


# HECA: A Heterogeneous Embedding Caching Algorithm for Multi-GPU CTR Prediction Model Training

Shuangyuan Wu , Yuan Zhao  and Xuechun Chen 

Super Scientific Software Laboratory, Department of Computer Science and Technology, China University of Petroleum-Beijing, Beijing 102249, China; wsy@cup.edu.cn (S.W.); yuan.zhao@student.cup.edu.cn (Y.Z.); xuechun.chen@student.cup.edu.cn (X.C.)

**Abstract:** As DNN (Deep Neural Network) demonstrate exceptional capability in learning and feature representation, more researchers are applying them to recommendation systems. However, the memory requirements for recommendation models have surged, creating an urgent need for cost-efficient solutions. This paper presents HECA, a novel and cost-efficient hierarchical hybrid storage management method, which includes several technologies. First, all embedding parameters are stored on CPU, with only a critical subset, high-frequency embedding parameters, preferentially identified and consistently retained in GPU as cache. For low-frequency embedding parameters, an efficient cache replacement algorithm is employed. Second, a three-stage pipeline parallelism method is used to cover communication latency, supported by an embedding prefetching strategy. Third, due to the skewness and sparse irregularity in the access patterns of embedding tables, a DSF (Deduplicate Sparse Feature) operation is introduced, reducing data transfer volume between CPU and GPU. As a result, this approach enables training on a single GPU even when the embedding parameters scale to hundreds of GB or even TB. Experimental results show that the proposed method not only maintains model accuracy but also significantly improves system performance, achieving an average 2.8x speedup across various batch sizes and hardware platforms.

**Keywords:** Click-Through Rate; Recommendation System; CPU-GPU Hybrid Training; Embedding Cache

## 1. Introduction

With the explosive growth of online information, recommendation systems have become an important strategy for managing information overload [1]. Numerous technology companies develop their recommendation systems, recognizing its critical importance to their business success [2]. As deep learning technology continues to advance [3–5], their applications in recommendation systems have enabled these systems to gain a deeper understanding of the intentions underlying user behavior. Thus, recommendation models based on deep neural networks have become essential tools for personalized recommendation tasks [6].

Click-Through Rate (CTR) [7–9] is the number of clicks an ad receives divided by the number of times the ad is shown in applications such as advertising and recommendation systems. In deep learning recommendation systems, CTR prediction is a crucial task [10,11]. With the rapid growth of training data volume, model parameters have reached hundreds of GB or even TB, which cannot be stored in the HBM (usually only tens of GB) on a single GPU device.

To address this issue, traditional data parallelism methods are not effectively applicable [4]. When model parallelism is employed, multiple GPUs are required to partition the large embedding parameter table into smaller segments, which are then distributed across different GPUs for training. However, this approach presents several challenges. Firstly, it increases training costs significantly [12]. Secondly, CTR prediction's memory-intensive embedding

**Citation:** Wu, S.; Zhao, Y.; Chen, X. HECA: A Heterogeneous Embedding Caching Algorithm for Multi-GPU CTR Prediction Model Training. *Electronics* **2024**, *1*, 0. <https://doi.org/>

Received:

Revised:

Accepted:

Published:

**Copyright:** © 2024 by the authors. Submitted to *Electronics* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

tables require substantial storage but involve only a small part of computation, leading to inefficient GPU utilization.

During the training of each batch on GPU, only the embedding parameters corresponding to that batch and MLP parameters are needed. The memory required for these parameters is typically only hundreds of MB, which can be fully accommodated on a single GPU. Therefore, many researchers have proposed a hybrid training strategy using both CPU and GPU to address this issue. The basic idea is to use low-cost CPU memory to store the large embedding parameter table, while leveraging GPU's parallel computing power for computation-intensive neural network forward and backward propagation operations.

Although this method alleviates memory demands, it introduces new challenges. On one hand, transferring embedding parameters between CPU and GPU involves handling large volumes of data, making the process time-consuming. On the other hand, despite the rapid advancement in GPU parallel computing capability and bandwidth, the relatively lower bandwidth of CPU limits the overall performance of CPU-GPU hybrid system.

