DISAGGREGATED MULTI-TOWER: TOPLOGY-AWARE MODELING TECHNIQUE FOR EFFICIENT LARGE SCALE RECOMMENDATION

Liang Luo [1] **Buyun Zhang** [1] **Michael Tsang** [1]
Yinbin Ma [1] **Yuxin Chen** [1] **Ching-Hsiang Chu** [1] **Yanli Zhao** [1] **Shen Li** [1]
Yuchen Hao [1] **Guna Lakshminarayanan** [1]
Ellie Dingqiao Wen [1] **Jongsoo Park** [1] **Dheevatsa Mudigere** [2] **Maxim Naumov** [1]

ABSTRACT

We study a *mismatch* between the deep learning recommendation models' flat architecture, common distributed training paradigm and hierarchical data center topology. To address the associated inefficiencies, we propose Disaggregated Multi-Tower (DMT), a modeling technique that consists of (1) Semantic-preserving Tower Transform (SPTT), a novel training paradigm that decomposes the monolithic global embedding lookup process into disjoint towers to exploit data center locality; (2) Tower Module (TM), a synergistic dense component attached to each tower to reduce model complexity and communication volume through hierarchical feature interaction; and (3) Tower Partitioner (TP), a feature partitioner to systematically create towers with meaningful feature interactions and load balanced assignments to preserve model quality and training throughput via learned embeddings. We show that DMT can achieve up to 1.9× speedup compared to the state-of-the-art baselines without losing accuracy across multiple generations of hardware at large data center scales.

1 INTRODUCTION

Recommendation models have played a critical role in online services including search engines, social media, and content platforms. Recent advancements in recommendation models are in large part brought by the use of neural networks and the exponential growth in model capacity. To date, deep-learning powered recommendation models with billions to even of trillions of parameters are no longer uncommon (Zhao et al., 2019; Jiang et al., 2019; Lian et al., 2021; Zhao et al., 2022).

Modern recommendation models consist of a *sparse* component that converts categorical features into dense representations using *embedding lookup tables*, and a *dense* neural network component that interacts the embedding of categorical features and dense features and produces an output. Since the embedding tables can be huge, the state-of-the-art practices train these models in a hybrid fashion: the sparse part is trained in a model parallel and the dense part in a data parallel manner (Naumov et al., 2020; Mudigere et al., 2022; Lian et al., 2021; Zhang et al., 2022a), and the embedding lookup requests/responses are routed to respective accelerators via AlltoAll collectives and dense parameters

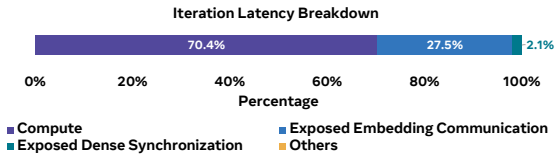

Figure 1. Exposed latency breakdown of training DCN on a cluster with 64 H100 GPUs with state-of-the-art optimizations.

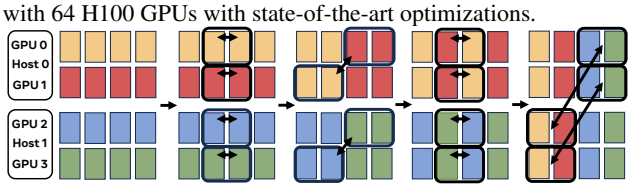

Figure 2. An example of a topology-aware communication pattern that is semantically-equivalent to AlltoAll.

are synchronized through AllReduce operations.

This training paradigm has faced increasingly more challenges due to the divergence of two model and hardware scaling trends (§2): (1) the use of more sparse features and higher dimensional embeddings (Kang et al., 2020) increases the communication volumes of the AlltoAll communications; (2) the speed of improvements of compute capacity in modern accelerators have significantly outpaced that of the physical network in datacenter environments (Luo et al., 2018; Sapio et al., 2019). For example, Table 1 lists recent datacenter developments, where the floating point

[1]Meta AI [2]Nvidia, work done while at Meta. Correspondence to: Liang Luo <liangluo@meta.com>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2024.

System	Peak FP Perf	Scale-out/GPU	Scale-up/GPU (unidirection)
V100, 2019	15.7 TF/s	100 Gbps	150 GB/s
A100, 2022	156 TF/s	200 Gbps	300 GB/s
H100, 2023	989 TF/s	400 Gbps	450 GB/s

Table 1. Recent generational upgrades reported by sources (Hazelwood et al., 2018; Smelyanskiy, 2019; Mudigere et al., 2022; Edwards, 2022; Morgan, 2023) and manufacturer show the improvements of compute capacity significantly outpaces that of the network bandwidth's, making communication the bottleneck.

compute power improved by 60×, whereas the scale-out bandwidth only increased by 4× within the same time frame. Consequently, the bottleneck of recommendation model training has shifted to the communication of the embedding lookup request and response and synchronization of the dense gradients. Even with modern optimizations such as quantized communications (Yang et al., 2021), automatic column-wise sharding (Zha et al., 2023), overlapped compute and communication (Rashidi et al., 2021) and fused embedding lookup kernel (Khudia et al., 2021), in a typical datacenter environment with 64 GPUs and fast RDMA network, up to 30% of time within an iteration is spent explicitly waiting on the network communication, see Figure 1, causing a large inefficiency in the datacenter utilization.

Recent proposals including Piper (Tarnawski et al., 2021), Alpa (Zheng et al., 2022), Megatron (Narayanan et al., 2021), and ZeRo (Rasley et al., 2020; Zhao et al., 2023; Zhang et al., 2022b) aim to find alternative paradigms for large scale training. As we show later, they are unable to suggest better parallelism that beats hybrid parallelism for modern recommendation models empirically. This implies that the current hybrid parallelism represents a near-optimal configuration in the known parallelism search space, and the fact that it fails to achieve satisfactory efficiency calls for more efforts in the domain of improving recommendation model training performance.

Fundamentally, we show that this inefficiency stems from a mismatch between the model architecture, training paradigm and datacenter topology: typical datacenters have different hierarchies of communication bandwidth, with drastically faster intra-host (e.g., NVLink) than cross-host connections (e.g., RDMA), but most recommendation models are flat, meaning both the embedding distribution and the feature interaction processes are global and monolithic.

To address this mismatch, we propose the Disaggregated Multi-Tower (DMT) modeling technique that transforms the model and takes advantage of the heterogeneity in datacenter for efficiency improvements (§3). We show that the AlltoAll communication corresponding to a large exposed latency in Figure 1 can be efficiently expressed through a set of local shuffles and localized communications shown in Figure 2. Further, we show that through careful design of modules, the latency and communication volume can be reduced without hurting model quality. Specifically, DMT achieves

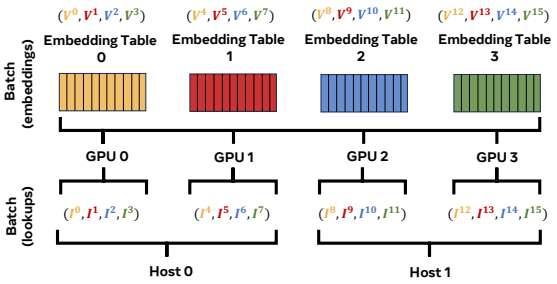

Figure 3. A conceptual view of the embedding lookup process.

this by (1) partitioning the sparse features into semantic-preserving and balanced *towers* through *Tower Partitioner* (TP), a learned feature partitioner; (2) decomposing the global embedding distribution into disjoint towers to exploit datacenter locality, through a novel training paradigm called *semantic-preserving tower transform* (SPTT); (3) introducing *tower modules* (TM) that are synergistic to SPTT and leverages hierarchical feature interaction to reduce both model complexity and communication volume.

