Herald: An Embedding Scheduler for Distributed Embedding Model Training

Chaoliang Zeng Xiaodian Cheng Han Tian Hao Wang Kai Chen
Hong Kong University of Science and Technology

ABSTRACT

Given the ability to represent categorical features, embedding models have gained great success on many internet services. State-of-the-art training frameworks enable embedding cache in GPU workers to benefit from hardware acceleration while supporting massive category representations (embeddings) in the limited-capacity GPU device memory. However, based on our measurements, naively adopting a cache system in embedding model training leads to non-negligible communications overhead between caches and the global parameter server. We observe that many such communications are avoidable, given the predictability and sparsity natures of embedding cache accesses in distributed training.

In this paper, we propose Herald, a runtime embedding scheduler that significantly reduces the cache overhead by leveraging information about the required embeddings in the input samples and the locations of those embeddings. Herald is composed of two key optimizations: It allocates samples in a training batch to proper workers for a high cache hit rate via a heuristic location-aware inputs partition mechanism, and applies an on-demand synchronization strategy for a low frequency of embedding synchronization. Preliminary simulation results show that Herald can reduce cache overhead by 39.3%-53.7% compared to a naive cache-enabled training system across different realistic datasets.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**;

KEYWORDS

Distributed Training, Embedding

APNet 2022, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07…$15.00
https://doi.org/10.1145/3542637.3542645

ACM Reference Format:
Chaoliang Zeng Xiaodian Cheng Han Tian Hao Wang Kai Chen . 2022. Herald: An Embedding Scheduler for Distributed Embedding Model Training. In *6th Asia-Pacific Workshop on Networking (APNet 2022), July 1–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3542637.3542645

1 INTRODUCTION

Deep learning models with embedding technique [4] are an important class of machine learning algorithms that models categorical features (e.g., words and product properties) with continuous semantic embedding vectors (or embeddings for short). These embedding models enable a wide range of internet services, e.g., information retrieval [12, 15], recommendation system [6, 7, 21], and natural language processing [11, 28, 30], consuming significant infrastructure capacity and compute cycles across production datacenters [24].

However, given a large mismatch between the limited memory capacity provided by a GPU and the memory requirement (tens of GBs to TBs [22, 31–34]) of embeddings, training embedding models with GPU acceleration is challenging. Hence, a recent line of work [3, 23] proposes cache-enabled training frameworks, which adopt the parameter server (PS) [20] framework to maintain globally shared embeddings and accelerate the embedding lookup operations by caching the hot embedding in GPU memory locally. Despite being promising, naively adopting a cache system in the distributed embedding model training still suffers from significant communications overhead between caches and PS. These communications are bidirectional, including cache pull when a cache does not hit the required embedding with the latest version[1], and cache push when a cache evicts or synchronizes an updated embedding.

Existing work optimizes the above cache overhead with extra limitations or model accuracy degradation. FAE [3] reduces the number of cache pull by intentionally training hot inputs that contain entirely hot embeddings, which requires sampling on the dataset to identify the hot embeddings and hot inputs. When training a large volume of streaming samples, pre-processing the dataset is difficult, if not impossible. HET [23] mitigates the overhead of both cache pull and cache push by applying a staleness-tolerant embedding

[1] In the rest of the paper, we refer to hitting the latest version of an embedding in the cache as a cache hit for short.

update method. Both solutions deviate from the original training process and do not provide any theoretical guarantee of model accuracy. However, model accuracy is important in production. For example, an order of 0.1% model accuracy loss may be intolerable in Facebook recommendations [2]. Therefore, synchronous training without a bias on training samples is widely incorporated into large-scale deep learning training systems [24].

To reduce the cache overhead without compromising model accuracy, we observe two critical characteristics of embedding cache accesses during the embedding model training: *predictability* and *sparsity*. Specifically, since the embeddings required by training samples and the current cache snapshot of each worker are visible before the computation, the incoming cache accesses and their results (hit or not) are predictable under a partition of batch inputs. Moreover, most in-cache embeddings are sparse enough that the number of training samples containing a specific embedding is less than the total number of training samples allocated to a worker in a typical distributed training setting. These two characteristics indicate that it has the potential to train an embedding in a fixed worker to increase the cache hit rate and avoid unnecessary cache updates.

