized to 4 GPUs. Hotline eliminates `all-to-all` communication and achieves a speedup of 1.13×.

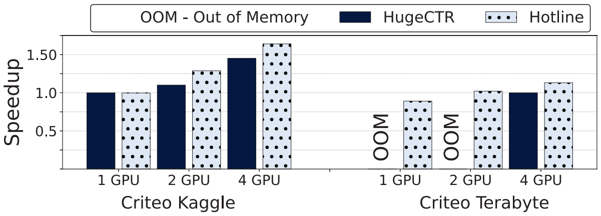


Fig. 22: The speedup of Hotline compared to HugeCTR. Hotline eliminates `all-to-all` communication.

It is unfair to compare Hotline, a hybrid training scheme, to a GPU-only training scheme. Hotline can train even large datasets such as Terabyte with a single GPU. These datasets would otherwise be unable to be trained on a single GPU. The GPU-only mode needs at least four GPUs for such datasets.

D. Comparison: CPU-based Design

Figure 23 compares Hotline to a multi-process CPU-based segregator and scheduler. Using the CPU for mini-batch segregation and working parameter gathering results in GPU stalls as the CPU cannot hide the latency behind popular μ-batch execution. Hotline outperforms this alternative approach, providing significant performance benefits.

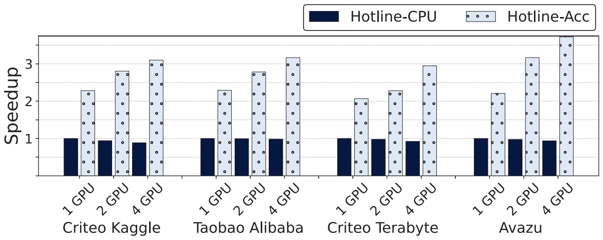


Fig. 23: Hotline speedup normalized to CPU-based Hotline implementation. Hotline provides up to 3.5× higher speedup.

E. Comparison: Lookahead-Based Software Baselines

Software-oriented approaches [14, 30, 31] have explored utilizing skewed embedding access patterns by prefetching future mini-batch embeddings into a GPU-based cache. However, such lookahead-based approaches introduce complexities related to data hazards, cache eviction, and model accuracy. For example, cDLRM [31] utilizes stale embeddings to mitigate data hazards at the expense of reduced model accuracy.

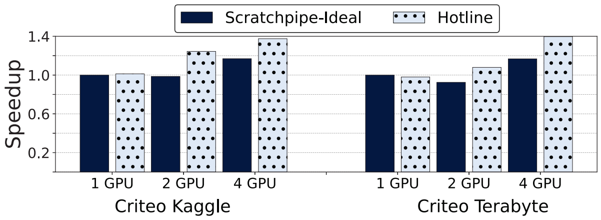


Fig. 24: Speedup of Hotline compared to ScratchPipe-Ideal.

Figure 24 compares Hotline to ScratchPipe [30]. ScratchPipe [30] is not open-source. Thus, the exact implementation is unknown. Due to this, we re-implemented ScratchPipe with optimistic assumptions and relaxed the stringent read-after-write (RAW) dependencies for model updates. We anticipate a more substantial speedup for Hotline if ScratchPipe adheres strictly to RAW dependencies. ScratchPipe-Ideal represents an ideal implementation of ScratchPipe [30] with relaxed read-after-write (RAW) dependencies. It performs similarly to Hotline for a single GPU. However, as the number of GPUs increases, ScratchPipe-Ideal encounters scalability issues due to `all-to-all` communication. In contrast, Hotline achieves an average speedup of 1.2× for 4 GPUs.

F. Sensitivity Studies

1) Varying Popular/Non-Popular μ-batch Ratio: We explored various popular to non-popular μ-batch ratios using a synthetic dataset. Real-world datasets consistently exhibited an average ratio of `3:1` for popular to non-popular μ-batches, each with 512 MB of frequently-accessed embeddings. Figure 25 illustrates Hotline's ability to effectively conceal embedding gather latency even with a `3:7` popular to non-popular μ-batch ratio. Such low ratios are rare in real-world datasets, typically following a Zipfian distribution [10, 32].