We optimized the implementation of DMT (§4) and applied it to two popular and widely-used open source models DLRM (Naumov et al., 2019) and DCN (Wang et al., 2021)) as well as an internal workload. In our experiments we show that across three generations of hardware (Nvidia V100, A100 and H100) on up to 512 GPUs, DMT outperforms the state-of-the-art baseline in training throughput by up to 1.9× without losing the statistical quality (§5).

2 Challenges of Large-Scale Recommendation Models Training

This section provides contexts for the challenges of training large-scale recommendation models in a data center setting.

2.1 Deep Learning-based Recommendation Models

Modern deep learning-based recommendation models share a similar structure: they usually consists of large embedding tables and dense networks (Zhao et al., 2019; Jiang et al., 2019; Lian et al., 2021; Zhao et al., 2022; Zhang et al., 2022a). These models take in both dense (continuous) and categorical (sparse) features as inputs. The sparse features are first processed by the embedding tables, from which embeddings are acquired. The dense features are then joined with the embeddings for feature interaction. Finally, the outcome of this interaction is fed to another dense network where a prediction is produced. Perhaps the most unique aspect about recommendation model is the embedding lookup process, and we explain it more in Figure 3 using an example: we have 4 GPUs spanning over 2 hosts, and each of which is given an input sample (I). Each sample contains 4

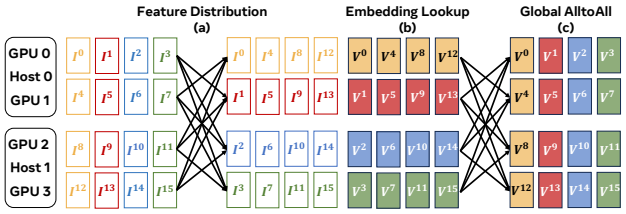

Figure 4. An operational view of the classic embedding lookup process in current systems.

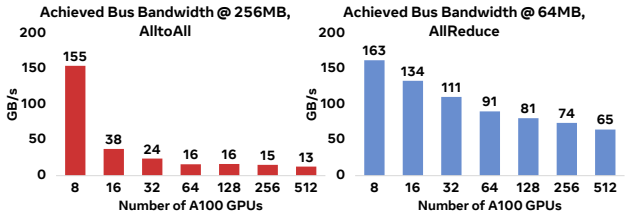

Figure 5. Scalability (weak scaling) of NCCL collectives in a datacenter environment, 8 GPUs/nodes, measured in bus bandwidth.

color-coded features, with inputs to these features labeled $I^{4r}...I^{4r+3}$ for clarity of explanation, where r is the rank of the GPU. Each I^k is itself a tuple of discrete values, and the goal of embedding lookup is to convert each I^k into a dense representation V^k for all ks.

2.2 Training Process

The current state-of-the-art frameworks, Neo (Mudigere et al., 2022) and TorchRec (Ivchenko et al., 2022), train these models using a hybrid parallelism: the embedding tables are sharded to different devices in model parallelism, and the dense networks run as data parallel. The adoption of the hybrid training paradigm is a result of two factors: (1) embedding tables are simply too large to be synchronized as a dense component and (2) in each iteration, access to these tables is sparse so only a few entries are updated. Since the training process of dense components is similar to models in other domains, we focus our discussion on the embedding lookup mechanism, highlighted in Figure 4.

The setup is the same as Figure 3 to facilitate illustration. At first, the local sparse features are communicated with devices where their respective embedding tables reside through a global AlltoAll communication (step a). Each GPU is then tasked with looking up embeddings for the global batch for the categorical features assigned to it (step b). The lookup process involves parallel queries into embedding table entries and pooling if necessary. Afterwards, another AlltoAll communication is used to return embeddings to each GPU (step c). This concludes the training forward pass for embedding lookup. In the backward pass, gradients to activated embedding table entries are routed through another AlltoAll.

2.3 Training Bottleneck Analysis

This training paradigm incurs significant overheads and hinders efficient training of such models in a typical datacenter environment, with the main bottleneck being the inefficient AlltoAll collectives triggered by the global embedding distribution process, as shown in Figure 1.

2.3.1 Degrading Collective Performance vs Scale.

The direct reason for poor training efficiency is the subpar collective performance. To show this, we measure the bus bandwidth achieved running DLRM's collectives, Allreduce and AlltoAll with NCCL (v2.18.3), using typical buffer size close to the one used in typical DLRM training rounded up to the nearest power of 2 (64MB of dense size, and 256MB of embedding size at the batch size of 16K, in float).

As shown in Figure 5, performance of collectives drops sharply versus scale, and the consequence of this is exacerbated by the fact that AlltoAll and AllReduce collectives are invoked at least 3 and 1 time(s) per iteration, respectively, in hybrid parallelism.

2.3.2 Mismatched model architecture, training paradigm, and data center topology.

The degrading communication performance is hardly a surprise at a large scale (Liu et al., 2017; Ramesh et al., 2020; Graham et al., 2016; Luo et al., 2020b;a), therefore it is superficial to blame the poor efficiency solely on it. Here, we would like to argue that the fundamental issue is the mismatch between model architecture, training paradigm and data center topology:

- The model architecture, which usually requires global feature interaction, together with the training paradigm, dictate multiple rounds of global AlltoAlls for embedding lookup. This means we need to send a byte on wire (forward and backward pass) just to read/write a byte (times pooling factor) from the GPU memory during embedding lookup (assuming no quantization). Since the discrepancy between memory bandwidth and network bandwidth is huge (orders of magnitudes) in current systems, this makes network a bottleneck in training.
- The training paradigm fails to exploit locality within a single host: the fast NVLink connection within a single host delivers much higher bandwidth compared to cross-host links. Since the latency of AlltoAll is bounded by the slower cross-host RDMA links, NVLink offers limited benefits for the final collective performance.

2.4 Ineffectiveness of Existing Solutions

Given the above analysis, it is worth asking: (1) can recent sharding improvements solve the problem? and (2) can we improve performance by only optimizing the training

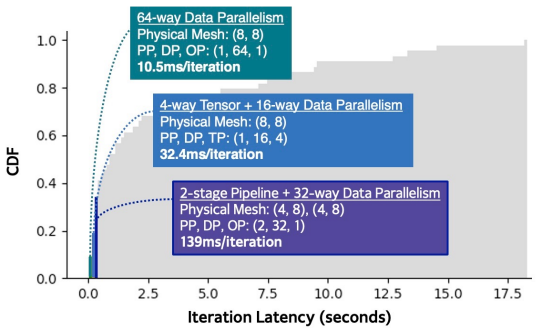

Figure 6. CDF of the iteration latency for various parallelism configurations running DLRM with Alpa.

paradigm without modifying the model itself? Unfortunately, the answer to both questions is likely, no.