Based on the above observations, we believe that a domain-specific training scheduler should be explored for efficient embedding model training. Such a scheduler should fully leverage information, including the required embeddings of input samples and the locations of those embeddings, to reduce the cache overhead. To this end, we propose Herald, an embedding scheduler for real-time batch inputs partitions and cache update decisions. Herald consists of two key components: 1) a heuristic location-aware inputs partition mechanism that allocates samples in a training batch to proper workers for a high cache hit rate, and 2) an on-demand synchronization strategy for a low frequency of embedding synchronization. These two optimizations work together to reduce embedding communications. Our theoretical analysis proves that accelerating embedding model training with Herald can preserve the model consistency and hence the same model accuracy as synchronous training.

Our preliminary simulation results with realistic datasets show that Herald can reduce the cache overhead by 39.3%-53.7% compared to a naive cache-enabled training system.

2 BACKGROUND & MOTIVATION

2.1 Overview of Embedding Models

The left part of Figure 1 demonstrates a typical embedding model architecture. The training samples of an embedding model contain dense continuous features and sparse categorical features. An embedding model leverages embedding tables to project sparse features into dense representations.

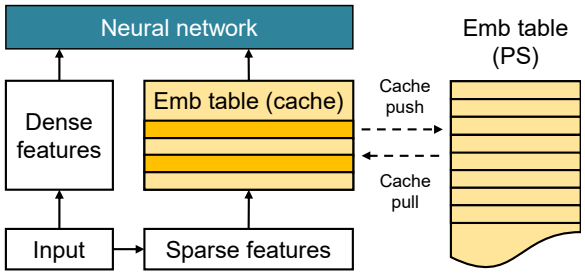


Figure 1: A typical architecture of cache-enabled embedding models.

Specifically, each embedding lookup may be interpreted as using a one-hot vector, where only the i-th position is 1 for the lookup of the i-th category, to obtain the corresponding row dense vector (embedding) of the embedding table. All embeddings of sparse features will be further reduced into a single vector and fed into the neural network along with dense features. In a typical embedding model, the embedding table contributes a large portion of model trainable parameters.

Traditional training frameworks [1, 5, 17, 25] are not well-suited to embedding model training, as they pay little attention to the embedding table. With the scaling up of embedding table size, storing the whole embedding table to a (GPU) worker for every use becomes difficult. Therefore, state-of-the-art training frameworks for embedding models manage embeddings on dedicated parameter servers [2, 16], and leverage embedding cache to reduce the communication of remote table lookups [3, 23], as shown in Figure 1.

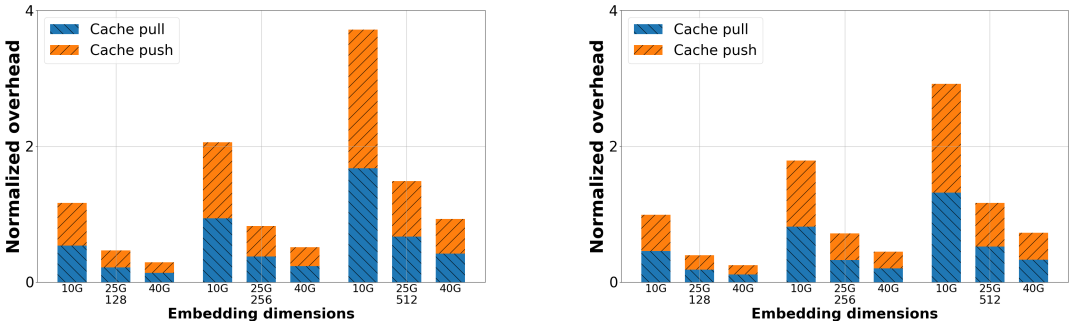
2.2 A Naive Cache System Is Not Enough

However, we find that embedding communication raised by embedding cache still contributes significant overhead compared to the training computation. We study the cache pull/push behaviors with a simulation on Criteo Kaggle dataset [9] to show this problem. More experiment settings can be found in §4.

