Software-Hardware Co-design of Heterogeneous SmartNIC System for Recommendation Model Inference and Training

Anqi Guo
Boston University
University of Rochester
anqiguo@bu.edu

Yuchen Hao
Meta Platforms
haoyc@meta.com

Chunshu Wu
Boston University
happycwu@bu.edu

Pouya Haghi
Boston University
haghi@bu.edu

Zhenyu Pan
University of Rochester
zhenyupan@rochester.edu

Min Si
Meta Platforms
msi@meta.com

Dingwen Tao
Indiana University
ditao@iu.edu

Ang Li
Pacific Northwest National Laboratory
ang.li@pnnl.gov

Martin Herbordt
Boston University
herbordt@bu.edu

Tong Geng
University of Rochester
tong.geng@rochester.edu

ABSTRACT

Deep Learning Recommendation Models (DLRMs) are important applications in various domains and have evolved into one of the largest and most important machine learning applications. With their trillions of parameters necessarily exceeding the high bandwidth memory (HBM) capacity of GPUs, ever more massive DLRMs require large-scale multi-node systems for distributed training and inference. However, these all suffer from the all-to-all communication bottleneck, which limits scalability.

SmartNICs couple computation and communication capabilities to provide powerful network-facing heterogeneous devices that reduce communication overhead. There has not, however, been a distributed system design that fully leverages SmartNIC resources to address scalability of DLRMs.

We propose a software-hardware co-design of a heterogeneous SmartNIC system that overcomes the communication bottleneck of distributed DLRMs, mitigates the pressure on memory bandwidth, and improves computation efficiency. We provide a set of SmartNIC designs of cache systems (including local cache and remote cache) and SmartNIC computation kernels that reduce data movement, relieve memory lookup intensity, and improve the GPU's computation efficiency. In addition, we propose a graph algorithm that improves the data locality of queries within batches and optimizes the overall system performance with higher data reuse. Our evaluation shows that the system achieves 2.1× latency speedup for inference and 1.6× throughput speedup for training.

CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**; **Neural networks**.

KEYWORDS

Recommendation System, SmartNIC, Heterogeneous System

ACM Reference Format:
Anqi Guo, Yuchen Hao, Chunshu Wu, Pouya Haghi, Zhenyu Pan, Min Si, Dingwen Tao, Ang Li, Martin Herbordt, and Tong Geng. 2023. Software-Hardware Co-design of Heterogeneous SmartNIC System for Recommendation Model Inference and Training. In *2023 International Conference on Supercomputing (ICS '23), June 21–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3577193.3593724

1 INTRODUCTION

Personalized Recommendation systems (Resys) are a widely used in applications providing online services such as product recommendations, video and music recommendations, and search services [5, 8, 11, 14, 27, 30, 36]. As recommendation prediction requirements and datasets have grown, deep learning recommendation models (DLRMs) [28] have shown substantial advantages in providing ranking and click through rate (CTR) predictions.

The size of DLRMs is significantly larger than traditional deep neural networks due to their data-intensive embedding operators that require hundreds of Gigabytes or even Terabytes of storage. The model size far surpasses the likely HBM capacity of accelerators. Moreover, the growth of the accelerators' HBM is not keeping up with the ever-growing DLRM size, as shown in Figure 4 [26, 33]. Therefore, high performance DLRM inference and training require large-scale, multi-node systems. As a result, the scalability issue caused by the communication bottleneck fundamentally hinders DLRM advances.

DLRMs pose three challenges to large-scale distributed systems: (1) *Communication bottleneck.* Strategies like data-parallelism are infeasible because the replications of the model are too large to fit into the accelerator's HBM. A hybrid model and data parallelism is often used to deal separately with massive embedding tables and smaller MLPs (Figure 1). The embedding operators are partitioned and distributed to each node and use all-to-all communication to exchange each share of the embedding tables (EMT). The all-to-all communication incurs a massive amount of data exchange resulting in communication as the bottleneck of the entire application. The exponential growth of communication volume indicates that this critical path will worsen. (2) *Memory bandwidth challenge.* DLRMs contain up to trillions of parameters and consume up to terabytes of memory. The embedding operators require high memory bandwidth and frequent access to the embedding tables. (3) *Computation efficiency challenge.* Compared with other machine learning models, DLRMs exhibit lower arithmetic intensity with irregular computations like data reshaping, flattening, and transposing. These irregular operations are primarily memory copies and make GPU's computation less efficient.

Advanced network interface cards known as SmartNICs have emerged to mitigate network communication challenges in scale-out data centers. Moreover, SmartNICs with computation support are particularly useful for domain-specific computation such as machine learning and streaming data analytics [3, 16, 17, 29, 39, 40]. As SmartNICs continue to advance in power, the combination of compute and communication support, together with their placement in the node (network facing), point to their use in overcoming the scalability challenges in DLRM training and inference.

Simply adding SmartNICs to a distributed system, however, only addresses point-to-point communication latency. Currently there is no distributed system design that leverages SmartNIC resources to overcome the other challenges: communication bottleneck, memory bandwidth pressure, improving computational efficiency. Existing work [20, 33, 35] uses software solutions targeting the communication bottleneck by reducing the embedding table size, or the communication volume of all-to-all and all-reduce collection. However, these approaches have limited benefit, and the software solution can not fundamentally resolve the performance bottleneck. Others [6, 38] introduce storage technologies to optimize the embedding operator's performance. But memory bandwidth and latency can't catch up with GPU's HBM, and, as the model size grows, memory bandwidth could become another bottleneck. Current GPU clusters used for DLRM [1, 26, 32] suffer from large communication volumes and frequent communication bottlenecks.

In this paper, we introduce a software-hardware co-design of a heterogeneous SmartNIC system for scalable DLRM inference and training that overcomes communication bottlenecks, mitigates memory bandwidth pressure, and improves computation efficiency. A set of SmartNIC designs of cache systems (including local cache and remote cache) and SmartNIC computation kernels exploits the locality of DLRMs to reduce data movement, relieve memory access intensity, and improve GPUs' computation efficiency.

Figure 2 illustrates the techniques used. **(1) Remote Cache.** The large volume and intense all-to-all communication primarily contribute to the communication bottleneck of distributed DLRM systems. The remote cache on the SmartNIC buffers frequently used remote embedding lookup results, reducing communication workloads and alleviating both networking and memory bandwidth pressure. **(2) Local Cache.** The local cache on the SmartNIC caches the popular local embedding tables allowing direct retrieval of embedding lookup results from the SmartNIC instead of interrupting the GPU. This vastly reduces the memory bandwidth burden on the GPUs' HBMs, improving overall node memory bandwidth. **(3) SmartNIC computation.** The SmartNIC's kernels for irregular computation complement the system nodes' computation resources, improving GPU efficiency by offloading irregular computations, and minimizing GPU kernel launch overhead and hardware usage. In additional, the computation kernels reduce gradient updates in backward propagation and decrease the workload of backward all-to-all communication.

We also introduce a **graph algorithm** that enhances the data locality of DLRM batches by clustering similar samples. More data reuse reduces embedding lookup requests and communication volume, increases cache hit rate, and eases system memory bandwidth pressure. High data locality batches also benefit GPU computation efficiency. This set of techniques works together to produce a synergistic effect, resulting in an outcome greater than the sum of their individual contributions.

To summarize, the contributions of this work include:

- A scalable software-hardware co-design for heterogeneous SmartNIC systems for both forward and backward propagation of DLRMs.
- A set of techniques for SmartNIC design that overcomes the communication bottleneck of distributed DLRMs, mitigates memory bandwidth pressure, and improves computation efficiency. A graph algorithm improves the data locality of batches and optimizes overall system performance with high data reuse.
- Evaluation results showing that heterogeneous SmartNIC systems can achieve 2.1× latency speedup for inference and 1.6× throughput speedup for training.

The remainder of the paper is organized as follows. Section 2 provides DLRM background and the paper's motivation. Section 3 presents details of the graph algorithm. Section 4 introduces the cache system and computation kernels for forward and backward propagation. Results from experiments are presented in Section 5. Section 6 discusses related work; concluding remarks are given in Section 7.