On one hand, NeuroShard (Zha et al., 2023) improves embedding distribution process with better load balance, but it cannot solve the problem of high AlltoAll latency even with a perfect load balance, as we show in §5. On the other hand, as we show later, applying recent advancements including Piper, Alpa, Megatron, and ZeRo on recommendation models have barely any effect, as they cannot address the nontrivial overhead from sparse components in recommendation models.

For example, we leveraged Alpa to search for the optimal parallelism strategy for the dense part of DLRM on 64 A100 GPUs by enumerating various physical and logical meshes, with the results summarized in Figure 6 as a CDF of the iteration latency for each valid configuration. We then highlight the bins where the fastest configuration that uses data, tensor and pipeline parallelism lies. Evidently, data parallelism stands out alone as the fastest parallelism for the dense part of DLRM. As a result, the widely adopted hybrid training approach indeed represents a (near) optimal configuration in the known parallelism search space. We cannot address the fundamental problems with existing tools.

3 Disaggregated Multi-Tower

We now describe DMT, a topology-aware modeling technique designed to address the fundamental mismatch of model, paradigm and data center topology. We introduce 3 main concepts that support DMT: (1) a semantic-preserving tower transform (SPTT) that exploits data center topology for faster communication; (2) a tower module (TM) synergistic with SPTT for model complexity and communication volume reduction; and (3) a learned, balanced feature partitioner (TP) that generates effective tower partition.

3.1 Semantic-Preserving Tower Transform

In this paper, we call a group of sparse features, dense layers that consume their embeddings, together with a group of GPUs that host them a *tower*. Typically, a tower is constructed on a collection of accelerators that have high communication locality (e.g., GPUs within a host). Originally, each model can be viewed as a single tower because feature interactions are global.

In this section, we show that if we have decomposed the single-towered model into a multi-towered variant by partitioning features into different groups, then through the use of SPTT, we can improve its training performance by tapping into data center topology without changing the semantics of the model. Without losing generality, we first assume the features are pre-partitioned for us, and a later section explains how such partitions can be obtained.

From an operational perspective, SPTT breaks down the global embedding AlltoAll operation into a series of steps, where each step is either a data shuffle operation, an intra-host collective, or a cross-host communication.

3.1.1 SPTT: a Walk-through

To best illustrate the idea of SPTT, we use the same example as in Figure 4, where we have four features assigned to two towers, and each tower consists of 2 GPUs within the same host. Concretely, tower 0 consists of the orange and red sparse features assigned to host 0, and tower 1 with the blue and green sparse features assigned to host 1. The SPTT process starts with the same global feature distribution via an AlltoAll call (step a), followed by the local embedding lookup process (step b). We then proceed with a *peer permute* to rearrange the received features on each GPU into a *peer order* (step c). Formally, we define peers of a GPU ranked g_i to be the collections of GPUs whose rank g_j satisfies $g_i \% L = g_j \% L$, where L is the total number of GPUs on a host, and the peer order is defined as the sorted total order of all GPUs with rank g_i using the tuple $(g_i \% T, g_i // L)$ as key, where T is the number of towers (hosts). In our example, the peer order of GPUs is (0, 2, 1, 3).

With the embeddings are now in peer order, step d, an intra-host communication (usually another AlltoAll operation) follows to exchange partial embeddings with local GPUs such that each GPU holds embeddings results for all its peers for all the features (embedding tables) sharded to it (in our case, we have each GPU holding only one feature/table). Although step d is a collectives communication, it is highly efficient thanks to the fast NVLink interconnection.

Step e involves a series of data shuffle operations - we first view the resulting tensor of step d by stacking it into a 2D tensor with dimension (features, peers), then we transpose it to the layout of (peers, features), and finally view it as a

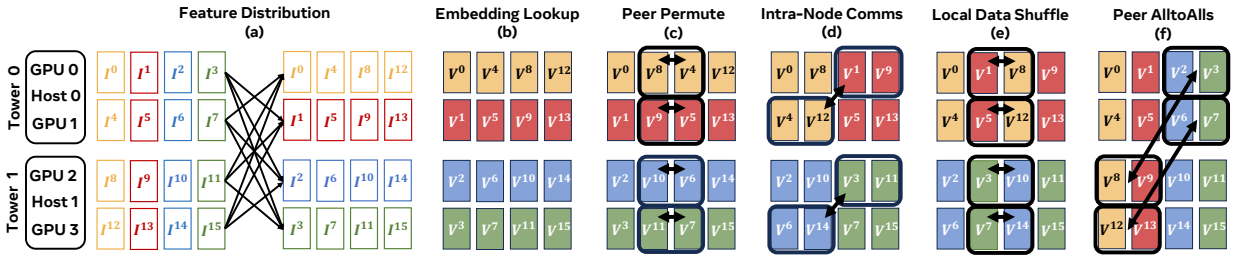

Figure 7. An operational view of the topology-aware embedding lookup process using semantic-preserving tower transform.

flattened tensor (peers $\times$ features) for the final step. The last step (step f) involves concurrent peer AlltoAlls, where each GPU communicates with its peers. With a total number of G GPUs in the cluster, there will be L such AlltoAlls in parallel, with each AlltoAll operate within a world of size equals to the number of towers $T = G//L$.

3.1.2 *Benefits of SPTT*

Since SPTT does not reduce total bytes on wire, and it introduces potential data shuffle overhead and additional communication steps, its advantage may not be obvious instantly. However, SPTT comes with two important properties that significantly benefit training throughput in a typical data center setting: (1) even if SPTT does not reduce bytes on wire, §2.3.1 shows that with the same data volume, running in a smaller world size improves communication throughput. Thus, with a reduction of world size by L times in step f, we expect a handy performance improvement at a large scale; (2) the unique data layout makes it possible to construct dense *tower modules* (detailed later) that can further reduce model complexity and cross-host communication sizes without incurring cross-host synchronization overhead. We evaluate these claims in later sections.

3.1.3 *Specialized SPTT*

While we discussed the general case of SPTT, we note that SPTT can be customized to fit different needs. First, a tower can be assigned to a set of K hosts instead of 1, and SPTT will hold without modification when $G\%K = 0$. This allows us to trade off reduction in world size of peer AlltoAll with increased overhead of step c-e. Second, Figure 7 showed a general case where embedding tables are assigned as a whole to a GPU, but SPTT can still work in case a table is too large to fit, by leveraging column-wise or row-wise sharding within a tower. For example, for single-hot features, step d is a simple AlltoAll operation that communicates the column-wise shards; but with multi-hot features, we can leverage row-wise sharding and step d becomes a ReduceScatter. Third, depending on the relative data volume of the sparse inputs versus the embedding volume, step b and step c can be swapped so data shuffle on the object with a smaller size. Last but not least, step c can even be omitted by using a virtual process group where the GPUs are already in the peer order.