The cache pull/push can be caused by either cache miss or cache update. When there is a cache miss, the cache will pull the required embedding and push an evicted embedding if necessary. For cache update, the cache will push the embedding updated by itself and pull the required embedding with the latest version updated by other caches.

Cache overhead matters. Figure 2 demonstrates the normalized cache overhead (the ratio of the embedding communications time to the model computation time) across different network settings and embedding sizes. It shows a similar trend between WDL [8] and DFM [14] models.

Overall, cache pull/push consume 0.29×-3.72× (0.25×-2.92×) computation time in WDL (DFM) model, where cache pull and cache push contribute the similar overhead (less than



(a) WDL. (b) DFM.

Figure 2: Normalized cache overhead in a naive cache system.

20% variance). When increasing individual embedding size, we have the following two observations. 1) Increasing embedding size does not significantly affect the total number of cache pull/push. We further break down the cache overhead and observe that cache updates contribute 78.5%-85.9% embedding communications. 2) When increasing the embedding size, the time for a single embedding communication scales linearly, while the model computation time increases sublinearly. As a result, the cache overhead is more serious in a large embedding size. Increasing the network bandwidth can mitigate the cache overhead, but network bandwidth increments in datacenters (4×-10× increments over the past few years) fail to catch up with GPU evolution (10×-20× faster computation from Nvidia V100 released in 2017 to Nvidia A100 released in 2020). The normalized cache overhead will further increase in this hardware evolution trend.

2.3 Opportunities and Observations

The cache overhead results from embedding communications in per-iteration embedding accesses and updates. Instead of tapping the performance limit of embedding communications [13, 19, 26], we take one step back and ask: Can we reduce the number of cache pull/push during the training? To give a positive answer, we first go through opportunities to reduce embedding communications during the training, and discuss the potential to adopt these optimizations based on two characteristics of embedding cache accesses, i.e., predictability and sparsity.

In the forward propagation, a required embedding hits the cache can prevent cache pull and potential cache push due to a cache eviction caused by inserting new embedding. Meanwhile, accessing an embedding in the forward propagation will incur a corresponding update in the backward propagation. However, synchronizing every embedding update is unnecessary, even in synchronous training. The reason is that an embedding is related to a sparse feature, and this feature is not necessarily trained in the following iterations by other workers. It may happen in two ways: 1) this feature does not appear in the later training samples, or 2) this feature is only trained by the same worker. In other words, the updated embeddings can only be synchronized when they are required by other workers in the following training. Putting all this together, we can reduce embedding communications by serving as much as possible in-cache embeddings and performing on-demand synchronizations.

We have the potential to leverage the above optimization opportunities based on the following two observations.

Predictability. There are two prerequisites for the above optimizations: knowing current cache snapshots as well as predicting and determining future embedding accesses. Fortunately, both of them are achievable. Since modern training frameworks decouple the training computation and the input preparation, which is performed in a data loader, we can foresee and decide possible embedding accesses as early as input generation for each worker. Meanwhile, maintaining cache snapshots in the data loader is trivial. Therefore, a designed data loader with an embedding scheduler can allocate input samples to workers based on their in-cache embeddings and determine embedding dependencies among iterations for on-demand synchronizations.

Sparsity. We can reduce possibly maximal embedding communications from the above optimizations, when the training workload of any of the in-cache embeddings can be accepted by only one worker, i.e., *the number of training samples containing this embedding is less than the total number of samples trained by a worker during the whole training cycle.* To measure the sparsity of the in-cache embeddings, we make profiling on real-world datasets. We consider embeddings with a frequency larger than 10^{-5} as cached and measure the number of sparse embeddings whose workload can be served by a single worker. At typical scales of training clusters (8-120 workers), Criteo Kaggle dataset contains 99.70%-99.96% of in-cache embeddings that have the potential to be trained in only one worker, and we observe similar results in other datasets (e.g., Avazu [18] and Criteo Search [27]).