2 DLRM BACKGROUND AND MOTIVATION

2.1 Deep Learning Recommendation Model

Figure 5 gives an overview of DLRMs. DLRMs have two types of inputs, dense features and sparse features, and predict the probability that a user would interact with an item; this is referred to as the Click Through Rate (CTR). Dense features contain continuous data like a user's age or the current time. Sparse features are categorical features such as posts, pages, or items. These categorical features are represented using IDs, which map to embedding vectors from

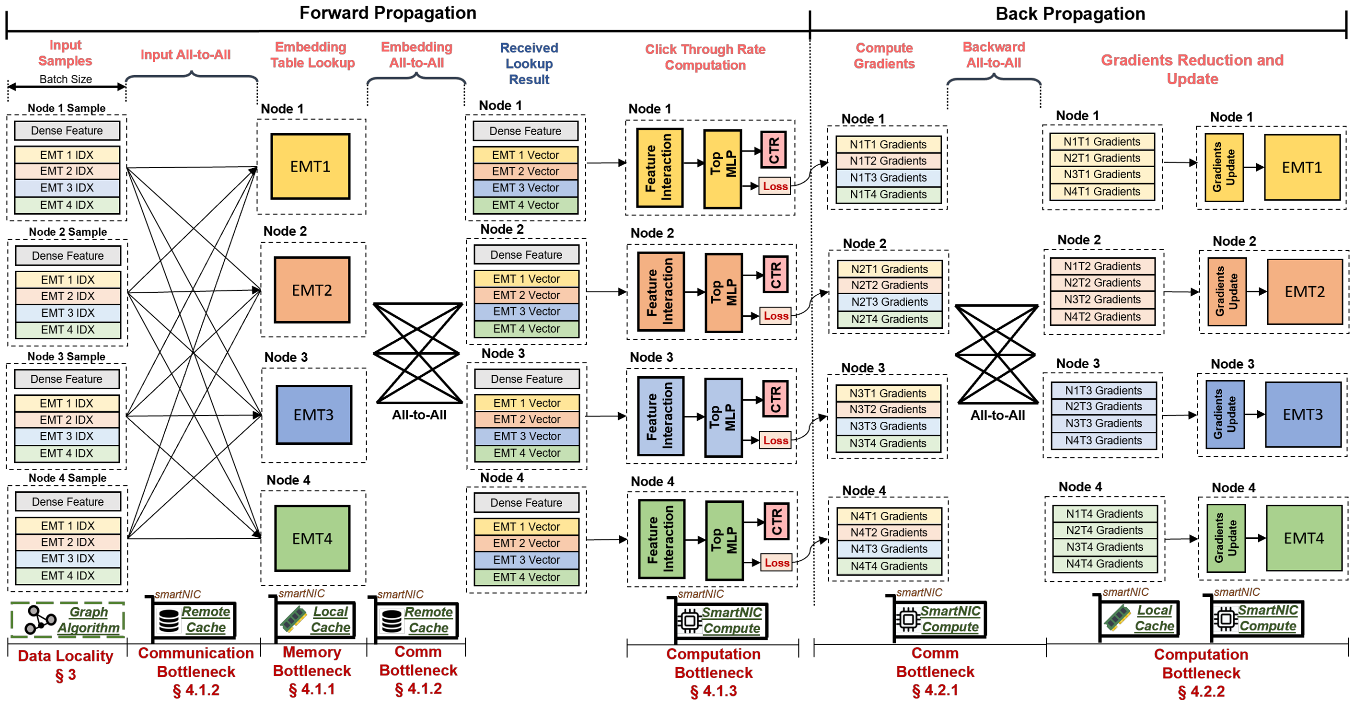


Figure 1: Deep Learning Recommendation Model Workflow Overview (EMT: Embedding Table, MLP: Multilayer Perceptron, CTR: Click Through Rate(Prediction), N1T1 Gradients: Computed gradients of Embedding Table 1 from Node 1) The heterogeneous SmartNIC system targets the memory bottleneck, communication bottleneck, computation bottleneck of forward propagation and backward propagation (Section 4). A graph algorithm improves the batch data locality (Section 3).

DLRM Challenge	Cache System		SmartNIC Computation	Graph Algorithm
	Remote Cache	Local Cache		
Communication	+++		++	+
Memory	++	++		+
Computation			+	+

Figure 2: SmartNIC Design and DLRM Challenges

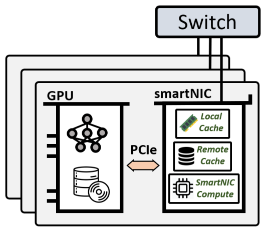


Figure 3: Heterogeneous SmartNIC System Overview

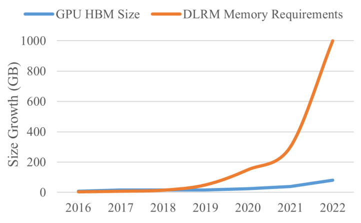


Figure 4: DLRM memory capacity requirements and GPU HBM growth

their corresponding embedding tables (EMT). Modern DLRMs consume thousands of categorical features with thousands of EMTs; these are handled by embedding operators. Feature interaction combines the output of the bottom multilayer perception (MLP) and feeds it into the top MLP for CTR prediction.

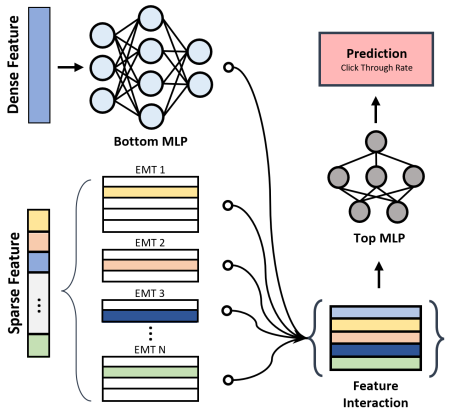


Figure 5: Deep Learning Recommendation Model.

2.2 Distributed DLRM System Challenges

The embedding operators are partitioned and distributed to the system since their size largely exceeds a single accelerator's HBM size. Samples are required to access each partition of embedding parameters before performing the next stage, resulting in data-dependent behavior of the embedding operators. This results in a combination of model and data parallelism for the distributed DLRMs shown in Figure 1. In forward propagation, input all-to-all, and embedding all-to-all are required to gather each partition of embedding parameters for each data-parallel sample group. CTR computation waits until all embedding operators are gathered for each sample.

These all-to-alls contribute to the major bottleneck during DLRM inference. In backward propagation, embedding vector gradients must be redistributed. Another backward all-to-all is therefore used to update the embeddings before the next batch iteration starts. These three all-to-alls are the throughput performance killer of DLRM training. As the prediction demand rises, the number of embedding operators grows bigger, and more nodes are added to meet the increasing query requests. Scalability is thus an urgent issue that hinders the development of DLRMs.

To satisfy a growing demand for "smart" networks, SmartNICs with substantial and tightly coupled communication and compute capability are being widely deployed in data centers. Some SmartNICs of interest in this work have an ARM processor, FPGA, domain-specific accelerators, and high bandwidth memory [3, 16, 17, 29, 39, 40]. But besides accelerating packet processing, SmartNICs provide an opportunity for offloading application workloads. Transparent use of SmartNICs alone, however, only improves point-to-point communication. Leveraging SmartNICs' tightly coupled communication and compute capabilities *for applications processing* can be challenging.