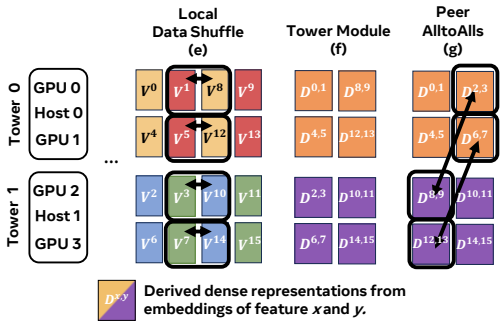

Figure 8. Tower Modules and Hierarchical Feature Interaction.

3.2 Tower Modules and Hierarchical Feature Interaction

SPTT created a large opportunity to further decrease model complexity and compress cross-host communication volume without incurring large global synchronization overhead or quality degradation through the idea of *hierarchical feature interaction*.

To do so, we can construct tower modules (TM) right at the end of step e in SPTT. As shown in Figure 8, we can derive compressed representations for features in a tower which in our setup is $D^{x,y} = TM(V^x, V^y)$, with the expectation that $|D^{x,y}| < |V^x| + |V^y|$ so the network volume at step g can be reduced. We view the benefits of tower modules from two aspects:

System Perspective TM can further compress output embedding before step f for a reduced cross-host communication volume and computation complexity. TM leverages the fact that data layout of the outcome of SPTT where each GPU holds the embeddings of all features in a tower for all its peers and that GPUs in a tower fully contain the global embeddings for that entire feature group, so the synchronization of TM on the backward pass only needs to happen within a tower (e.g., a host), exploiting the communication locality (e.g., NVLink) for throughput gain. TM can compress the embedding outputs of tables by installing additional dense modules, and its potential reduction of model compute complexity is also sizable: for example, consider a

pair-wise dot-product feature interaction module with $|F|$ features, then the original global feature interaction has a complexity of $O(|F|^2)$; with a dot-product based TM with a feature reduction ratio r, the new compute complexity is reduced to $O(\frac{|F|^2}{T^2} + r^2|F|^2)$.

Model Perspective With potentially reduced output size of each tower produced by TM, the global feature interactions at later stage of the models no longer operate on the full feature set, leading to potential loss in model quality. However, we can compensate for this loss of quality through carefully designed, higher-order feature interactions using TM with the following observations:

- Feature interaction can be sparse, and hence global feature interaction may have inherent redundancy (Adnan et al., 2023). With recent models incorporating multi-domain features (Zhou et al., 2023; Deldjoo et al., 2020), not all pair-wise feature interaction is meaningful. Thus, by carefully partitioning global features into groups of features with meaningful interaction, we can capture the most important interactions that contribute to model quality with lower computation cost.
- TM essentially introduces an additional order of interaction. Higher order interactions through more stacked interaction layers has been a major contributor to better quality in recommendation models (Wang et al., 2021; Lian et al., 2018; Kang et al., 2020). TM can be viewed as a special mechanism to achieve higher order interactions as it captures group-level interactions followed by cross-group interactions in a hierarchical manner. Further, with inherent heterogeneity in the TM, the ability to capture a diverse set of interaction is enhanced, which is shown to be more effective than stacking homogeneous interaction modules alone (Zhang et al., 2022a).
- Mixture of Experts (MoE) is a powerful paradigm that extends the capacity and expressiveness of models (Fedus et al., 2022; Lepikhin et al., 2020). Traditional application of MoEs is on the batch dimension through a gated network, and on the contrary, TM can be viewed as a MoE on the feature dimension.

The following sections detail how we create such meaningful feature partitions and design concrete tower architectures for specific recommendation models.

3.3 Learned, Balanced, and Meaningful Feature Partition via Tower Partitioner

We must take both desired system property (balanced, so that work is balanced across hardware) and model property (cohesive, so that features that have meaningful interactions are kept together) into account when producing feature partitions to generate towers. Traditional approaches to this problem usually involves some form of feature interaction probes followed by a manual process for balanced assignments. This approach can be cumbersome when the natural number of feature clusters do not match the desired tower sizes (data center topology). To solve this issue, we propose a learned, balanced, end-to-end Tower Partitioner (TP).

Without lose of generality, assume we have a feature interaction result I of dimension ($|F|$, $|F|$), obtained from an original model with normalized feature embedding F_i (a N-dimensional embedding vector for feature i), using a chosen kernel K, where $|F|$ is the cardinality of features in the model. In TP, we limit our discussion to K being the cosine similarity kernel $K = F_i^T F_j$ due to its popularity. Specifically,

$$I(i,j) = abs(K(F_i, F_j))$$

This yields a interaction matrix with values between 0 and 1, where 0 indicates a pair of features have nothing in common (orthogonal), and 1 indicates the features are strongly positively or negatively related. We then convert I into a distance matrix $D = f(I)$. Notice that this conversion need not preserve the exact distance, as long as it preserves the relative distance among embeddings.

With D, we embed features in a Euclidean space of dimension n, obtaining a set of coordinates for each feature $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,n})$ whose pairwise distance corresponds to distance matrix D. To save computation, and to reduce noise in the embedding process, we have $n < N$. We use an optimizer (e.g., Adam) to solve for X_i to minimize the following objectives:

$$\sum_{i=1}^{|F|} \sum_{j=1}^{i-1} (d_{i,j} - D(i,j))^2$$

where $d_{i,j} = \|X_i - X_j\|_2$ is the Euclidean distance between embedded coordinates for feature i and j.

We then use a constrained K-Means (Bradley et al., 2000) algorithm to partition the embedded features into balanced groups, with the constraint that the maximum group size to be at most K (tunable) times the minimum tower size.

One may question why not directly use the raw embedding Fs for TP. This is because in a minibatched training setting with batch size B, the raw embedding R is of shape (B, F, N), and averaging along B dimension has no meaning because the embedding index for each feature can be different across batches. And instead, opting for the average feature affinity with $mean(RR^T, dim = 0)$ is semantically meaningful because the similarity between features across samples should be coherent.

Compared to traditional graph cut-based, NP-hard algorithms (Wikipedia, 2023) which usually optimizes only for

an aggregate metrics in each partition (e.g., sum of similarities within each partition) that may not have inherent meaning, TP provides an efficient means to directly optimize for the target metrics. TP provides two ways to find meaningful partitions: the first is to assign features that are more different to a group, increasing diversity in each tower, and this is done by setting $f(I) = I$ when obtaining D. We call this the *diverse strategy*; the second is to assign features that are more similar to a group, increasing coherence in each tower: this can be achieved by setting $f(I) = 1 - I$ to derive D and we call this the *coherent strategy*. We believe the better choice can vary by model and dataset, and we simply try both to find the optimal setting.

4 Implementation and Optimization

We built DMT in PyTorch on top of Neo (Mudigere et al., 2022) and TorchRec (Ivchenko et al., 2022). This section describes a few implementation details.