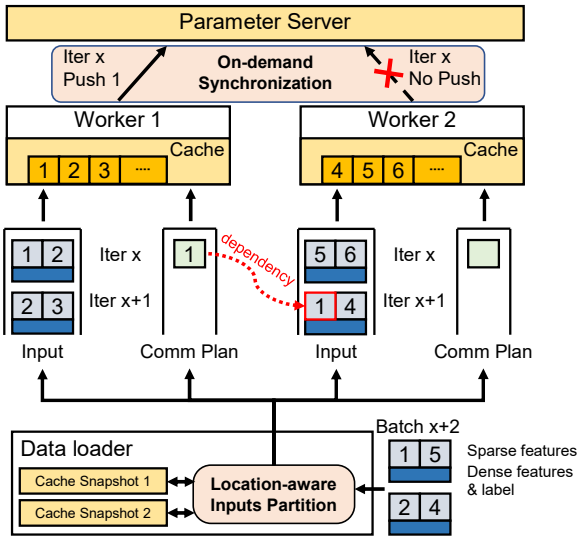


Figure 3: Herald accelerates embedding model training with a location-aware inputs partition mechanism and an on-demand synchronization strategy.

3 DESIGN

3.1 Overview

Based on the above ideas, we design a real-time embedding scheduler called Herald to accelerate cache-enabled embedding model training frameworks, as shown in Figure 3. The key ideas of Herald are 1) a location-aware inputs partition mechanism and 2) an on-demand embedding synchronization strategy for updated embeddings followed by communication plans as explained below.

The inputs partition mechanism in the data loader will allocate the incoming batch inputs to proper workers based on current embedding locations (detailed in §3.2) and generate embedding dependencies of this partition. An embedding dependency appears when an embedding with the latest version is cached on a worker and there is another worker assigned to train this embedding in this iteration. Therefore, the worker caching the latest embedding should synchronize this embedding to PS before this iteration training begins. In this way, embedding dependencies of this iteration become communication plans of the last iteration. In every iteration, a worker receives both training samples and a communication plan at the beginning of computation, and synchronizes the listed embedding in the communication plan during the backward propagation. To generate communication plans of an iteration, we need to partition the latter batch inputs before this iteration starts, which is possible as prefetching the training samples is common in the training process. To realize the above location-aware scheduling, the data loader maintains the cache snapshot of each worker to provide information on embedding locations.

Algorithm 1: Location-aware Inputs Partition

input : Batch samples (*Inputs*), worker list (*Workers*), and current cache snapshots
output : Inputs partition, communication plans, and updated cache snapshots

1 **for** i *in Inputs* **do**
2 **for** w *in Workers* **do**
3 $score_{(i,w)} = |cache(w) \cap embs(i)|$;
4 **end**
5 **end**
6 Init all workers as available;
7 **for** i *in Inputs* **do**
8 Find worker w with the largest score among available workers;
9 Allocate i to w;
10 **if** $workload(w) == size(Inputs)/size(Workers)$ **then**
11 Mark w as unavailable;
12 **end**
13 **end**
14 **for** w *in Workers* **do**
15 $comm\_plan_w = embs(Inputs - alloc(w)) \cap cache(w)$;
16 **end**
17 Update cache snapshots based on the partition result;

Herald only controls the embedding update by pushing the latest embedding to PS. We rely on the cache consistency protocol in existing cache systems to pull the update from PS on individual worker cache.

3.2 Location-aware Inputs Partition

Since a brute-force search to find the optimal partition solution with minimal cache overhead is unrealistic in real-time, we design a heuristic partition algorithm as shown in Algorithm 1.