2.3 Characteristics of DLRM Data Power Law Distribution

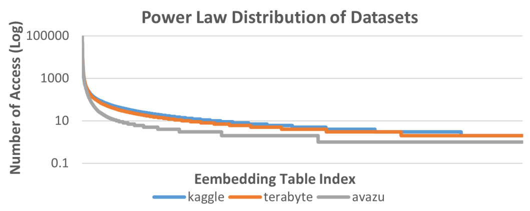


Figure 6: Power Law Distribution of Datasets

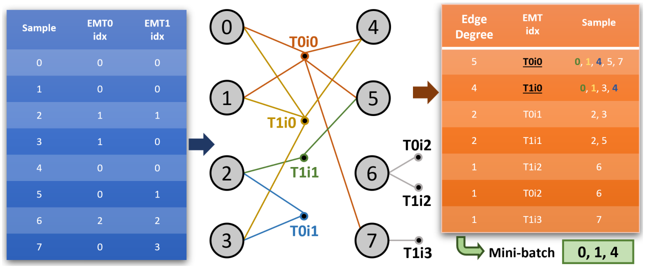
The frequency distribution of a categorical feature's embedding follows a power law distribution (as shown in Figure 6). A small fraction of embeddings results in most of the accesses as, typically, a large fraction of users is interested in a small number of popular items, e.g., web pages or movies. This characteristic leads to data reuse opportunities for the corresponding embedding vectors and for architectures to exploit the resulting locality to overcome the DLRM communication bottleneck.

3 GRAPH ALGORITHM

This section discusses a graph algorithm that enhances the data locality within batches of queries.

3.1 Graph Mini Batch

DLRMs use batches as a processing unit to expose parallelism and enable high throughput. Although a sample's sparse features follow a power law distribution, batches of forward and backward propagation are formed by sequentially fetching independent. Performance can be improved further by improving data locality within the batch: Batches with better data locality can enhance every phase of DLRM processing with reduced embedding lookup, lower memory bandwidth pressure and communication volume, and higher compute efficiency.



Sample	EMT0 idx	EMT1 idx
0	0	0
1	0	0
2	1	1
3	1	0
4	0	0
5	0	1
6	2	2
7	0	3

Edge Degree	EMT idx	Sample
5	T0i0	0, 1, 4, 5, 7
4	T1i0	0, 1, 3, 4
2	T0i1	2, 3
2	T1i1	2, 5
1	T1i2	6
1	T0i2	6
1	T1i3	7

Figure 7: Graph Algorithm (T0i0: Embedding Table (EMT) 0, index 0) Left (blue) table indicates a sample batch that can be viewed as an incidence matrix. The right (orange) table indicates the scoreboard ranking popularity of edges in the hypergraph. A mini batch of samples with high similarity is generated as input to forward propagation.

Instead of fetching samples sequentially from the dataset, we perform the graph algorithm to select a mini batch of samples with high data locality. A larger number of samples is pre-loaded as the input of the graph batch algorithm to select a group of closely related samples for the mini batch. In Figure 7, the blue table indicates the pre-loaded batch of samples that the graph algorithm uses as input. The table can be viewed as a lookup incidence matrix with rows of samples and columns of lookup indices.

Based on the lookup incidence matrix, a hypergraph is formed where samples are nodes and embedding table lookup indices are edges (see middle graph in Figure 7). Sample nodes are connected to index edges corresponding to the lookup table. The hyperedges with higher degrees are formed because DLRM sparse features follow a power law distribution with a higher chance of multiple samples looking up the same popular indices. As the lookup table indicates, embedding table 0 with index 0 (edge T0i0) is requested by sample nodes 0, 1, 4, 5, 7, and table 1 index 0 (edge T1i0) is requested by sample nodes 0, 1, 3, 4. The hypergraph is generated as each of the embedding tables' indices is iterated. Within the graph, an edge with high degrees means more data reuse as a single embedding lookup can serve more sample nodes. Samples nodes that share more overlapped edges have more similarities (share more sparse features) with better data locality. The graph algorithm filters samples that have overlapping embedding lookup indices, which are indices accessed by a larger number of samples. The graph algorithm saves lookup requests, reducing all-to-all communication and reducing memory transfer bandwidth.

The simplified graph algorithm workflow is shown in Algorithm 1. The embedding lookup incidence matrix is generated along with pre-loading the batch of samples from the dataset with a counter attached to each embedding index to register the degree of the index edges. After loading the batch samples, hyperedge degree is sorted using the edge degree counter. The table on the right indicates the popularity of the EMT index from top to bottom. Embedding table 0 index 0 (T0i0) is the most popular index with sample nodes 0, 1, 4, 5, and 7 connected with high similarity.

In the real-world case, however, if the mini batch only chooses samples based on the top edges' degrees, samples in the mini batch cannot be closely related as each sample would have more sparse

Algorithm 1 Graph algorithm

```
1:  Mini_Batch[mini_batch_size]
2:  Incidence_Matrix[batch_size][EMT_idx]
3:  Hyperedge_Cntr[batch_size]
4:  Hyperedge_Threshold = n
5:
6:  # Feed samples from dataset to the incidence matrix
7:  while len(Incidence_Matrix)<batch_size do
8:      Incidence_Matrix ← sample
9:      Update Hyperedge_Cntr[EMT_idx]
10: end while
11:
12: # Get similar samples
13: Sample_List[batch_size] # Initiated as a priority queue
14: for hyper_edge in
15: sorted(Hyperedge_Cntr[0:Hyperedge_Threshold]) do
16:     for sample_id in Hyperedge_Sample[hyper_edge] do
17:         Sample_List ← sample_id
18:     end for
19: end for
20: Mini_Batch ← Sample_List[0:mini_batch_size]
21:
22: # Delete mini_batch samples
23: for Sample_id in Mini_Batch do
24:     Update Hyperedge_Cntr, Incidence_Matrix
25: end for
26:
27: return Mini_Batch
```

features. So we introduce the hyperedge threshold that picks the top *n* popular embedding table indices and does a node selection that selects the sample nodes that appear most often in these indices.

A priority queue is used to maintain the sample list, which is continually updated in descending order based on the number of times each sample appears. During the graph algorithm's execution, the priority queue stores sample nodes in the order of decreasing similarity, with the most similar samples at the top of the queue. Upon completion, the first mini batch size of samples forms a mini batch and feeds into the forward propagation pipeline. As Figure 7 indicates, edges T0i0 and T1i0 are selected as popular indices. Within these two indices, sample nodes 0, 1, and 4 appeared most with a high similarity. The mini batch fetches these three nodes and mini batch of size 3 is formed. The mini batch samples are deleted from the lookup incidence matrix and the Hyperedge counter is updated for the next batch. New samples are fetched from the dataset and forwarded to the next iteration.

3.2 Refresh Batch

As in the previous section, the graph algorithm always finds the most similar (sharing sparse features) samples within batches. So as iterations of the graph algorithm continue, samples that have less common sparse features are always left in the batch. These samples are not chosen by the graph algorithm leading to fewer chances for data reuse. Besides these, the inference of DLRM also has latency requirements. Samples that are left too long in the batch is not served and cause users to never hear back from requests.

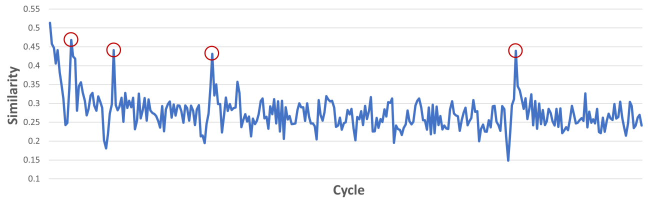


Figure 8: Similarity of Samples within Mini Batches. The red circle indicates a batch refresh with new samples.

To address these issues, we introduce a batch refresh mechanism. The graph algorithm selects samples within the batch one-by-one until it is empty and then refills the batch from the dataset when one of the two criteria has been reached. First, we use a downgrade factor to trace the similarity among the samples in the mini batch selected by the graph algorithm. As the downgrade factor reaches a threshold, this means that the similarity of the mini-batch is too low to form a data reuse opportunity. Batch refresh is triggered to avoid continuing poor data reuse within the mini batch. Second, a timing counter is attached to each sample. When any of the samples has waited too long for service, batch refresh is triggered to consume the batch and refill with new samples. As shown in Figure 8, batch similarities are reduced as processing continues. The red circle means the batch refresh mechanism is triggered to refresh the samples within the batch.