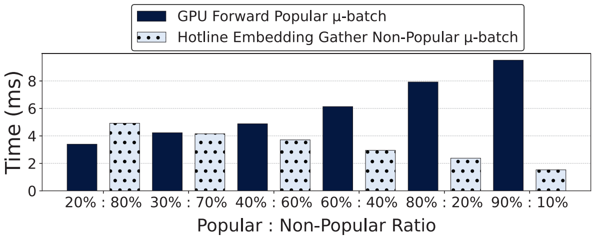


Fig. 25: Effect of varying the ratio of popular to non-popular μ-batches. Hotline effectively conceals embedding gather latency even with a `3:7` popular to non-popular μ-batch ratio.

2) Varying Mini-batch Size: Hotline benefits increase with larger mini-batch sizes, as shown in Figure 26. The scheduler issues fewer input-dispatch commands, and larger mini-batches provide more parallelism opportunities for GPUs.

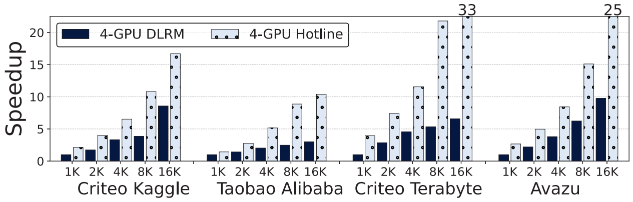


Fig. 26: Hotline speedup with varying mini-batch sizes. The benefits of Hotline increase with larger mini-batch sizes.

3) Varying EAL Size: Figure 27 shows popular inputs captured with varying the EAL size. For highly skewed datasets like Criteo and Avazu, a 2MB logger is sufficient to capture frequently-accessed indices. However, EAL sizes above 4MB offer diminishing returns for the less skewed Taobao dataset.

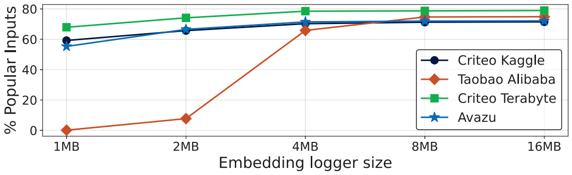
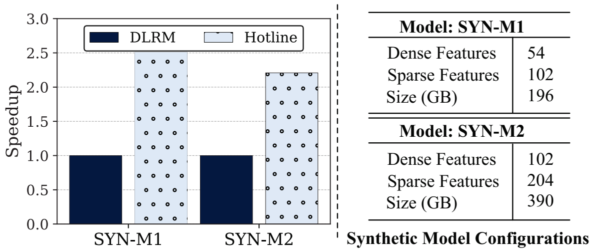


Fig. 27: EAL design space exploration shows a 4MB SRAM sufficiently captures the frequently used embedding indices.

4) Varying Model Size: We generated synthetic models and datasets with multi-hot encoded inputs to understand the efficacy of Hotline to model size increase. Multi-hot encoded lookups influence the frequency of popular μ-batches. However, due to the heavy-tailed distribution of accesses, over 75% of these inputs are popular. As shown in Section VII-F1, this high proportion of popular inputs adequately conceals the parameter gathering latency for non-popular μ-batches.

Figure 28 shows the performance of Hotline across two synthetic models and datasets. Our experiments show that the benefits of Hotline are sustained even for larger models. As the model size increases, the sparse features increase from 102 to 204, and the performance gains decrease from 2.5x to 2.2x. This decrease can be attributed to the Hotline accelerator's parallel lookup engine size remaining the same at 64. With more sparse features, the Hotline accelerator requires more cycles to segregate the input mini-batch, given the fixed size of the parallel lookup engine.



Model: SYN-M1	
Dense Features	54
Sparse Features	102
Size (GB)	196

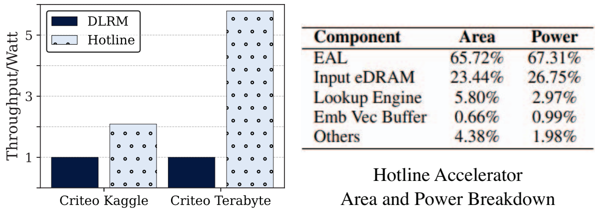
Model: SYN-M2	
Dense Features	102
Sparse Features	204
Size (GB)	390

Synthetic Model Configurations

Fig. 28: Performance of Hotline versus Intel-Optimized DLRM across synthetic models for a 4-GPU system. The benefits of Hotline are sustained even for larger models.