Some observations are as follows. First, recommendation systems datasets often exhibit a power-law distribution characteristic. Specifically, when accessing the embedding parameter table, some embedding vectors are accessed frequently, while some are rarely used. Second, the principle of locality suggests that storing frequently accessed embedding parameters on GPU effectively creates a cache for the large embedding parameter table on CPU. This approach places data closer to computation, thus accelerating the training process. Third, introducing caching and prefetching mechanisms may cause write-after-read issue, potentially leading to consistency problem. To achieve significant performance improvement while ensuring the convergence and correctness of experiment, it may be necessary to relax consistency requirement appropriately. Finally, the indices of the embeddings accessed in each iteration are often discontinuous and the access locations can be widely spaced apart, which leads to irregular memory access patterns.

Considering the aforementioned challenges and observations, this paper proposes **HECA**, a **H**eterogeneous **E**mboding **C**aching **A**lgorithm for irregular memory access based on CPU-GPU heterogeneous systems. By storing all embedding parameters on CPU and implementing a caching strategy on GPU, HECA enables the training of large CTR models on fewer GPUs or even a single GPU. Our key contributions include:

1. High-frequency embedding parameters are stored on GPU, and a software-based cache management method is proposed to support training with large-scale parameters ranging from hundreds of GB to TB.
2. The entire training process is divided into three stages: DataLoader, Collector, and Trainer. This three-stage pipeline parallelism method is constructed, which prefetches the embedding parameters needed for the next iteration while training the current iteration.
3. An optimization algorithm, DSF (Deduplicate Sparse Feature), is designed to eliminate redundant sparse features, ensuring that each embedding vector is transferred between CPU and GPU only once.

## 2. Related work

Although there has been a lot of optimization work in accelerating deep neural network training [4,13–15], research specifically focuses on accelerating deep recommendation systems training remains relatively scarce. Deep recommendation systems models typically involve computation-intensive deep neural network and memory-intensive embedding table. While the computational demands increase with the growth in training data, the rise is relatively moderate. Therefore, our research does not focus on accelerating the computation-intensive DNN components [16–21], but rather on optimizing the storage and access of large, memory-intensive embedding table formed by numerous categorical features, such as user IDs [22–24], item IDs [25–27], and ad IDs [11].

Several research efforts have explored about training large-scale embedding table. Persia [28] designs a hybrid training system where CPU acts as a parameter server, supporting the training of models with up to 100 trillion parameters. HET [29] proposes a consistency

model to ensure cache coherence, allowing staleness in the data read and write operations. XDL [30] focuses on the deep optimization of high-dimensional sparse data representation and computation tasks. DES [31] develops an equivalent replacement strategy to reduce communication costs. HierPS [32] designs a hierarchical storage management system for large-scale deep learning advertising systems. RecSSD [12] introduces a high-performance SSD storage solution specifically for neural recommendation inference, supporting the training of large parameter models. Mixed precision and caching methods for compressing model parameters are proposed by [33]. cDLRM [34] and ScratchPipe [35] employ proactive strategies to optimize embedding access and communication cost.

### 3. Background

#### 3.1. Deep CTR Model Architecture

CTR prediction models hold a critical position in computational advertising, as their accuracy directly impacts revenue growth for advertising companies. Driven by the demands of the internet market, the evolution of CTR models has undergone significant changes. Prior to 2010, traditional Logistic Regression (LR) methods were predominantly used for CTR prediction. Subsequently, models like Factorization Machines (FM) and Gradient Boosting Decision Trees (GBDT) became mainstream. The emergence of deep learning in 2015 marked the beginning of various innovative CTR model architectures. Traditional CTR models have laid an important foundation for deep learning models, and they can be viewed as the starting point and basis for modern deep learning-based CTR models.