4 SYSTEM HARDWARE ARCHITECTURE

The scalability of DLRM is the central issue that hinders the recommendation system's development as the model size grows even more significant to provide more accurate predictions. The main bottlenecks that impact the scalability of DLRM are memory, communication and computation. In this section, we introduce three aspects of design on SmartNIC for both forward and backward propagation: local cache, remote cache and SmartNIC computation kernel, which address those three bottlenecks.

4.1 Forward Propagation

4.1.1 Local Cache. DLRM has a high demand on memory bandwidth because the embedding operators request frequent and high volume embedding table lookups. The embedding tables can also consume up to terabytes of memory. If the system use a GPU as an accelerator, the memory bandwidth and capacity requirement exceed the GPU's HBM capability. In order to handle the embedding layer, multiple GPUs are aggregated to meet the memory requirement introduced by the large size embedding tables. Embedding tables are distributed to each GPU's HBM memory. As in the previous section, each node receives embedding lookup from every other node. As the scale of the system grows, and the size of the DLRM embedding layer grows, embedding lookup requests increase, which creates more load on the memory. The memory itself could also be the bottleneck of the system's scalability as large numbers of embedding table lookups can introduce significant memory access overheads. For large scale GPU-based clusters, nodes are connected through network interface cards (NICs). Embedding lookup requests are sent through the network, received by NICs, and sent to the GPUs through PCIe. With more frequent requests, the overhead of PCIe and GPU memory access could also diminish the embedding operator's performance.

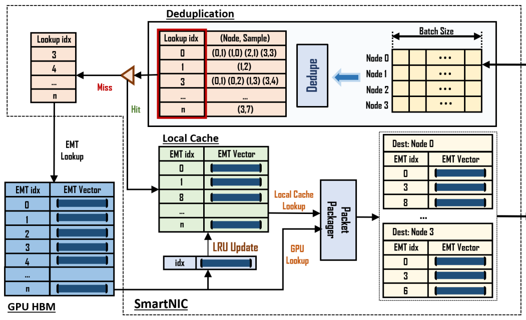


Figure 9: Local cache on the SmartNIC buffers popular embeddings. Hits on local cache save lookup requests to the GPU's HBM. Dedupe eliminates redundant index lookups.

We introduce local cache on SmartNICs to address the DLRM memory bottleneck. The local cache can relieve memory pressure on the GPU by reducing the lookup overhead on the PCIe and GPU memory accesses. Embedding operator requests are first received by the SmartNIC and sent to the GPU for lookup. Local cache on the SmartNIC buffers the lookup result from the GPU. Before generating the GPU memory lookup index list, the lookup indexes are checked if it is hit on the local cache on SmartNIC. If yes, there is no need to send it to the GPU as the embedding vectors are already on SmartNIC. After the GPU's memory lookup, the local cache updates its buffer using the least recently used (LRU) policy. It is very likely that lookup indices hits on the local cache because it buffers the most popular indices of the embedding table. With the introduction of local cache, reduced GPU interruption and memory pressure alleviate the memory bottleneck of processing the embedding operator.

Before the input all-to-all phase, each node sends the corresponding embedding lookup based on the distribution of the embedding table. After the all-to-all, each node receives the lookup requests from other nodes with batches of lookup indices. Because of the power law distribution, a large number of duplicated lookups between nodes and samples take up memory bandwidth. The Dedupe kernel on the SmartNIC uses a deduplication table to keep a record of the lookup index with its corresponding source node and sample ids. The list of lookup indices is checked if it is a hit.

Afterward, a list of lookup indices is summarized and used to issue a lookup request to the GPU's memory. If a hit, then the lookup result is sent to the packet packager to prepare a lookup result packet and update the local cache (using LRU). If a miss, indices are temporarily saved to the GPU lookup request table. As deduplication finishes, lists of lookups are sent to the GPU. When the GPU sends the lookup result back to SmartNIC, the lookup vector is inserted into the local cache on SmartNIC updating cache. The result is formatted by the SmartNIC using the deduplication table to check its source node and sample index. As lookups from dedupe finish, the embedding all-to-all phase exchanges lookup results among the nodes.

4.1.2 *Remote Cache.*

DLRM size is significant because of the embedding operator, which can require terabytes of memory, and is growing as more embeddings provide more accurate predictions. Since replicating the entire model on each device is infeasible, a hybrid model and data parallelism are introduced where the embedding tables are distributed among nodes. A large system can therefore support a large shared memory capacity and lookup bandwidth. As shown in Figure 1, all-to-all communication is needed to gather each node's queries and distributing the embedding tables' lookup results. The all-to-all communication, however, introduces enormous pressure on the system's network and grows quadratically with batch size.

In forward propagation, there are two all-to-all communications. The input all-to-all requests the embedding table lookup for each node's sample queries' sparse features. The embedding all-to-all returns embedding lookup results to each of the sample's original source nodes. The computation kernels wait until the all-to-all phase is finished to continue and compute the prediction results.

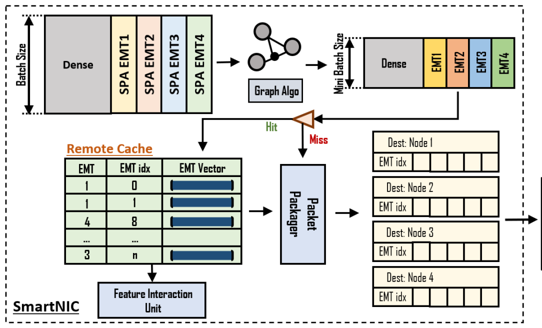


Figure 10: Remote cache on SmartNIC buffers remote embedding tables. Queries check if a remote embedding is a hit on the remote cache before issuing the remote lookup request.

We introduce the remote cache on SmartNIC to address the communication bottleneck by reducing the all-to-all communication workloads. After embedding all-to-all, remote embedding table lookup results are sent back to the original source node. The results are buffered by the remote cache on SmartNIC. The remote cache uses the least recently used (LRU) policy to update the storing vectors as iteration goes. As the system warms up, the remote cache on SmartNIC stores the most popular remote embedding table's vectors. Figure 10 indicates the workflow of how sample queries make use of the remote cache. After the graph algorithm generates mini batch of query samples, samples first check if the lookup is hit on the remote cache. If it hits, this lookup request is not needed and can directly get the remote embedding results from the remote cache. The result is held until embedding all-to-all finishes. Since DLRM samples follow power law distribution, popular remote lookup has a higher chance is requested in the future. The remote cache that buffers the popular embeddings reduces the unnecessary duplicated remote embedding lookup request. If the lookup is missed on the remote cache, it is forwarded to the packet packager for remote lookup.

4.1.3 *SmartNIC Computation.*

DLRM generally exhibits lower arithmetic intensity compared to the traditional DNN model. DLRM consists of various computations including matrix multiplication, no-linear activation, and irregular computation like data reshape matrix flattening, and matrix transposing. These irregular computations can not use GPU's hardware resources efficiently as GPUs are good at massive parallel scalar computations. The irregular compute introduces frequent memory copy that takes enormous amounts of hardware and HBM resources, and can not use the GEMM operator on the GPU. Kernel calling overheads are another factor that largely hinders the computation efficiency of accelerators.

We introduce computation and data management kernels (shown in Figure 11) on SmartNICs to minimize overhead and improve the GPU's computation efficiency. After the embedding all-to-all, each node receives the embedding lookup result from the remote node and is ready to send the result to the GPU for prediction calculation. Instead of dumping the unmanaged raw data directly to the GPU, feature interaction processing, which includes irregular computation operations, are offloaded to the SmartNIC, which has dedicated hardware for higher computation efficiency. The raw data are reorganized by the kernel on the SmartNIC before forwarding to GPU for computation. The computation kernel on smartNIC combines the dense and sparse features as a matrix and the matrix is transposed as the input to feature interaction which uses the matrix multiplication kernel. After feature interaction processing, the result is flattened to convert the 2D matrix into a 1D vector, which is then sent to the GPU for top MLP computation. These dedicated irregular kernels on smartNIC improve the computation efficiency and save hardware utilization for GPU.