G. Comparison: Performance/Watt, Area, and Power

Figure 29 shows the Throughput/Watt improvement and area/power consumption of Hotline components. The EAL consumes the most power and area due to its SRAM structure. Despite the 7.01 mm^2 area overhead and extra power consumption, Hotline's performance benefits outweigh the power overheads, providing 3.9× performance/Watt improvement .



Component	Area	Power
EAL	65.72%	67.31%
Input eDRAM	23.44%	26.75%
Lookup Engine	5.80%	2.97%
Emb Vec Buffer	0.66%	0.99%
Others	4.38%	1.98%

Hotline Accelerator
Area and Power Breakdown

Fig. 29: Throughput/Watt comparison with Area and Power breakdown of the Hotline accelerator.

H. Multi Node Distributed Training

In the multi-node setup, we evaluated large synthetic models (SYN-M1 and SYN-M2) described in Section VII-F4. We compared Hotline to HugeCTR across configurations of 1, 2, and 4 nodes, with mini-batches of 4k, 8k, and 16k. SYN-M1 (196GB) fits only in a 4-node setup, while SYN-M2 (390GB) exceeds the capacity of 4 nodes (16 NVIDIA V100 GPUs).

Figure 30 demonstrates Hotline achieving a 1.89× speedup on 4 nodes by eliminating `all-to-all` communication, which constitutes over 50% of total training time. Scaling from one to two nodes encounters challenges due to `all-reduce` synchronization over InfiniBand and CPU-mediated parameter gathering from other nodes. Nevertheless, Hotline's efficiency enables training large models on a single GPU without increasing the GPU count.

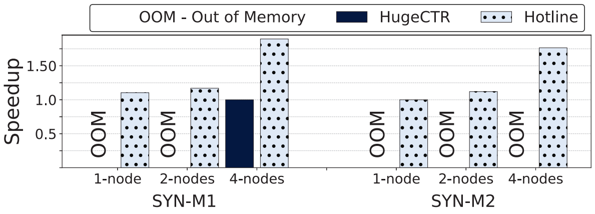


Fig. 30: Hotline scalability for multi-node setting with large synthetic models. The speedup is benchmarked with 4-node HugeCTR for SYN-M1 and 1-node Hotline for SYN-M2.

VIII. Related Work

Recommendation models and designs: Prior work has primarily focused on optimizing the inference phase of recommendation models, as shown in [33–40]. However, some solutions have also been proposed for training optimizations and acceleration, such as [32, 41–43]. These solutions, however, do not maximize throughput by effectively utilizing memory and bandwidth in a distributed GPU system. Recently, NEO [44] was introduced, which leverages 4-D parallelism for recommendation model training. While NEO can further benefit from embedding placement and patterns in training data, it is orthogonal to Hotline.

Embedding parameter placement: Prior methods [10, 11] rely on offline profiling and static embedding placement based on training data skew. In contrast, Hotline dynamically adapts to changing data patterns without such overheads. Recent works [1, 4] explore alternative embedding table placements but lack preprocessing to reduce communication overheads. Bandana [45] suggests storing embedding tables in non-volatile memory with DRAM caching. Other approaches [32, 41, 42, 46] accelerate near-memory processing but lack support for distributed training with GPUs. Prior work [47, 48] performs embedding placement across GPU-device for GPU-only training, while Hotline targets a two-tier memory hierarchy for hybrid training.

Mitigating memory intensive training: Prior work has focused on optimizing the model using mixed-precision training or eliminating rare categorical variables to reduce embedding table size [49, 50]. However, changing the data representation or embedding tables requires accuracy re-validation. Compression and sparsity have also been used to reduce model memory footprint [51–55]. In contrast, Hotline performs full-precision training without the overheads of compression/decompression and sparse operations. It only leverages access skew and is independent of these techniques.