Embedding Table Sharding We leverage the framework's embedding sharder to support placement of embedding tables within a tower to individual GPUs. Specifically, for a large batch size and single-hot embedding, we pin embedding types to column-wise shard due to its lower communication volume, and for small-batch size and multi-hot embedding, row-wise sharding is used.

Tower Module Architectures We introduce two concrete TM architectures for two widely-used recommendation models by industry, DLRM and DCN. Note our goal here is not to highlight specific module implementations but to demonstrate a systematic and practical way of constructing TM by lifting the main interaction architecture from the underlying model. In both cases, we constrained our choice of operators from the ones used in the interaction arch when building TM. Each TM takes in two inputs: a *embs* of shape $(B, |F|, N)$ which is the output of step e in Figure 7 for each tower and a output feature dimension D.

Listing 1. Tower Module for DLRM

```
def forward(embs, c, p, D):
    B, F, N = embs.shape
    embs_flat = embs.view(B, -1)
    o1 = linear(in_f = N*F, o_f = p*D)(
        embs_flat).view(B, -1)
    o2 = linear(in_f = N, o_f = c*D)(embs).
        view(B, -1)
    return cat([o1, o2], dim = 1)
```

In DLRM, the tower architecture is an ensemble of a simple linear combination of the input features and a projection on the embedding dimension. These operators resemble those used in DLRM overarch. The parameter c and p controls the output feature count for these two linear operators. The output dimension becomes $O = D(c|F| + p)$.

Listing 2. Tower Module for DCN

```
def forward(embs, D):
    B, F, N = embs.shape
    o = crossnet(F*N, ...)(embs)
    return linear(in_f = F*N, o_f = F*D)(o)
        .view(B, -1)
```

As an optimization, we implement dot product with a manual pair-wise routine when the batch size is large and $|F|$ is relatively small, as we find the generated *cublasGemvTensorStridedBatched* CUDA kernel vastly slower than the manual routine on the data layout we have.

The TM in DCN is a smaller CrossNet, the main interaction module in DCN, with an output dimension of $O = |F|D$. In both cases, we can derive the *compression ratio* of network communication for step e in Figure 7 as $CR = \frac{\sum_{i=0}^{T} O_i}{|F|N}$.

5 Evaluation

We now assess the ability of DMT to generalize across models and preserve model quality, and highlight its throughput gain over state-of-the-art baselines with a detailed ablation study of the factors that contributed to DMT's gain.

5.1 Experimental Setup

Models We consider two families of recommendation models. The first family is the open-source models which include two types of interactions: dot-product (DLRM) and CrossNet (DCN). They share the same embedding table sizes and have about 90GB of total parameters and consumes 14 to 96MFlops/sample to train. The second family contains an internal, extra-large model (XLRM for short) similar to those described in industry (Mudigere et al., 2022; Lian et al., 2021). This model contains about 2 trillion parameters and has a complexity around 700 MFlops/sample.

To create meaningful and balanced partitions for DMT, we use the dot-product based TP on a 2D plane with $R = 1$ for constrained K-Means. We use this grouping for both training throughput and accuracy evaluation. We pin each tower module to a single host to best leverage NVLink.

Hardware Where applicable, we run our experiments on multiple hardware platforms based on the Nvidia V100, A100 and H100 GPUs. We vary the number of GPUs used in our benchmarks from 16 to 512. Our infrastructure guarantees full bisection bandwidth between any pair of hosts with no oversubscription.

Strong Baseline We use the official implementations in the state-of-the-art recommendation framework TorchRec (TorchRec, 2023; Ivchenko et al., 2022) from Meta for the open source models as the baseline. We turn on all applicable optimizations including quantized embed-

Config	Batch Size	AUC	Epoch Time
Baseline (DLRM)	2048	0.8030	6.5hrs
Strong Baseline (DLRM)	128K	**0.8047**	**29mins**
Baseline (DCN)	128K	0.7963	58mins
Strong Baseline (DCN)	128K	**0.8002**	**27mins**

Table 2. We use a Strong Baseline that achieves better evaluation AUCs and higher throughputs compared to TorchRec's results using the same hardware. Baseline's epoch time is as reported.

ding and gradient communication, pipelined data fetching, and TorchRec's auto planner to determine the best sharding strategy (e.g., column-wise, row-wise, or hierarchical sharding for embedding tables). We exclude data parallelism for embedding tables in the search space because synchronizing embedding tables that way is generally impractical (Mudigere et al., 2022). To help improve performance, we manually include a column-wise sharding factor to balance system loads when there are more GPUs than embedding tables so TorchRec can tap into the collective bandwidth of the whole cluster. For dense component sharding, we use data parallel, suggested by Alpa, as the optimal strategy (Figure 6). We then build DMT on top. Additionally, we improve on the evaluation accuracy reported by the official implementation, by 0.17% and 0.39% respectively for the open source models using the Adam optimizer attached with a tuned learning rate schedule. With these changes incorporated, we refer to the baseline used in the following evaluation as Strong Baseline, whose characteristics are shown in Table 2.

5.2 Accuracy and Generalizability

We first evaluate SPTT and TM in their generalized abilities to capture interactions and minimize accuracy losses across recommendation models. We summarize model size and complexity as reported by the PyTorch profiler. To properly reflect run to run variance, we run each experiment at least 9 times and report the 1-epoch median evaluation AUC along with its standard deviation. We use the Adam optimizer and keep all hyperparameters identical for DMT and all open source model's experiments for fair comparison. We set global batch sizes to 128K for all models. To test the generalization of DMT as a modeling technique itself, we do not put effort in hyperparameter tuning for individual DMT models, inline with prior art (Narang et al., 2021). We evaluate DLRM and XLRM for 4B and 35B samples on the Criteo and an internal dataset, respectively.

Model	AUC (Std)	MFlops/Sample	Parameters (G)
DLRM	0.8047 (0.0004)	14.74	22.78
SPTT-DLRM	0.8053 (0.0004)	14.74	22.78
DCN	0.8002 (0.0003)	96.22	22.79
SPTT-DCN	0.8001 (0.0002)	96.22	22.79

Table 3. semantic-preserving tower transform achieves neutral AUC compared to unmodified DLRM models.

5.2.1 Semantic-Preserving Tower Transform

We create a separate a pass-through tower for each embedding feature in DMT and compare AUC with the original models to show that since semantic-preserving tower transform only orchestrates dataflow but leaves compute intact, it has no implication on model quality, as shown in Table 3.

5.2.2 Tower Modules

We use the TM detailed in §4 for the open source models. Since TM modifies semantics, we show that we can recover global interactions through hierarchical interaction, provided that there is a sufficient amount of towers.