In Algorithm 1, we first measure a score between every input sample and worker (Line 3). The score is defined as the number of embeddings that exist in the worker cache (with the latest version) and are required by the input sample simultaneously. Then, we allocate each input to the worker with the highest score (Line 8-9) while ensuring evenly distributed workloads among workers (Line 10-12). Given the partition result, we generate communication plans, i.e., a list of embeddings to be synchronized, for the last iteration with a basic idea that as long as the worker cache contains the latest embeddings required by other workers, this embedding is inserted into the worker's communication plan (Line 15). As discussed in §3.1, communication plans generated in batch $i + 1$ can control the synchronization behavior

in iteration i with the inputs prefetching. Finally, we update cache snapshots based on the partition result (Line 17).

3.3 Model Consistency Analysis

In this part, we show that the training process will not be affected by choice of input partition algorithm (either a naive partition or a location-aware partition) under bulk synchronous parallelism (BSP). Considering a parameter optimizer followed by stochastic gradient descent algorithm (SGD), the gradient calculation for model weights w on a given batch of n training samples is as follows:

$$\nabla w = \frac{1}{n}\Sigma_{i=1}^{n}\frac{\partial L(x_i, w)}{\partial w}, \tag{1}$$

where x_i is the i-th training sample of the batch, and L is the loss function. Based on Equation 1, the gradient of the batch is the sum of the individual gradient of each training sample in the batch. Since the individual gradient depends on sample features and current model weights, which are synchronized before the gradient calculation for each iteration under BSP, partitioning the batch into m mini-batches take no effect on the gradient result:

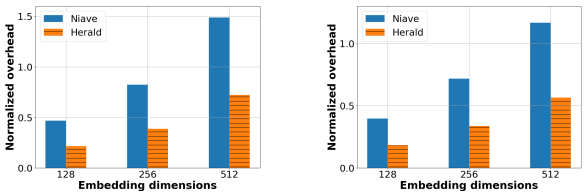
$$\frac{1}{n}\Sigma_{i=1}^{n}\frac{\partial L(x_i, w)}{\partial w} = \frac{1}{n}\Sigma_{i=1}^{m}\Sigma_{j=1}^{n/m}\frac{\partial L(x_{ij}, w)}{\partial w}, \tag{2}$$

where x_{ij} is the j-th training sample in the i-th mini-batch (worker). Therefore, any partition result generated by any partition algorithm will preserve the same gradients in BSP, and finally converge to the same model.

4 PRELIMINARY RESULTS

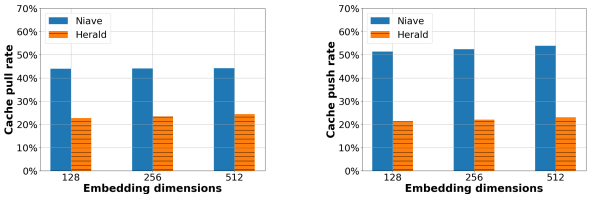
In this section, we present the preliminary simulation results for evaluating the performance improvement when applying Herald. We first have a performance deep dive on Herald with Criteo Kaggle dataset [9]. Then, we extend our simulation to other representative datasets and show that Herald can preserve the performance superiority.

Experiment settings. There are 8 LRU cache instances (workers), each of which has a 1.6 GB capacity (10% of 16 GB, a typical memory size of GPU). The dataset is evenly partitioned into these 8 cache instances, with a mini-batch size of 128. In a naive cache-enable training system, the batch samples are allocated workers sequentially. We record cache push and cache pull behaviors to evaluate the cache overhead. We measure the computation time of embedding models on HET [23] with an Nvidia V100 GPU, where we make all cache accesses hit to eliminate the embedding communication overhead. The normalized overhead is measured by the ratio of the embedding communications time, which is calculated by $\frac{Transmitted\ embeddings\ in\ bytes}{Bandwidth}$, to the model computation time, and embeddings use `double` data type.



(a) WDL. (b) DFM.

Figure 4: Normalized cache overhead comparison in 25G network.



(a) Cache pull. (b) Cache push

Figure 5: Performance breakdown.