With the rise of deep learning, neural network-based deep recommendation models have become essential tools for addressing personalized recommendation tasks[6,9]. Microsoft develops Deep & Cross Network[36], one of the most fundamental and classic models in deep learning-based CTR prediction, designed to enhance the click-through rate of ads in the Bing search engine. The recommendation team at Google App Store proposes the Wide & Deep[37] model, which combines linear and sparse features. Building on the Wide & Deep model, Huawei replaces the Wide component with Factorization Machines (FM) to develop the DeepFM[38] model. Subsequently, Google replaces the original Wide component with a Cross Network, developing the DCN[39] model. Researchers have also improved the Deep component by incorporating an Attention Network, resulting in the AFM[40] model. Currently, deep CTR models continue to be updated and iterated, leading to ongoing improvements in performance.

#### 3.2. Huge Embedding Table

Training data typically includes categorical values, such as user IDs and item IDs, which cannot be directly processed by computer. To handle these data effectively, neural networks introduce embedding layers that map sparse high-dimensional features into a lower-dimensional dense space, resulting in dense vectors. These categorical features are transformed into a dense two-dimensional tensor known as an embedding table, with dimensions  $N \times D$ , where  $N$  represents the number of features and  $D$  denotes the embedding dimension. This approach significantly reduces memory requirements compared to one-hot encoding method. However, in industrial applications, where  $N = 10^9 \sim 10^{11}$  and  $D = 10^4$ , the memory consumption of the embedding table continues to grow, reaching hundreds of GB or even TB. These data constitute the majority of parameters in deep CTR recommendation models, making it infeasible to store all of this data within a single High Bandwidth Memory (HBM).

#### 3.3. CPU-GPU Hybrid Storage Training

Modern GPUs are not only powerful graphic processors but also possess highly parallel computing capability, with peak computational performance and memory bandwidth significantly surpassing those of CPU [41]. Optimization techniques that rely solely on CPU or GPU have shown limited effectiveness. Thus, new technologies are required to

fully leverage the potential of heterogeneous system. As the use of CPU and GPU becomes more widespread, their distinct characteristics and advantages have become increasingly recognized, making CPU-GPU collaborative computing a necessary choice for achieving high-performance computing. This has also spurred extensive research into heterogeneous computing technologies [42].

CPU-GPU hybrid training involves storing the vast embedding tables in large-capacity CPU memory, while keeping a portion of the embedding as a cache on GPU, and performing both forward and backward computations on GPU.

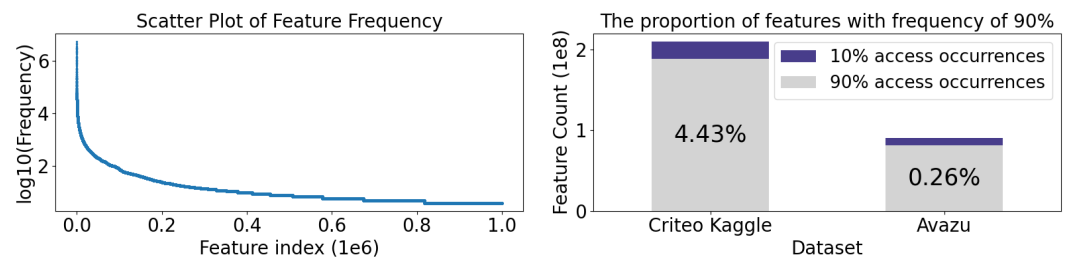
During the forward computation phase, embedding vectors corresponding to feature indices are first retrieved from the embedding table in CPU memory through a look-up process. These vectors are then transferred to GPU for further processing. Afterwards, the embedding vectors are fed to the deep neural network, where they pass through several FC layers, ReLU activation layers, and Dropout layers. This process then produces the prediction value. Finally, forward propagation is completed by computing the loss between the predicted value and the true label.

In the back propagation phase, the parameters of each layer need to be updated. Special attention is given to updating embedding parameters. Since the embedding involved in training for the current batch are already cached on GPU, hybrid training eliminates the gradient flow back to CPU, further minimizing the communication data volume and latency between CPU and GPU.

## 4. Motivation

### 4.1. Skewed Embedding Access Pattern

Power-law distribution have attracted widespread attention due to their mathematical properties and significant physical implications in various natural and human phenomena. Examples include city populations, earthquake magnitudes, and blackout scales, all of which are considered to follow power-law distributions [43]. Statistical analysis of a dataset reveals that features and their occurrence frequencies conform to a power-law distribution. As illustrated in Figure 1, the most frequent feature appears up to  $10^7$  times, while some features occur fewer than 10 times, with 4.43% of features accounting for 90% of the total data frequency. This skewed distribution presents valuable opportunity for optimization.