Model	AUC (Std)	MFlops/Sample	Parameters (G)
DLRM Strong Baseline	0.8047 (0.0004)	14.74	22.78
DMT 2-DLRM	0.8046 (0.0004)	8.95	22.78
DMT 4-DLRM	0.8045 (0.0003)	8.95	22.78
DMT 8-DLRM	0.8045 (0.0001)	8.95	22.78
DMT 16-DLRM	0.8047 (0.0003)	8.75	22.78
DMT 26T-DLRM	0.8047 (0.0004)	8.95	22.78
DCN Strong Baseline	0.8002 (0.0003)	96.22	22.79
DMT 2T-DCN	0.7998 (0.0002)	43.71	22.81
DMT 4T-DCN	0.8003 (0.0002)	50.01	22.79
DMT 8T-DCN	**0.8006** (0.0002)	62.60	22.81
DMT 16T-DCN	0.8001 (0.0002)	87.19	22.79
DMT 26T-DCN	0.8001 (0.0002)	96.22	22.79

Table 4. Median evaluation AUC (with standard deviation) comparing open source models with their DMT counterparts, which achieve on par or better AUC with on par or lower resources.

Open Source We show the AUC results of applying tower modules on top of SPTT. For DLRM, we set $p = 1, c = 0, D = 128$ for 16 towers, and $c = 1, p = 0, D = 64$ for 2-8, 26 towers. For DCN, we set $D = 128$ for 2-16 towers (for 26-tower DCN, we simply use SPTT alone). The baselines use $N = 128$ as the default embedding dimension. While there are other parameters to TM that achieve similar results, we picked these because they achieved best throughput (discussed in the next section).

It is worth noting that changing the number of towers have impacts on the total parameter count: more towers may introduce more parameters in the tower modules, but can reduce parameters in the over arch. This effect is more pronounced in DCN than DLRM, because dot-product is parameter-free but CrossNet is not. Since the Criteo dataset has 26 sparse features, we set number of towers to 26[1].

Table 4 summarizes the results: DMT is able to achieve on par (within one standard deviation) or better AUC (more than one standard deviation) with on par parameter count and flops usage.

To assess how AUC varies according to the compression

[1]Note that this is not a hard limitation for DMT: if we wish to have more towers to accommodate to a larger cluster size, we can always perform column sharding to embedding tables.

Compression Ratio	AUC (Std)	MFlops/Sample	Parameters (G)
2	0.8045 (0.0001)	8.95	22.78
4	0.8036 (0.0003)	7.93	22.78
8	0.8022 (0.0004)	7.41	22.78
16	0.8000 (0.0003)	7.16	22.78

Table 5. Median evaluation AUC (with standard deviation) versus varying tower compression ratio of DMT 8T-DLRM.

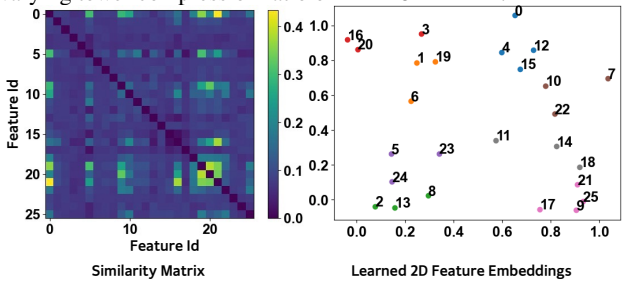

Figure 9. The similarity matrix of TP using the *coherent* strategy and partition of features into 8 color-coded towers.

ratio (CR), we set D of towers in DMT 8T-DLRM to 64, 32, 16 and 8, with a corresponding compression ratio of 2, 4, 8 and 16. We summarize the result in Table 5: we observe an expected gradual degradation of AUC with larger compression ratio used in the tower arch.

XLRM We partition the sparse features into 16 towers and keep the tower module's operators same as its main interaction type. Since the XLRM's embedding table sizes are significantly larger than those in the open source models, we use a column-wise sharding scheme to slice embedding tables into equal-dimension chunks, and invoke TP for tower partition. We train both the original model and the DMT model on a CTR task, and we observe a normalized entropy (He et al., 2014) improvement of 0.02%.

5.2.3 *Effectiveness of TP*

We now assess the role of TP in creating meaningful feature partitions and how it affects model quality.

Config (LR)	TP (Std)	Naive (Std)	p-Value
DMT 16T-DLRM (1e-3)	**0.7990** (0.0003)	0.7981 (0.0003)	0.0006
DMT 8T-DCN (2e-3)	**0.8006** (0.0002)	0.8003 (0.0003)	0.0023

Table 6. TP creates more meaningful feature to tower assignment compared to a naive assignment, achieving better median AUC with statistical significance in various settings.

Open Source To demonstrate how TP improves AUC under various hyperparameter settings, we introduce a balanced but naive, sequential feature to tower assignment scheme with a stride equal to the number of towers as the baseline, e.g., for 8 towers the resulting assignment is [[0, 8, 16, 24], [1, 9, 17, 25], [2, 10, 18], [3, 11, 19], [4, 12, 20], [5, 13, 21], [6, 14, 22], [7, 15, 23]]. We report the resulting AUC using TP and this assignment in Table 6. Evidently, TP's partition leads to higher model quality. We derive the staticstical signifcance of our results by performing the Mann-Whitney U test for 9 repeated experiments. With p values low enough we reject the null hypothesis that two experiments using TP and naive assignments have equal chance of yielding better AUC in these settings.

XLRM We find patterns in D captured by TP mostly manifest as interactions between dedicated item, item-user, and dedicated user features. This naturally leads us to first partition features into these 3 categories, and further partition them through TP into individual towers of desirable counts. Our experiments suggests this strategy of partition is the main contributor of the NE improvement observed in §5.2.2. Our further experiments suggest once we have features grouped into these three groups, exact partitions of these groups into desired towers have marginal effect.

5.3 Performance and Scalability

This section demonstrates the throughput improvement of DMT across a wide range of hardware platforms and scales. To minimize variance introduced by the data ingestion pipeline, we use a random dataset for throughput evaluation.

5.3.1 *Overall Throughput Gain*

We first look at the overall speedup of DMT.

Open Source We vary the number of towers in DMT to match the number of hosts from 16 to 64. Each partition corresponds to one configuration in §5.2. We keep the local batch size fixed in this experiment at 16K.

We summarize the result in Figure 10. Overall, DMT is able to achieve up to 1.9X and 1.8X speedup across 3 hardware platforms[2] at a large scale. On the DLRM side, we see a general trend of increasing speedup as we upscale our cluster, because the effects of the reduced communication volume from tower modules outweigh the overhead of data shuffle and compute latency from SPTT and tower modules as scale grows. On the DCN side, DMT scores a large speedup at a small scale. This is mostly due to the reduced model complexity of DMT as characterized in Table 4. This phenomena is especially pronounced in older GPUs, which is more bounded by compute capacity.

XLRM We train DMT and the original XLRM on 128 V100 and A100 GPUs. DMT-XLRM is Note that DMT-XLRM achieves a lower speedup compared to open source models because XLRM is much more compute-bound.

5.3.2 *Ablation Study*

We conduct a comprehensive ablation study on the open source models to understand the origin of speedups by analyzing the effect of SPTT and TM on throughput and quan-

[2]Since our V100 cluster can support at most 16 hosts, no data is provided for the 32, 64-host cases.