Embedding Representation: Previous research has explored various methods to represent categorical features within limited memory, aiming to accommodate multiple feature values with a restricted number of embeddings. The hashing trick [56] applies a simple hash function to constrain feature embeddings. Compositional Embeddings [41] leverages complementary partitions of categorical features, utilizing multiple smaller embedding tables and combining embeddings from each table. ROBE [57] accesses contiguous blocks in shared memory for enhanced memory access. Unified Embeddings [58] consolidates all categorical features within a single embedding table, allowing for collisions of feature values within and across features. DHE [59] employs an orthogonal approach, representing feature values using MLPs and encoders instead of embeddings. Hotline can be applied atop any embedding representation technique.

Embedding Prefetching: Previous studies [14, 30, 31] have investigated prefetching embeddings into a GPU-based cache for the next mini-batch of training. However, this prefetching-based approach introduces complexities such as data hazards, complex cache eviction policies, and asynchronous training with limited scalability. In contrast, Hotline avoids these complexities through pipeline scheduling within a minibatch.

Machine learning accelerators: There are proposals for accelerators designed to execute the compute portion of deep learning models [60–64], including some for collaborative filtering-based recommender models [43, 65, 66]. However, Hotline does not aim to design a specialized architecture for optimizing the computing of deep learning-based recommender models. Hotline accelerator can pipeline and potentially enhance these existing accelerators.

IX. Conclusions

This paper proposes Hotline, a heterogeneous acceleration pipeline to address memory and bandwidth constraints in recommendation models. Hotline leverages the insight that only a few embedding table entries are popular and frequently accessed. It sends inputs directly to the GPUs, which use frequently-accessed embeddings. It retrieves the required embeddings for the remaining inputs while the GPUs process popular inputs. Hotline uses a novel accelerator to dynamically segregate and dispatch the data, hiding the data transfer latency behind the GPU execution of popular inputs. Our experiments on real-world datasets and models show that Hotline reduces average training time by 2.2× compared to Intel-optimized CPU-GPU DLRM baseline.

Acknowledgements

This project is part of the *STAR Lab* at The University of British Columbia (UBC). We thank the entire Advanced Research Computing Center team at UBC [67]. We also thank the anonymous reviewers from ISCA 2024 for their invaluable feedback. Muhammad Adnan's Ph.D. is supported by Intel Transformation Server Architecture (TSA) and the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number RGPIN-2019-05059] Grants. The views and conclusions contained herein are those of the authors. They should not be interpreted as representing the official policies or endorsements, expressed or implied, of NSERC, the Canadian Government, Georgia Tech, or The University of British Columbia.

References

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

[2] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. *Embedding-Based Retrieval in Facebook Search*, page 2553–2561. Association for Computing Machinery, New York, NY, USA, 2020.

[3] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 1042–1057, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450386104. doi: 10.1145/3470496.3533044. URL https://doi.org/10.1145/3470496.3533044.

[4] 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.

[5] 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. ISBN 9781450386104. doi: 10.1145/3470496.3533727. URL https://doi.org/10.1145/3470496.3533727.

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

[7] T. Ishkhanov, M. Naumov, X. Chen, Y. Zhu, Y. Zhong, A. G. Azzolini, C. Sun, F. Jiang, A. Malevich, and L. Xiong. Time-based Sequence Model for Personalization and Recommendation Systems. *CoRR*, abs/2008.11922, 2020.

[8] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, unjie Qian, Wencong Xiao, and Fan Yang. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '19, page 947–960, USA, 2019. USENIX Association.

[9] NVIDIA Merlin: HugeCTR. https://github.com/NVIDIA-Merlin/HugeCTR.

[10] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant Nair. Accelerating Recommendation System Trainingby Leveraging Popular Choices. In *VLDB*, 2022.

[11] 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, 2022.

[12] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and Sangheon Pack. WAVE: Popularity-based and collaborative in-network caching for content-oriented networks. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 316–321, 2012.

[13] Fragkiskos Papadopoulos, Maksim Kitsak, M. A. Serrano, Marian Boguna, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489 (7417):537–40, Sep 27 2012.