Optimization	Pull	Push	Overall
Naive	1	1	1
On-demand synchronizations	1	0.84	0.91
Location-aware input partition	0.52	0.85	0.69
Herald	0.52	0.42	0.46

Table 1: Breakdown of contribution by each optimization (embedding dimensions = 128).

Overall improvement. In general, applying Herald can reduce 51.5%-53.7% cache overhead. To illustrate the benefit of such an overhead reduction from the model training perspective, Figure 4 shows the normalized cache overhead in the 25G network. Herald can effectively reduce the normalized cache overhead from 0.47-1.49 (0.40-1.17) to 0.22-0.72 (0.18-0.57) in WDL [8] (DFM [14]) model.

Performance breakdown. We decompose the cache overhead into cache pull overhead and cache push overhead, as shown in Figure 5. Figure 5a shows that Herald can effectively reduce the cache pull rate by 44.7%-48.2%, and in Figure 5b, Herald improves the cache push by 57.1%-58.4%. To figure out the reasoning behind these improvements, we further break down the performance in terms of the contribution by each optimization, as shown in Table 1. We find that location-aware input partition contributes the major improvements. For cache pull, location-aware input partition allows embeddings required by inputs most likely hits caches. Meanwhile, the cache push performance is jointly optimized by location-aware input partition and on-demand synchronizations, where the designed partition mechanism reduces embedding dependencies while on-demand synchronizations make such dependencies reduction benefit to communications reduction.

Herald overhead. To benefit embedding model training in runtime with Herald, the time overhead of processing a single batch in Herald must be smaller than the training time of a batch, so that the overhead of Herald can be hidden by the pipeline. To leverage the multi-core feature in the CPU, we accelerate Herald by OpenMP [10] with no more than 8 threads. We measure the average time consumption of parallelized Herald to be less than 10 ms per batch (and the brute-force search to be the order of minutes) on Intel(R) Xeon(R) Gold 5115 CPU. Meanwhile, we measure a minimalist distributed training system consisting of two GPUs connected via PCIe, where WDL and DFM models take more than 10 ms to train a batch. It indicates that Herald will not be the bottleneck in embedding model training.

Performance on other datasets. We take an extended evaluation with the same settings as Figure 4 on other real-world datasets: Avazu [18] and Criteo Search [27]. Herald can consistently reduce the cache overhead. It improves the performance by 39.5%-42.7% and 39.3%-44.0% on Avazu and Criteo Search datasets, respectively.

5 DISCUSSION AND FUTURE WORK

Optimization on cache replacement. The vision of Herald is to let all individual embeddings fixed to a particular worker for training, thus reducing the communication overhead. It requires that an embedding is not only accessed by a fixed worker (achieved by location-aware input partition), but also not evicted from the worker cache. Although this paper pays little attention to the cache replacement policy, it is possible to make an optimized cache replacement decision by considering the following embedding requirements in later iterations.

Prefetching communication plan. In an iteration, workers may execute different sizes of communication plans. This kind of work imbalance will result in idle workers during synchronization. To address this problem, workers transmitting fewer embeddings can prefetch and execute the communication plans in later iterations. However, it requires additional embedding dependency checking. As long as an embedding is not trained within the current iteration to the iteration of the checked communication plan, this embedding can be synchronized as early as the current iteration. Based on this idea, the data loader can re-schedule communication plans to balance the communication workload among workers.

Point-to-point (P2P) embedding synchronization. In this paper, we follow a distributed cache model as same as HET [23], where each cache only communicates with PS. In this cache model, an embedding synchronization requires at least two steps, one cache push and one cache pull. We can reduce the synchronization path by P2P synchronization between two workers. Moreover, P2P embedding synchronization can eliminate the potential network bottleneck caused by the PS architecture [29]. Herald can support P2P embedding synchronization by introducing receiving communication plans, which list the embeddings that workers should receive in an iteration.

6 RELATED WORK

FAE [3] and HET [23] are two cache-enabled embedding model training frameworks. In general, they leverage the skewness feature of datasets to accelerate embedding accesses with high popularity, while Herald further identifies the sparsity feature among those cached embeddings.