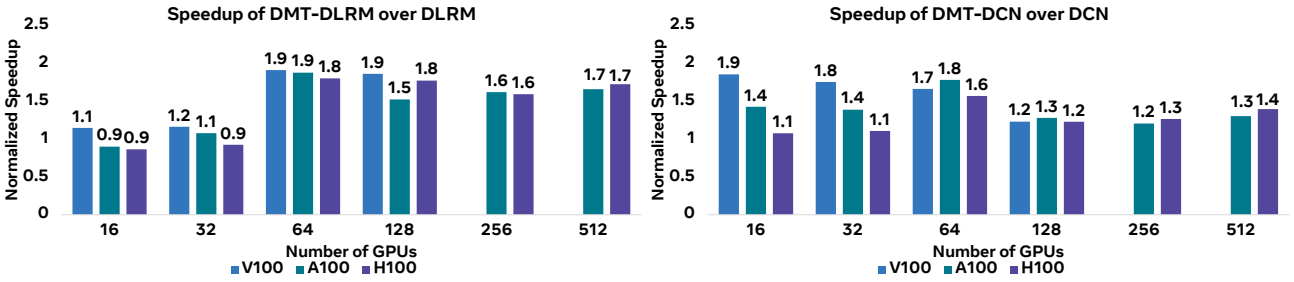

Figure 10. Speedup of DMT over DLRM and DCN across different hardware platforms and scales.

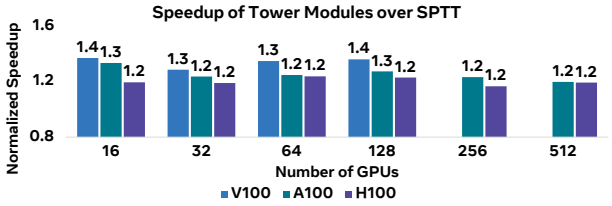

Figure 11. Speedup of tower modules over SPTT on DLRM.

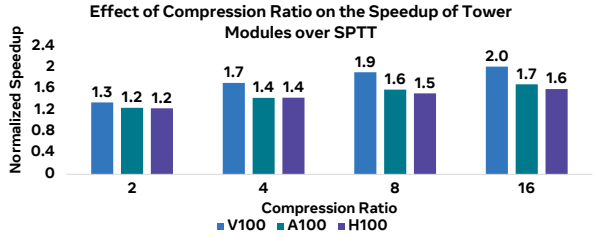

Figure 12. Effects of compression ratio on the speedup of DMT 8T-DLRM over DMT-SPTT.

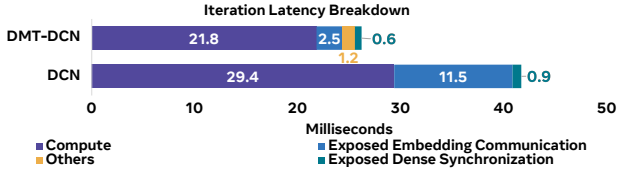

Figure 13. DMT improves training latency of all components.

tifying how component latency has improved with DMT.

Effects of SPTT and TM We compute the speedup of DMT with TM over DMT SPTT-only from 16 to 512 GPUs (2-26 towers using the same configuration as in Figure 10), and summarize in Figure 11. Noticeably, as the training scale grows, the effects of tower modules grow, contributing up to 1.4X additional throughput gain.

Effects of Compression Ratio in TM Compression ratio provides a flexible mechanism to tradeoff AUC with throughput. Figure 12 quantify this on DMT 8T-DLRM. Unsurprisingly, by cross-referencing with Table 5, we observe an increased throughput of up to 2X at a cost of AUC of $< 0.5\%$ with a high compression ratio of 16.

Reduction in Component Latency Finally, we provide a breakdown of iteration latency comparing DMT-DCN and DCN to quantify the reduction of latency in each component in Figure 13, running on 64 H100 GPUs. Evidently, DMT-DCN sees most improvements in latency from compute (1.4X) and exposed embedding communication (4.6X).

6 Discussion

Hierarchical Collectives and SPTT The use of hierarchical collectives for improving performance is not new, as explored by MICs (Zhang et al., 2022b), FSDP (Zhao et al., 2022), DeepSpeed-MoE (Rajbhandari et al., 2022), PLink (Luo et al., 2020b), NetHint (Chen et al., 2022), Heirring (Thangakrishnan et al., 2020) and PXN (Mandakolathur & Jeaugey, 2022). However, the novelty of SPTT is as follows: (1) it is an unique application of hierarchical collectives to break down the monolithic embedding lookup process which can be specialized to accelerate different pooling types using highly efficient local shuffles and communications (§3.1.3); (2) while hierarchical collectives does not reduce bytes on wire, the synergy between SPTT and tower modules make it natural to further compress cross-host communication volumes, making significant additional speedups possible (§5.2.1); (3) compared to hierarchical collectives, SPTT incurs overhead per iteration instead of per collectives.

Quantization Compared to quantization, DMT may result in better model quality and provide asymptotically better speedup. Our extended experiment shows quantizing the XLRM model to FP8 already causes 0.1% significant quality degradation without extensive tuning; further, on 1024 H100 GPUs, quantized DMT-XLRM can still outperform FP8-quantized XLRMs by up to 1.2X in throughput.

7 Conclusion

We proposed DMT, a novel topology-aware modeling technique for recommendation models that improves throughput by up to 1.9× at a large scale ($>$ 64 GPUs) while preserving model quality. DMT achieved this through semantic-preserving tower transform, a novel training paradigm that exploits data center locality, tower modules, synergistic modules that reduces and compresses model complexity and communication volumes, and TP, a learned, balance feature partitioner that creates meaningful towers.

REFERENCES

Adnan, M., Maboud, Y. E., Mahajan, D., and Nair, P. J. Adrec: Advanced feature interactions to address covariate-shifts in recommendation networks. *arXiv preprint arXiv:2308.14902*, 2023.

Bradley, P. S., Bennett, K. P., and Demiriz, A. Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0): 0, 2000.

Chen, J., Zhang, H., Zhang, W., Luo, L., Chase, J., Stoica, I., and Zhuo, D. {NetHint}:{White-Box} networking for {Multi-Tenant} data centers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 1327–1343, 2022.

Deldjoo, Y., Schedl, M., Cremonesi, P., and Pasi, G. Recommender systems leveraging multimedia content. *ACM Comput. Surv.*, 53(5), sep 2020. ISSN 0360-0300. doi: 10.1145/3407190. URL https://doi.org/10.1145/3407190.

Edwards, C. Meta's grand teton brings nvidia hopper to its data centers — nvidia blog. https://blogs.nvidia.com/blog/2022/10/18/meta-grand-teton/, 10 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

Graham, R. L., Bureddy, D., Lui, P., Rosenstock, H., Shainer, G., Bloch, G., Goldenerg, D., Dubman, M., Kotchubievsky, S., Koushnir, V., Levi, L., Margolin, A., Ronen, T., Shpiner, A., Wertheim, O., and Zahavi, E. Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction. In *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, pp. 1–10, 2016. doi: 10.1109/COMHPC.2016.006.

Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., Law, J., Lee, K., Lu, J., Noordhuis, P., Smelyanskiy, M., Xiong, L., and Wang, X. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 620–629, 2018. doi: 10.1109/HPCA.2018.00059.