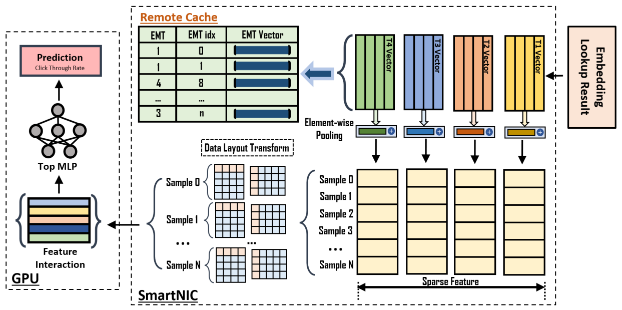


Figure 11: SmartNIC computation with irregular computation kernels including data layout transform, matrix transpose, matrix flattening, and element-wise pooling. Remote cache is updated with the remote embeddings after embedding all-to-all.

4.2 Backward Propagation

DLRM training workflow differs from inference in that synchronization is needed between batches of training samples. In backward propagation, computation kernels for training (training cores) are implemented and supported in the SmartNIC. Embedding gradients are calculated after forward propagation and used to update the embedding table's value. A backward all-to-all gathers every node's partial embedding gradients and collectively updates the distributed embedding value. The next batch of training samples is issued when all embedding tables have been updated. A remote cache would not be effective in training because the cached embeddings are being updated. The updated embeddings are located in the embedding table's original node.

In the forward propagation phase of the training process, the local cache on the SmartNIC reduces the system memory accessing pressure. In backward propagation, the SmartNIC computation kernel improves throughput with two levels of reduction, local and global, using the reduction computation kernel on the SmartNIC.

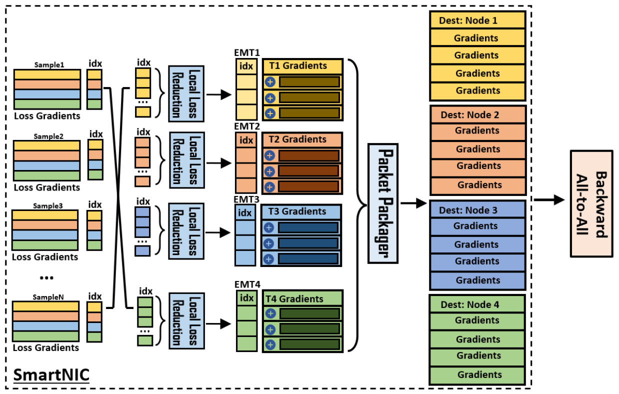


Figure 12: Backward propagation local gradient reduction using SmartNIC computation

4.2.1 Local Gradient Reduction. The embedding gradients are generated during backward propagation. After the GPU's gradient calculation, gradients are sent to the SmartNIC for backward all-to-all communication that updates the corresponding embedding tables. The DLRM samples adhere to a power law distribution, which increases the likelihood that samples within batches on each node access the same embedding value and result in repeated updates of the gradients for those embeddings. The local gradient reduction kernel combines the gradients for the same embeddings on each node and calculates the overall gradient for each embedding. After all samples have been handled, the packet generator packs the updated gradients for each embedding table and sends them to their destinations. Local gradient reduction kernel reduces the workload of backward all-to-all and saves GPU overhead.

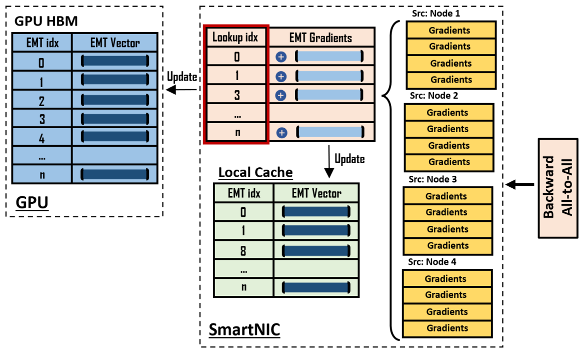


Figure 13: Backward propagation global gradient reduction using SmartNIC computation. The gradient is updated both in the SmartNIC's local cache and the GPU's HBM.

4.2.2 *Global Gradient Reduction.*

After backward all-to-all, each node has received embedding gradients from every other node. A table of embedding vector gradients is updated as they are received. When this is completed, the final gradients are used to update the embedding table's value. The local cache is used in the forward propagation process and stores the embedding vector value from the GPU's HBM (using LRU). So the local cache is guaranteed to have stored the embedding vector that needs to be updated by the gradients. The final gradient is updated on both the local cache on the SmartNIC and the GPU's HBM. The updated local cache buffers the updated embedding for future batches. With the local cache and reduction kernel on the SmartNIC, memory bandwidth pressure, PCIe overhead, and interruption overhead of the GPU's function calls are all minimized.

4.3 An Alternate Design

An alternate design was also explored where the entire application is offloaded onto the SmartNIC. Even though large amounts of PCIe, kernel call, and software stack overheads are saved, the performance does not approach that of the heterogeneous architecture (with GPU) because of the limited hardware resources and higher computational capability of the GPU.

5 EVALUATION

5.1 Experimental Setup

The DLRM model evaluation is based on Meta Research open source code [7]. The CPU baseline uses a 16-core 32-thread Intel® Xeon® Gold 6226R; the GPU baseline uses an Nvidia RTX8000. To evaluate the SmartNIC hardware, we use an Xilinx Alveo U280 FPGA with configure hardware implementations of runtime, cache system, and computation kernels using High Level Synthesis (Vitis HLS 2022.1). For scalability, we used MPI as the backend for CPU and GPU evaluation. Multiple Alveo U280 FPGAs are connected through transceivers as a cluster to evaluate the multi-SmartNIC system. Since there is no exact heterogeneous SmartNIC-GPU system, we evaluated the real SmartNIC parameters on the muti-node Alveo U280 cluster and plugged it into a simulator based on DLRM open-source code [7] with a GPU as an accelerator.

We evaluated the system using three well-established recommendation model datasets: Criteo Kaggle, Criteo Terabyte and Avazu for both inference and training. Table 1 shows the parameters of these datasets.

Table 1: DLRM Datasets Parameters

Dataset	Kaggle	Terabyte	Avazu
Dense Feature	13	13	1
Sparse Feature	26	26	21
EMT Rows	33.8M	226M	9.3M
Row Dim	16	64	16

The evaluation figures use the following abbreviations: BL = Baseline, GA = Graph Algorithm, LC = Local Cache on SmartNIC, RC = Remote Cache on SmartNIC, SC = SmartNIC Computation, FS = Full (complete) SmartNIC system.

5.2 Performance Evaluation

5.2.1 *Graph Algorithm.*

The graph algorithm improves system performance in inference by 1.2 × on average, with less impact in training. It is more practical and effective in inference due to high number and randomness of the query requests in real-time: As the algorithm rearranges batch clusters of similar queries, system communication and memory pressure are reduced. In training, the dataset samples are given with more straightforward data pre-processing which achieve comparable data locality.

5.2.2 *Forward Propagation.*

This section evaluates the system performance with forward propagation using the graph algorithm, local cache, remote cache, SmartNIC computation, and full SmartNIC centric design.

(1) Local Cache: Figure 16 illustrates the latency speedup of using local cache and graph algorithm with respect to three datasets. The forward propagation latency performance is not significantly impacted by the local cache and the speedup remains steady as the size increases. This is mainly because batches of the forward propagation are pipelined. The latency of batches is hidden if the lookup request misses the local cache on the SmartNIC and fetches the lookup result from GPU. Using the graph algorithm improves the latency by speedup of average 1.2× of three datasets as it enhances the data locality within batches, reducing remote communication and memory access, and thus improving overall latency. Figure 18 shows the Hit/Miss rate of local cache on SmartNIC and GPU's memory. As the size of the local cache increases, fewer memory accesses are needed from the GPU's memory. The local cache relieves the memory bandwidth pressure on the GPU.