FAE maintains a uniform cache among all workers and synchronizes all cached embeddings as dense weights in every iteration. To reduce cache miss, FAE proposes a hot-embedding aware data layout for dataset pre-processing to identify both hot embeddings and hot inputs, which contain only hot embeddings. Moreover, FAE determines whether the training batch contains only hot inputs based on the testing loss. On the contrary, Herald does not need the prior knowledge of the hot embeddings before scheduling, and does not intervene in the batch formation but intelligently partitions a batch into mini-batch, which can preserve model consistency.

HET applies a distributed cache model. To reduce cache overhead, it supports staleness for both cache read and cache write operations, which will harm the model accuracy. This optimization is orthogonal to Herald. If the compromise on the model performance is acceptable, Herald can benefit from the same staleness operations to further reduce embedding communications.

7 CONCLUSION

This paper presents Herald, a runtime embedding scheduler for efficient cache-enabled embedding model training. By leveraging characteristics of predictability and sparsity existing in embedding cache accesses, Herald applies a location-aware inputs partition mechanism and an on-demand synchronization strategy to reduce cache communications during the training. Preliminary simulation results show that Herald can significantly reduce the cache overhead across different realistic datasets. In the next, we will integrate our solution to a training framework and evaluate the end-to-end training improvement.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their insightful comments. This work is supported in part by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001) and the Hong Kong RGC TRS T41-603/20-R, GRF 16213621 and GRF 16215119.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*.

[2] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. 2021. Understanding training efficiency of deep learning recommendation models at scale. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[3] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J Nair. 2021. Accelerating recommendation system training by leveraging popular choices. *Proceedings of the VLDB Endowment* (2021).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* (2013).

[5] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*.

[9] CriteoLabs. [n. d.]. Criteo display ad challenge. https://www.kaggle.com/c/criteodisplay-ad-challenge. ([n. d.]).

[10] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* (1998).

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. 2019. MOBIUS: towards the next generation of query-ad matching in baidu's sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[13] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. 2021. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*.

[15] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[16] Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, et al. 2019. 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*.

[17] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*.

[18] Kaggle. [n. d.]. Avazu mobile ads ctr. https://www.kaggle.com/c/avazu-ctr-prediction. ([n. d.]).

[19] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. 2019. Parallax: Sparsity-aware data parallel training of deep neural networks. In *Proceedings of the Fourteenth EuroSys Conference 2019*.

[20] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*.

[21] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-Based Product Retrieval in Taobao Search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

[22] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2021. Understanding capacity-driven scale-out neural recommendation inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE.

[23] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2021. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *Proceedings of the VLDB Endowment* (2021).

[24] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, et al. 2020. Deep learning training in facebook data centers: Design of scale-up and scale-out systems. *arXiv preprint arXiv:2003.09518* (2020).

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* (2019).

[26] Cèdric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. 2019. SparCML: High-performance sparse communication for machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[27] Marcelo Tallis and Pranjul Yadav. 2018. Reacting to Variations in Product Demand: An Application for Conversion Rate (CR) Prediction in Sponsored Search. *arXiv preprint arXiv:1806.08211* (2018).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*.

[29] Xinchen Wan, Hong Zhang, Hao Wang, Shuihai Hu, Junxue Zhang, and Kai Chen. 2020. Rat-resilient allreduce tree for distributed machine learning. In *4th Asia-Pacific Workshop on Networking*.

[30] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[31] Xinyang Yi, Yi-Fan Chen, Sukriti Ramesh, Vinu Rajashekhar, Lichan Hong, Noah Fiedel, Nandini Seshadri, Lukasz Heldt, Xiang Wu, and Ed H Chi. 2018. Factorized deep retrieval and distributed tensorflow serving. In *ser. Conference on Machine Learning and Systems*.

[32] 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. *Proceedings of Machine Learning and Systems* (2020).

[33] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*.

[34] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*.