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., and Candela, J. Q. n. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, pp. 1–9, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329996. doi: 10.1145/2648584.2648589. URL https://doi.org/10.1145/2648584.2648589.

Ivchenko, D., Van Der Staay, D., Taylor, C., Liu, X., Feng, W., Kindi, R., Sudarshan, A., and Sefati, S. Torchrec: a pytorch domain library for recommendation systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 482–483, 2022.

Jiang, B., Deng, C., Yi, H., Hu, Z., Zhou, G., Zheng, Y., Huang, S., Guo, X., Wang, D., Song, Y., et al. Xdl: an industrial deep learning framework for high-dimensional sparse data. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, pp. 1–9, 2019.

Kang, W.-C., Cheng, D. Z., Yao, T., Yi, X., Chen, T., Hong, L., and Chi, E. H. Learning to embed categorical features without embedding tables for recommendation. *arXiv preprint arXiv:2010.10784*, 2020.

Khudia, D., Huang, J., Basu, P., Deng, S., Liu, H., Park, J., and Smelyanskiy, M. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., and Sun, G. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1754–1763, 2018.

Lian, X., Yuan, B., Zhu, X., Wang, Y., He, Y., Wu, H., Sun, L., Lyu, H., Liu, C., Dong, X., et al. Persia: A hybrid system scaling deep learning based recommenders up to 100 trillion parameters. *arXiv preprint arXiv:2111.05897*, 2021.

Liu, M., Luo, L., Nelson, J., Ceze, L., Krishnamurthy, A., and Atreya, K. Incbricks: Toward in-network computation with an in-network cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 795–809, 2017.

Luo, L., Nelson, J., Ceze, L., Phanishayee, A., and Krishnamurthy, A. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 41–54, 2018.

Luo, L., West, P., Krishnamurthy, A., and Ceze, L. Srifty: Swift and thrifty distributed training on the cloud, 2020a. URL https://arxiv.org/abs/2011.14243.

Luo, L., West, P., Nelson, J., Krishnamurthy, A., and Ceze, L. Plink: Discovering and exploiting locality for accelerated distributed training on the public cloud. *Proceedings of Machine Learning and Systems*, 2:82–97, 2020b.

Mandakolathur, K. and Jeaugey, S. Doubling all2all performance with nvidia collective communication library 2.12 — nvidia technical blog. https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/, 2022.

Morgan, T. P. The iron that will drive ai at meta platforms - the next platform. https://www.nextplatform.com/2022/10/20/the-iron-that-will-drive-ai-at-meta-platforms/, 2023.

Mudigere, D., Hao, Y., Huang, J., Jia, Z., Tulloch, A., Sridharan, S., Liu, X., Ozdal, M., Nie, J., Park, J., Luo, L., et al. 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*, pp. 993–1011, 2022.

Narang, S., Chung, H. W., Tay, Y., Fedus, W., Fevry, T., Matena, M., Malkan, K., Fiedel, N., Shazeer, N., Lan, Z., et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

Naumov, M., Kim, J., Mudigere, D., Sridharan, S., Wang, X., Zhao, W., Yilmaz, S., Kim, C., Yuen, H., Ozdal, M., Nair, K., Gao, I., Su, B.-Y., Yang, J., and Smelyanskiy, M. Deep learning training in facebook data centers: Design of scale-up and scale-out systems, 2020.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*, pp. 18332–18346. PMLR, 2022.

Ramesh, B., Suresh, K. K., Sarkauskas, N., Bayatpour, M., Hashmi, J. M., Subramoni, H., and Panda, D. K. Scalable mpi collectives using sharp: Large scale performance evaluation on the tacc frontera system. In *2020 Workshop on Exascale MPI (ExaMPI)*, pp. 11–20, 2020. doi: 10.1109/ExaMPI52011.2020.00007.

Rashidi, S., Denton, M., Sridharan, S., Srinivasan, S., Suresh, A., Nie, J., and Krishna, T. Enabling compute-communication overlap in distributed deep learning training platforms. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 540–553, 2021. doi: 10.1109/ISCA52012.2021.00049.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

Sapio, A., Canini, M., Ho, C.-Y., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D. R., and Richtárik, P. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

Smelyanskiy, M. Zion: Facebook next- generation large memory training platform. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–22, 2019. doi: 10.1109/HOTCHIPS.2019.8875650.

Tarnawski, J. M., Narayanan, D., and Phanishayee, A. Piper: Multidimensional planner for dnn parallelization. *Advances in Neural Information Processing Systems*, 34, 2021.

Thangakrishnan, I., Cavdar, D., Karakus, C., Ghai, P., Selivonchyk, Y., and Pruce, C. Herring: Rethinking the parameter server at scale for the cloud. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

TorchRec, F. R. dlrm/torchrec_dlrm at main · facebookresearch/dlrm. https://github.com/facebookresearch/dlrm/tree/main/torchrec_dlrm/, 2023.

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., and Chi, E. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pp. 1785–1797, 2021.

Wikipedia. Minimum k-cut - wikipedia. https://en.wikipedia.org/wiki/Minimum_k-cut, 2023.

Yang, J. A., Park, J., Sridharan, S., and Tang, P. T. P. Training deep learning recommendation model with quantized collective communications. In *Proceedings of the 3rd international workshop on deep learning practice for high-dimensional sparse data*, 2021.

Zha, D., Feng, L., Luo, L., Bhushanam, B., Liu, Z., Hu, Y., Nie, J., Huang, Y., Tian, Y., Kejariwal, A., et al. Pre-train and search: Efficient embedding table sharding with pre-trained neural cost models. *Proceedings of Machine Learning and Systems*, 5, 2023.

Zhang, B., Luo, L., Liu, X., Li, J., Chen, Z., Zhang, W., Wei, X., Hao, Y., Tsang, M., Wang, W., Liu, Y., Li, H., Badr, Y., Park, J., Yang, J., Mudigere, D., and Wen, E. Dhen: A deep and hierarchical ensemble network for large-scale click-through rate prediction, 2022a. URL https://arxiv.org/abs/2203.11014.

Zhang, Z., Zheng, S., Wang, Y., Chiu, J., Karypis, G., Chilimbi, T., Li, M., and Jin, X. Mics: Near-linear scaling for training gigantic model on public cloud. *arXiv preprint arXiv:2205.00119*, 2022b.

Zhao, W., Zhang, J., Xie, D., Qian, Y., Jia, R., and Li, P. Aibox: Ctr prediction model training on a single node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 319–328, 2019.

Zhao, W., Jiao, X., Hu, M., Li, X., Zhang, X., and Li, P. Communication-efficient terabyte-scale model training framework for online advertising. *arXiv preprint arXiv:2201.05500*, 2022.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Gonzalez, J. E., et al. Alpa: Automating inter-and intra-operator parallelism for distributed deep learning. *arXiv preprint arXiv:2201.12023*, 2022.

Zhou, H., Zhou, X., Zeng, Z., Zhang, L., and Shen, Z. A comprehensive survey on multimodal recommender systems: Taxonomy, evaluation, and future directions. *arXiv preprint arXiv:2302.04473*, 2023.