[14] Saurabh Agarwal, Ziyi Zhang, and Shivaram Venkataraman. Bagpipe: Accelerating deep recommendation model training, 2022. URL https://arxiv.org/abs/2202.12429.

[15] Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, Liqin Zhao, Zhi Wang, Peng Sun, Yu Zhang, Di Zhang, Jinhui Li, Jian Xu, Xiaoqiang Zhu, and Kun Gai. XDL: An Industrial Deep Learning Framework for High-Dimensional Sparse Data. DLP-KDD '19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] Dhiraj Kalamkar, Evangelos Georganas, Sudarshan Srinivasan, Jianping Chen, Mikhail Shiryaev, and Alexander Heinecke. Optimizing Deep Learning Recommender Systems Training on CPU Cluster Architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.

[17] 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, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. ISBN 9781450349130. doi: 10.1145/3038912.3052569. URL https://doi.org/10.1145/3038912.3052569.

[18] Nvidia. Nvlink, . https://www.nvidia.com/en-us/data-center/nvlink/.

[19] meta. Meta recommender model training on ZionEX devices. https://www.infoq.com/news/2021/05/facebook-zionex-training/.

[20] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[21] Alibaba. User Behavior Data from Taobao for Recommendation. https://tianchi.aliyun.com/dataset/dataDetail?dataId=649userId=1.

[22] CriteoLabs. Criteo Display Ad Challenge, . https://www.kaggle.com/c/criteo-display-ad-challenge.

[23] CriteoLabs. Terabyte Click Logs, . https://labs.criteo.com/2013/12/download-terabyte-click-logs.

[24] Kaggle. Avazu mobile ads CTR. https://www.kaggle.com/c/avazu-ctr-prediction.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[26] Nvidia. NVIDIA Collective Communications Library (NCCL). https://docs.nvidia.com/deeplearning/nccl/index.html, .

[27] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2015. URL http://download.tensorflow.org/paper/whitepaper2015.pdf.

[28] MLPerf Benchmarks. https://mlcommons.org/en/training-normal-10/.

[29] Carole-Jean Wu, Robin Burke, Ed H. Chi, Joseph Konstan, Julian McAuley, Yves Raimond, and Hao Zhang. Developing a Recommendation Benchmark for MLPerf Training and Inference, 2020.

[30] Youngeun Kwon and Minsoo Rhu. Training personalized recommendation systems from (gpu) scratch: Look forward not backwards. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 860–873, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450386104. doi: 10.1145/3470496.3527386. URL https://doi.org/10.1145/3470496.3527386.

[31] Keshav Balasubramanian, Abdulla Alshabanah, Joshua D Choe, and Murali Annavaram. Cdlrm: Look ahead caching for scalable training of recommendation models. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 263–272, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474246. URL https://doi.org/10.1145/3460231.3474246.

[32] A. Ginart, M. Naumov, D. Mudigere, Jiyan Yang, and J. Zou. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *ArXiv*, abs/1909.11810, 2019.

[33] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. Centaur: A Chiplet-based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations. *arXiv preprint arXiv:2005.05968*, 2020.

[34] 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. *arXiv preprint arXiv:2001.02772*, 2020.

[35] Nvidia. Accelerating wide deep recommender inference on gpus, 2017. https://developer.nvidia.com/blog/accelerating-wide-deep-recommender-inference-on-gpus/.

[36] 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.

[37] U. Gupta, C. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. 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.

[38] Haojie Ye, Sanketh Vedula, Yuhan Chen, Yichen Yang, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. Grace: A scalable graph-based approach to accelerating recommendation model inference. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 282–301, 2023.

[39] Daniar H Kurniawan, Ruipu Wang, Kahfi S Zulkifli, Fandi A Wiranata, John Bent, Ymir Vigfusson, and Haryadi S Gunawi. Evstore: Storage and caching capabilities for scaling embedding tables in deep recom-

mendation systems. 2023.

[40] Samuel Hsia, Udit Gupta, Bilge Acun, Newsha Ardalani, Pan Zhong, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Mp-rec: Hardware-software co-design to enable multi-path recommendation. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 449–465, 2023.