(2) Remote Cache: Figure 19 shows the latency speedup of forward propagation with Remote Cache on the SmartNIC and graph algorithm. Remote cache largely caches the remote embedding lookup because of the power law distribution characteristic. When the query sample hits the remote cache, the remote cache lookup communication request is saved, reducing both queries' latency and the system's communication workload, and improving communication efficiency with less communication congestion. Figure 20 shows saved remote lookup requests using remote cache and graph algorithm. As the remote cache size increases, more lookups are saved. Graph algorithm clusters similar samples to reuse lookups more effectively. Figure 21 shows the decline of the local cache hit rate as remote cache size increases. This is because embedding table accesses follow a power law distribution. A larger remote caches store more popular embedding indices, causing less popular embeddings to remain in GPU's HBM. Queries for remote embeddings are more likely to request these unpopular embeddings.

(3) SmartNIC Computation: Figure 17 shows the speedup of using SmartNIC computation in forward propagation. The figure indicates that there is not a noticeable effect on latency compared to training. This is because inference batches are processed in a pipeline, and the main bottlenecks are embedding lookup and communication. The latency reduction from the SmartNIC irregular computation kernel only accounts for a small portion of the entire process. However, during training, SmartNIC computation is important for reducing gradients, which is described below.

(4) Forward Propagation Overall Speedup: Figure 22 shows the overall latency speedup of the techniques across three datasets.

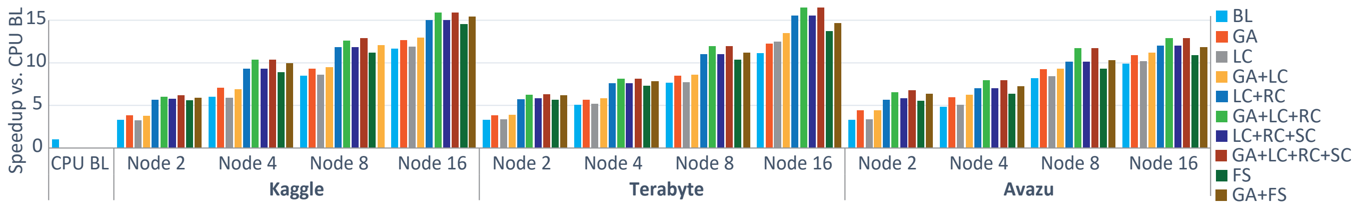


Figure 14: Inference Scalability

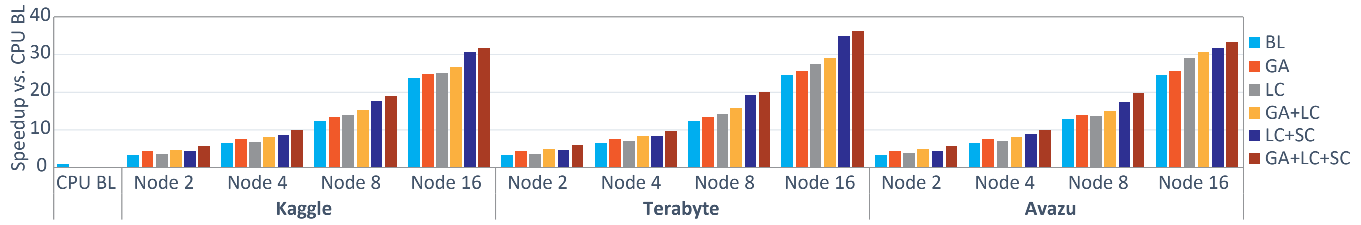


Figure 15: Training Scalability

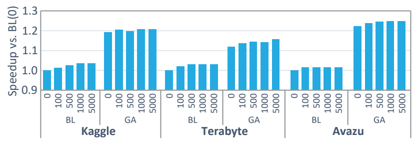


Figure 16: Latency Speedup of Forward Propagation Using Local Cache on SmartNIC

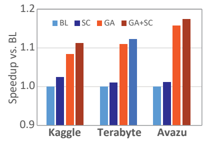


Figure 17: Latency Speedup of Forward Propagation using SC

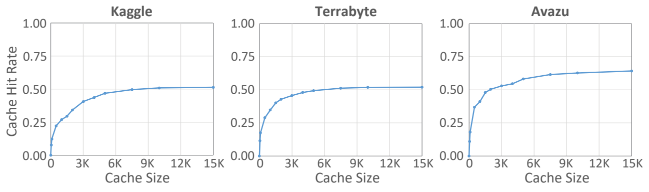


Figure 18: Forward Propagation Hit Rate of Local Cache on SmartNIC. Missing on local cache indicates an embedding lookup on the GPU's HBM.

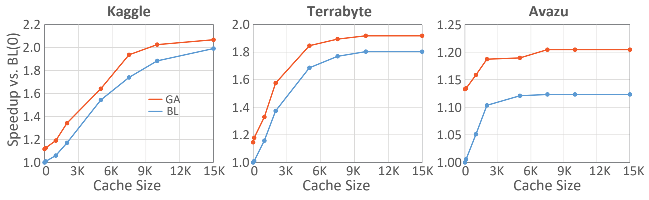


Figure 19: Latency Speedup of Forward Propagation Using Remote Cache on the SmartNIC

The speedup is higher for Kaggle and Terabyte that for Avazu. The graph algorithm improves the data locality of samples within batches and enhances the overall system latency. The figure highlights that remote cache and graph algorithm significantly enhance forward propagation by reducing communication workloads and increasing data reuse. Fewer communication requests significantly reduce the all-to-all bottleneck. Meanwhile, local cache and SmartNIC computation show limited impact on forward propagation as

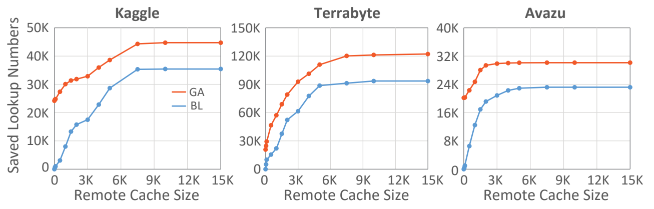


Figure 20: Saved Lookups (Remote Embedding Lookup Requests) of Forward Propagation Using Remote Cache

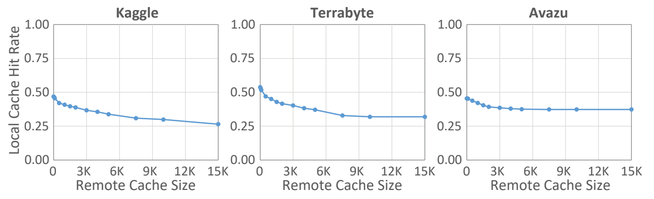


Figure 21: Hit Rate of Local Cache Using Remote Cache of Forward Propagation

inference batches are processed in a pipeline, hiding the latency between them. These two techniques mainly alleviate GPU memory bandwidth pressure, kernel overhead, and hardware resource utilization.

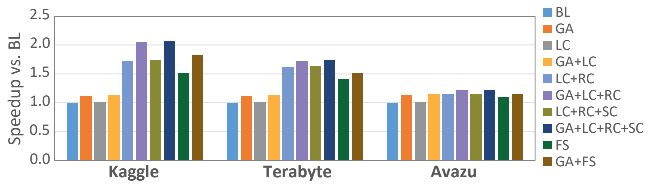


Figure 22: Comparison of latency speedup of forward propagation using the graph algorithm, local cache, remote cache, SmartNIC computation, and the full SmartNIC system.

5.2.3 Backward Propagation. In backward propagation, samples are trained in sequential batches, with each batch starting only after the previous one is completed. During this process, the embedding vectors are updated with gradients computed in the backward pass. The remote cache is not beneficial in this stage as the cached embedding vectors become outdated after backward propagation.