**Figure 1.** The left figure shows feature frequency. The right figure shows the proportion of features with 90% frequency in Criteo Kaggle and Avazu.

During the data loading phase, the DSF operation eliminates duplicate features within each batch, ensuring that each feature is transferred only once between CPU and GPU.

### 4.2. High Communication Latency

Due to the lower bandwidth of CPU, fetching embeddings and transferring them to GPU has become a performance bottleneck in CPU-GPU heterogeneous systems. To address this issue, our optimization strategy focuses on reducing the amount of memory accesses and data transfer volume to CPU. We employ a DSF operation to ensure that each feature and its corresponding embedding vector are transferred between CPU and GPU only once, thereby mitigating performance bottlenecks caused by CPU's limited bandwidth. Additionally, we implement a caching strategy on GPU, which includes static caching embedding for high-frequency features and dynamically managing the cache for

less frequent features. This dynamic management involves operations such as lookup, insertion, and eviction, with detailed explanations provided in the following sections.

#### 4.3. Staleness Tolerance

In our caching optimization approach, HECA employs a prefetching strategy for embedding parameters to reduce wait time and cover transfer delay as much as possible. However, it may bring write-after-read consistency issue. During the prefetching phase, when an embedding vector has been loaded into GPU cache, it will be retrieved directly from GPU cache. If this vector is involved in training during the current iteration and is updated after the iteration ends, the prefetched embedding vector may become outdated, leading to write-after-read consistency problem. HECA not only demonstrates that the model's loss trend remains accurate, but also show an improvement in system performance. Therefore, allowing some flexibility in consistency requirements, while still ensuring experimental convergence and accuracy, can improve system performance.

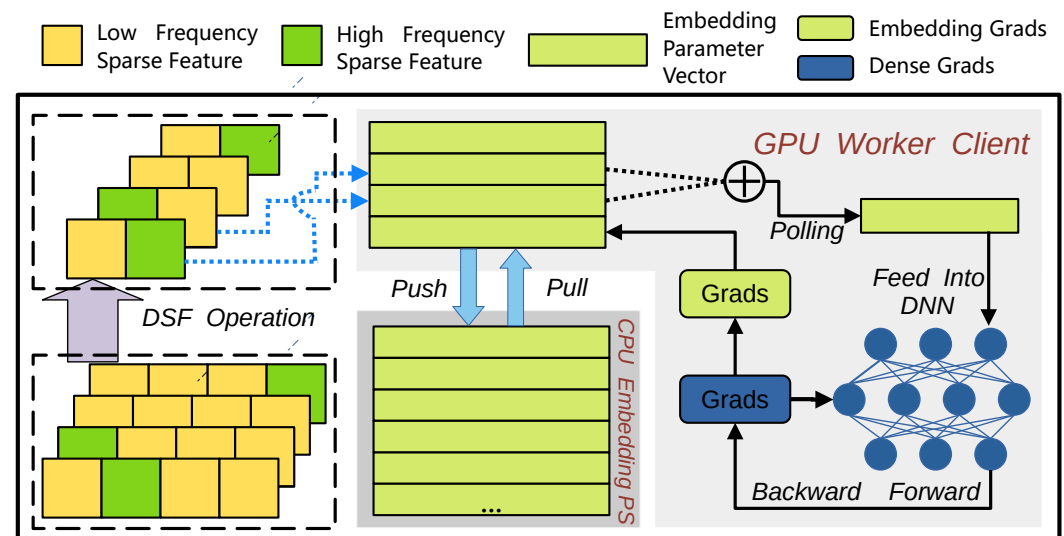
#### 4.4. Design Goal: Training At GPU Memory Speed

The final design goal is to decouple the massive memory requirement from the relatively slow growth of computation, aiming to train large deep CTR models with less resource, potentially even a single GPU, thereby avoiding increased training costs due to the large memory demand of model parameters. HECA leverages the strengths of both CPU and GPU processing, fully utilizing the capability of heterogeneous computing system while optimizing communication costs between CPU and GPU. By this way, the recommendation model can benefit from being trained entirely on the GPU while significantly reducing the memory cost of GPU.