[41] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. *Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems*, page 165–175. Association for Computing Machinery, New York, NY, USA, 2020.

[42] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C. Wu, M. Hempstead, and X. Zhang. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 790–803, 2020.

[43] Jongse Park, Hardik Sharma, Divya Mahajan, Joon Kyung Kim, Preston Olds, and Hadi Esmaeilzadeh. Scale-Out Acceleration for Machine Learnng. October 2017.

[44] 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. ISBN 9781450386104. doi: 10.1145/3470496.3533727. URL https://doi.org/10.1145/3470496.3533727.

[45] 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. *Proceedings of Machine Learning and Systems*, 1:40–52, 2019.

[46] 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, 2021.

[47] 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. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_atSgd9Np52.

[48] 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*, pages 4461–4471, 2022.

[49] Mengdi Huang Nvidia Inc. Vinh Nguyen, Tomasz Grel. Optimizing the Deep Learning Recommendation Model on NVIDIA GPUs. https://developer.nvidia.com/blog/optimizing-dlrm-on-nvidia-gpus.

[50] Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. Mixed-Precision Embedding Using a Cache, 2020.

[51] Yang Sun, Fajie Yuan, Min Yang, Guoao Wei, Zhou Zhao, and Duo Liu. A Generic Network Compression Framework for Sequential Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1299–1308, New York, NY, USA, 2020. Association for Computing Machinery.

[52] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko. Gist: Efficient Data Encoding for Deep Neural Network Training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789, 2018.

[53] Xiaorui Wu, Hong Xu, Honglin Zhang, Huaming Chen, and Jian Wang. Saec: similarity-aware embedding compression in recommendation systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 82–89, 2020.

[54] Jeremy Fowers, Kalin Ovtcharov, Karin Strauss, Eric

Chung, and Greg Stitt. A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication. In *International Symposium on Field-Programmable Custom Computing Machines*. IEEE, May .

[55] Zheng Wang, Yuke Wang, Boyuan Feng, Dheevatsa Mudigere, Bharath Muthiah, and Yufei Ding. El-rec: efficient large-scale recommendation model training via tensor-train embedding table. In *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1007–1020. IEEE Computer Society, 2022.

[56] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015.

[57] Aditya Desai, Li Chou, and Anshumali Shrivastava. Random offset block embedding (robe) for compressed embedding tables in deep learning recommendation systems. *Proceedings of Machine Learning and Systems*, 4: 762–778, 2022.

[58] Benjamin Coleman, Wang-Cheng Kang, Matthew Fahrbach, Ruoxi Wang, Lichan Hong, Ed Chi, and Derek Cheng. Unified embedding: Battle-tested feature representations for web-scale ml systems. *Advances in Neural Information Processing Systems*, 36, 2024.

[59] 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 & Data Mining*, pages 840–850, 2021.

[60] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12, New York, NY, USA, 2017. Association for Computing Machinery.

[61] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Misra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *MICRO*, 2016.

[62] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Wei, and D. Brooks. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278, June 2016.

[63] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ISCA*, 2016.

[64] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, , Adrian Caulfield, Todd Massengill, Ming Liu, Mahdi Ghandi, Daniel Lo, Steve Reinhardt, Shlomi Alkalay, Hari Angepat, Derek Chiou, Alessandro Forin, Doug Burger, Lisa Woods, Gabriel Weisz, Michael Haselman, and Dan Zhang. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro*, 38:8–20, March 2018.

[65] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kim, and Hadi Esmaeilzadeh. TABLA: A unified template-based framework for accelerating statistical machine learning. March 2016.

[66] Divya Mahajan, Joon Kyung Kim, Jacob Sacks, Adel Ardalan, Arun Kumar, and Hadi Esmaeilzadeh. In-rdbms hardware acceleration of advanced analytics. *Proc. VLDB Endow.*, 11(11):1317–1331, July 2018. ISSN 2150-8097. doi: 10.14778/3236187.3236188. URL https://doi.org/10.14778/3236187.3236188.

[67] UBC Advanced Research Computing, "UBC ARC Sockeye." UBC Advanced Research Computing, 2019, doi: 10.14288/SOCKEYE.