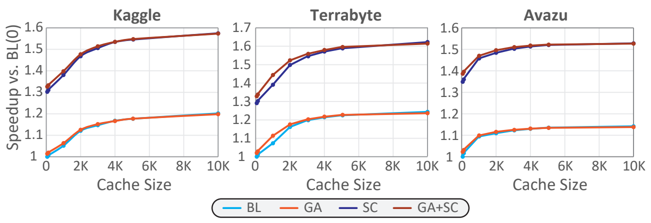


Figure 23: Throughput Speedup of Backward Propagation Using Local Cache on SmartNIC

Figure 23 shows the training throughput speedup of graph algorithm, local cache, and using SmartNIC compute with three datasets. As the figure indicates, as local cache size increases, the throughput increases accordingly. This results from more embedding table requests being serviced by the local cache of the SmartNIC, reducing the pressure on GPU memory bandwidth and the embedding lookup overhead. Figure 24 shows the hit/miss rate of local cache on SmartNIC.

SmartNIC computation is crucial in improving training throughput by handling irregular computation and reducing gradient updates shown in figure 23. Two levels of reduction, including gradient reduction on the local node and global gradients reduction of all other nodes, are performed on the SmartNIC.

Figure 25 indicates an overall throughput speedup. The graph algorithm improves the overall throughput speedup by an average of 1.1×. Local cache improves the throughput speedup by 1.3×. SmartNIC computation improves the throughput by 1.4×. Overall the throughput speedup can reach 1.5×.

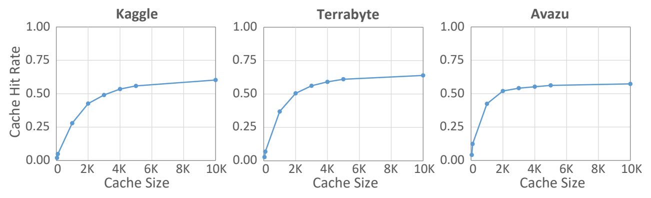


Figure 24: Backward Propagation Hit Rate of Local Cache on SmartNIC. Missing local cache indicates embedding lookup on the GPU's HBM.

We also evaluated the effect of batch size on the throughput speedup with SmartNIC computation shown in figure 26. The results show that as batch size increases, the speedup becomes bounded by the computation bottleneck.

5.3 System Scalability

We evaluated the system's scalability of inference and training using 2, 4, 8, and 16 nodes shown in figure 14 and 15. The embedding tables are evenly distributed among each node. We use 2-node CPU MPI as an overall baseline.

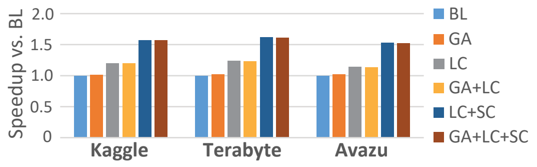


Figure 25: Throughput Speedup of Backward Propagation Using Graph Algorithm, Local Cache and SmartNIC Computation Kernels

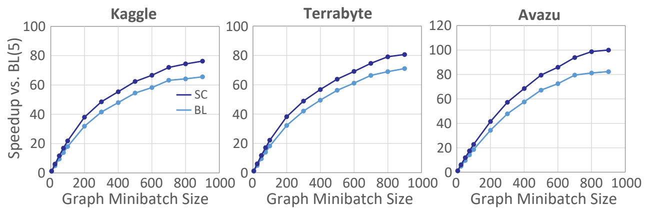


Figure 26: Throughput Speedup wrt Training Batch Sizes

Figure 14 shows inference scalability. We tested the same workloads with respect to the number of nodes and found that as the number of nodes increases, the per-node embedding table size decreases resulting in reduced memory bandwidth pressure. However, communication workloads increase as the node number scales up. Overall, the inference latency speedup shows better scalability using techniques on SmartNICs.

Figure 15 shows the scalability of training throughput speedup. As the system scales out, all-to-all communication and backward propagation is mainly the bottleneck that limits the scalability of training throughput. SmartNIC computation plays an important role in driving the scalability of the system's throughput speedup.

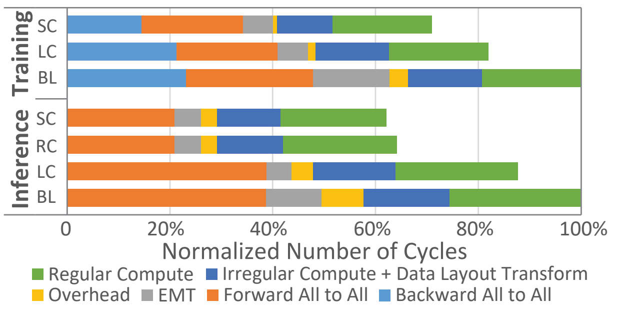


Figure 27: Time Breakdown of DLRM Inference and Training. Overhead includes NIC to GPU PCIe latency, kernel call overhead, host-to-device, etc. Backward all-to-all refers to gradient update in backward propagation.

Figure 27 shows a breakdown of these techniques for inference and training. In inference, all-to-all communication takes nearly 40% of the total time. The local cache on SmartNIC reduces the embedding lookup time. The remote cache saves a significant amount of all-to-all communication time, as popular embeddings are stored locally on the SmartNIC and do not require communication requests. Both the local cache and remote cache eliminate overhead, e.g., the host-to-NIC latency via the PCIe bus. In training, SmartNIC computation reduces both backward all-to-all for gradients update and irregular computation latency.

6 RELATED WORK

Commercial FPGA-based SmartNICs have been released by various vendors. AMD's offerings include the Alveo U25 [39] and the SN1000 [40] SmartNIC with FPGA programmable logic and an ARM core. Intel's include the Infrastructure Processing Unit (Intel IPU) [16] and the Intel FPGA SmartNIC [17]. Nvidia provides the DPU [29] for data center AI and networking workloads. Broadcom's Stingray SmartNIC [3] has an 8-core ARM CPU and a P4 packet processing engine. See also [2, 34] for surveys.

SmartNICs are being used as computation resources to offload networking functions and applications. Catapult [4] uses FPGA-based network solution to offload network applications and certain AI applications. Work by [18] proposed a configurable network protocol on intelligent NICs. COPA [21] provides a software/hardware framework that makes the underlying FPGA hardware (SmartNIC device) agnostic to middleware. FCsN [9, 10] proposed a high-performance FPGA centric SmartNIC framework, which supports domain-specific computation, low-latency communication, and host-detached scheduling. FAST extends some of these capabilities to a switch-attached configuration [12, 13]. INCA [31] provides general-purpose compute capabilities for SmartNICs that can be utilized when the network is inactive. sPIN [15] provides a portable programming model to offload simple packet processing.

DLRM requires high memory bandwidth and capacity [23, 37, 41–43]. Hashing functions are optimized in work by Weinberger, et al. [37]. Sethi, et al. [33] optimizes embedding partitioning and placement techniques. Lim, et al. and Lin et al. [24, 25] use various quantization schemes to reduce communication volume. These studies address embedding operators using software and algorithm solutions that do not fundamentally solve the DLRM bottleneck in a hardware aspect.

Much attention is given to using GPUs as computation accelerators. Mudigere, et al. [26] introduced a software-hardware co-design system using GPU for distributed training. Kwon, et al. [22] propose a software runtime system that manages GPU DRM as a fast "scratchpad." Other work explores using storage technology to enhance the performance embedding operator of DLRM. Eisenman, et al. [6] present a storage system that reduces the DRAM footprint using Non-volatile Memory. Wilkening, et al. [38] proposed a near data processing solution that improves the performance of underlying SSD storage for embedding table operators. These studies, however, are not focused on the communication bottleneck that arises as the DLRM scales up. Zhu, et al. [44] present an FPGA cluster for recommendation inference for embedding lookups and computation. Jiang, et al. [19] proposed a recommendation inference engine using FPGAs' high bandwidth memory and a pipelined dataflow. These studies do not target scalability as the recommendation model grows.