### 5. Our work

#### 5.1. Overview

HECA introduces an innovative heterogeneous memory management approach that optimizes the CTR model by leveraging the principle of locality, enabling model training to be completed using only a single GPU. The overall architecture is illustrated in Figure



**Figure 2.** HECA's architecture includes DNN CPU-GPU hybrid training, DSF (Deduplicate Sparse Feature), cache management method.

2. The DSF operation deduplicates features, reducing the length of feature batches. The CPU acts as the embedding parameter server (PS), storing the global embedding parameter table, while the GPU functions as a cache for frequently accessed embedding entries and executes DNN training. Additional optimization techniques include constructing a three-stage pipeline, updating parameters using dense and embedding gradients during



training, prefetching embeddings for the next batch, and designing a cache structure with a dynamic cache management strategy within the pull and push modules. These techniques collectively reduce GPU memory requirements, as well as the data transfer volume and latency between the CPU and GPU, making the heterogeneous system both cost-effective and efficient.

### 5.2. Deduplication of Sparse Features

In the previous analysis, we highlighted that the sparse features in the training dataset exhibit a significant power-law distribution, a phenomenon commonly referred to as skewed access, where a small number of features account for the vast majority of all feature frequencies. However, the process of transferring these features from CPU to GPU often encounters substantial issues with redundant transfers due to this skewed distribution.

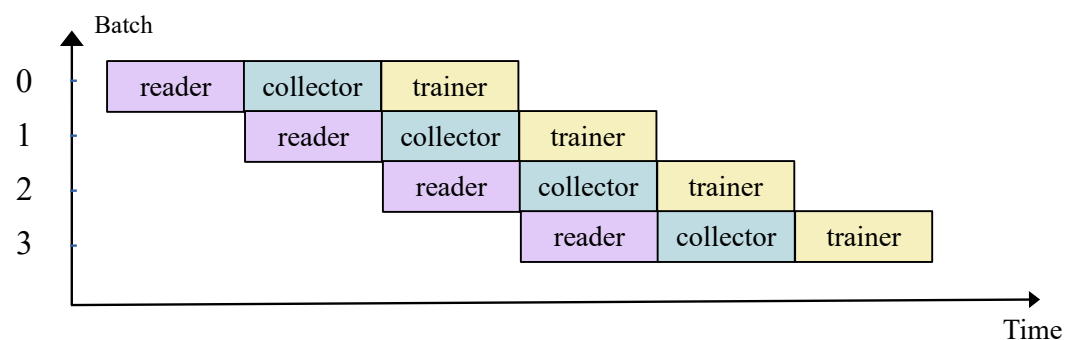
To address this problem, HECA rearrange and relocate the raw data during the dataset loading process and introduce two new tensors to store the processed information. The first tensor, *DeduplicateKeyId*, is used to store all unique features, while the second tensor, *GlobalKeyId*, records the occurrence order of these unique features. When a feature occurs again, only its index is transferred to the *DeduplicateKeyId* tensor instead of all its data. This process does not significantly increase the training time of the model. When a feature occurs again, only its index is transferred to the *DeduplicateKeyId* tensor instead of all its data. This process does not significantly increase the training time of the model. Moreover, feature Deduplication also positively impacts subsequent data prefetching. Each feature now requires only a single lookup, and the retrieved embeddings need to be transferred only once, thereby reducing unnecessary computation and data transfer volume.

### 5.3. Identification of Hot Features

The small subset of features with extremely high frequency, known as hot features, should be given special attention. Therefore, HECA preprocesses the static dataset by counting the frequency of each feature. By setting a threshold, we defined features with frequencies above this threshold as hot features. Although this operation is based on a small dataset, the feature distribution observed in the small dataset is considered consistent with that of the larger dataset due to the random sampling method.