7 CONCLUSION

Processing DLRMs is one of the important applications in large-scale online services as DLRMs have evolved into the single largest machine learning application. The software-hardware co-design heterogeneous SmartNIC system targets the scalability challenges of DLRM processing, including communication, memory, and computation. The graph algorithm clusters similar queries into batches resulting in higher system efficiency and performance. This system pushes the performance boundary of current software and hardware platforms.

8 ACKNOWLEDGEMENTS

This research was partially supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This work was supported, in part, by the NSF through awards CCF-1919130 and CCF 2151021, by grants from Red Hat, and by AMD and Intel both through donated FPGAs, tools, and IP.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. 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 Mane, 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 Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://doi.org/10.48550/ARXIV.1603.04467

[2] C. Bobda, J. Mandebi, P. Chow, M. Ewais, N. Tarafdar, J.C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M.C. Herbordt, H. Shahzad, P. Hofstee, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier. 2022. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Transactions on Reconfigurable Technology and Systems* 15, 3 (2022), 1–42. doi: 10.1145/3506713.

[3] Broadcom. 2019. Stingray PS250 2x50-Gb High-Performance Data Center SmartNIC. https://docs.broadcom.com/doc/PS250-PB

[4] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783710

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[6] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. 2018. Bandana: Using Non-volatile Memory for Storing Deep Learning Models. https://doi.org/10.48550/ARXIV.1811.05922

[7] facebookresearch. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. https://github.com/facebookresearch/dlrm

[8] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (dec 2016), 19 pages. https://doi.org/10.1145/2843948

[9] Anqi Guo, Tong Geng, Yongan Zhang, Pouya Haghi, Chunshu Wu, Cheng Tan, Yingyan Lin, Ang Li, and Martin Herbordt. 2022. FCsN: A FPGA-Centric SmartNIC Framework for Neural Networks. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 1–2. https://doi.org/10.1109/FCCM53951.2022.9786193

[10] Anqi Guo, Tong Geng, Yongan Zhang, Pouya Haghi, Chunshu Wu, Cheng Tan, Yingyan Lin, Ang Li, and Martin Herbordt. 2022. A Framework for Neural Network Inference on FPGA-Centric SmartNICs. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. 01–08. https://doi.org/10.1109/FPL57034.2022.00071

[11] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. 2019. The Architectural Implications of Facebook's DNN-based Personalized Recommendation. https://doi.org/10.48550/ARXIV.1906.03109

[12] P. Haghi, A. Guo, Q. Xiong, C. Yang, T. Geng, J.T. Broaddus, R. Marshall, D. Schafer, A. Skjellum, and M.C. Herbordt. 2022. Reconfigurable switches for high performance and flexible MPI collectives. *Concurrency and Computation: Practice and Experience* 34, 2 (2022). doi: 10.1002/cpe.6769.

[13] P. Haghi, W. Krska, C. Tan, T. Geng, P.H. Chen, C. Greenwood, A. Guo, T. Hines, C. Wu, A. Li, A. Skjellum, and M.C. Herbordt. 2023. FLASH: FPGA-Accelerated Smart Switches with GCN Case Study. In *ICS 2023: International Conference on Supercomputing.*

[14] 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. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA).* 620–629. https://doi.org/10.1109/HPCA.2018.00059

[15] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. SPIN: High-Performance Streaming Processing In the Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17).* Association for Computing Machinery, New York, NY, USA, Article 59, 16 pages. https://doi.org/10.1145/3126908.3126970

[16] Intel. 2021. Intel® Infrastructure Processing Unit (Intel® IPU). https://www.intel.com/content/www/us/en/products/network-io/smartnic.html

[17] Intel. 2022. Intel® FPGA SmartNIC. https://www.intel.com/content/www/us/en/products/details/fpga/platforms/smartnic.html

[18] R.G. Jaganathan, K.D. Underwood, and R. Sass. 2003. A configurable network protocol for cluster based communications using modular hardware primitives on an intelligent NIC. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.* 286–287. https://doi.org/10.1109/FPGA.2003.1227273

[19] 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. 2020. MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions. https://doi.org/10.48550/ARXIV.2010.05894

[20] Jie. 2020. Training Deep Learning Recommendation Model with Quantized Collective Communications.

[21] Venkata Krishnan, Olivier Serres, and Michael Blocksome. 2020. COnfigurable Network Protocol Accelerator (COPA) † : An Integrated Networking/Accelerator Hardware/Software Framework. In *2020 IEEE Symposium on High-Performance Interconnects (HOTI).* 17–24. https://doi.org/10.1109/HOTI51249.2020.00018

[22] Youngeun Kwon and Minsoo Rhu. 2022. Training Personalized Recommendation Systems from (GPU) Scratch: Look Forward Not Backwards. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22).* Association for Computing Machinery, New York, NY, USA, 860–873. https://doi.org/10.1145/3470496.3527386

[23] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. https://doi.org/10.48550/ARXIV.1909.11942

[24] Hyeontaek Lim, David G. Andersen, and Michael Kaminsky. 2018. 3LC: Lightweight and Effective Traffic Compression for Distributed Machine Learning. https://doi.org/10.48550/ARXIV.1802.07389

[25] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2017. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. (2017). https://doi.org/10.48550/ARXIV.1712.01887

[26] 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. 2022. 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* (New York, New York) *(ISCA '22).* Association for Computing Machinery, New York, NY, USA, 993–1011. https://doi.org/10.1145/3470496.3533727

[27] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, Krishnakumar Nair, Isabel Gao, Bor-Yiing Su, Jiyan Yang, and Mikhail Smelyanskiy. 2020. Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems. https://doi.org/10.48550/ARXIV.2003.09518

[28] 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. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. https://doi.org/10.48550/ARXIV.1906.00091

[29] Nvidia. 2021. NVIDIA BLUEFIELD-2 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf

[30] 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. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. https://doi.org/10.48550/ARXIV.1811.09886

[31] Whit Schonbein, Ryan E. Grant, Matthew G. F. Dosanjh, and Dorian Arnold. 2019. INCA: In-Network Compute Assistance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19).* Association for Computing Machinery, New York, NY, USA, Article 54, 13 pages. https://doi.org/10.1145/3295500.3356153

[32] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. https://doi.org/10.48550/ARXIV.1802.05799

[33] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. 2022. RecShard: Statistical Feature-Based Memory Optimization for Industry-Scale Neural Recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22).* Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3503222.3507777

[34] H. Shahzad, A. Sanaullah, and M.C. Herbordt. 2021. Survey and Future Trends for FPGA Cloud Architectures. In *IEEE High Performance Extreme Computing Conference.* DOI: 10.1109/HPEC49654.2021.9622807.

[35] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining* (Virtual Event, CA, USA) *(KDD '20).* Association for Computing Machinery, New York, NY, USA, 165–175. https://doi.org/10.1145/3394486.3403059

[36] Brent Smith and Greg Linden. 2017. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing* 21, 3 (2017), 12–18. https://doi.org/10.1109/MIC.2017.72

[37] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2009. Feature Hashing for Large Scale Multitask Learning. https://doi.org/10.48550/ARXIV.0902.2206

[38] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. RecSSD: Near Data Processing for Solid State Drive Based Recommendation Inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21).* Association for Computing Machinery, New York, NY, USA, 717–729. https://doi.org/10.1145/3445814.3446763

[39] Xilinx. 2020. Alveo U25 SmartNIC Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u25.html

[40] Xilinx. 2022. The Industry's First SmartNIC With Composable Hardware. https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html

[41] Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. 2020. Mixed-Precision Embedding Using a Cache. https://doi.org/10.48550/ARXIV.2010.11305

[42] Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. 2021. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models. https://doi.org/10.48550/ARXIV.2101.11714

[43] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. https://doi.org/10.48550/ARXIV.2003.05622

[44] Yu Zhu, Zhenhao He, Wenqi Jiang, Kai Zeng, Jingren Zhou, and Gustavo Alonso. 2021. Distributed Recommendation Inference on FPGA Clusters. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL).* 279–285. https://doi.org/10.1109/FPL53798.2021.00057