### 5.4. Pipeline Parallelism

As presented in Figure 3, the entire training process is divided into three modules, including *dataLoader*, *collector*, and *trainer*. The *dataLoader* handles reading the dataset files. The *collector* transfers the tensors required for training from CPU to GPU, while the *trainer* executes the forward and backward propagation computations.

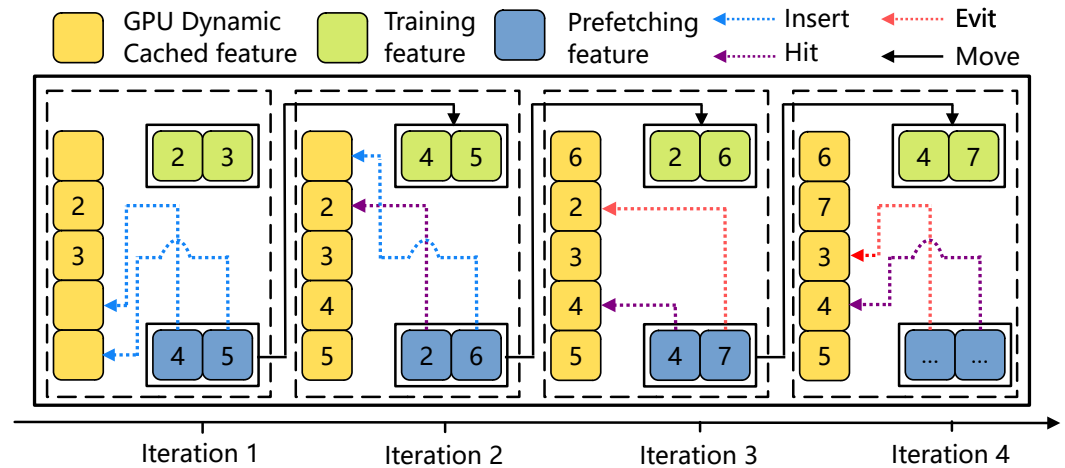


**Figure 3.** The entire training process is divided into three stages to cover non-training time.

### 5.5. Prefetch Operation

Informed prefetching and caching strategies, as outlined in [44], can significantly reduce the execution time of I/O-intensive applications. In traditional training methods, after the forward and backward computations for the current batch are completed, the

system waits to fetch all embedding entries for the next batch from CPU. This approach results in two main sources of waiting time: (1) the time required to look up embedding entries from the global embedding table on CPU, and (2) the time needed to transfer the retrieved entries to GPU via the slow PCI-e bus. These two waiting times occur in each iteration. As shown in Figure 4, during training, it is possible to parallelize the prefetching



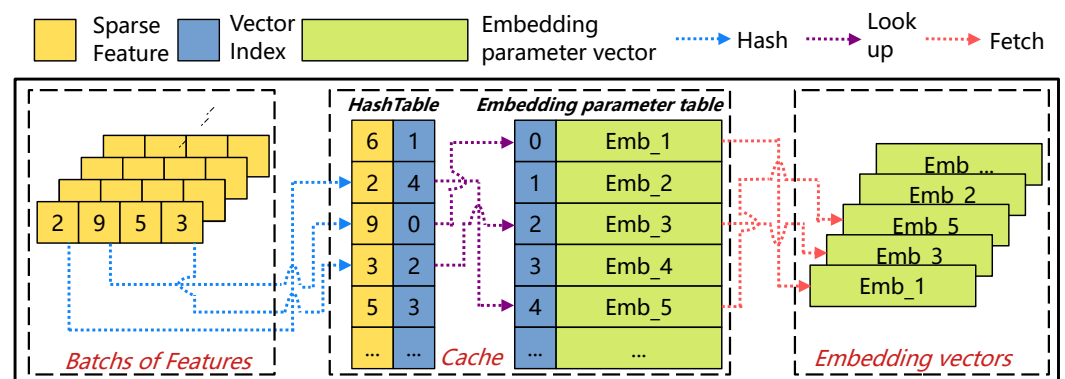
**Figure 4.** The figure shows the training and pre-fetching of 4 batches feature, including the insertion, eviction and hit operations.

of all embedding entries needed for the next batch while training the current batch. Given that the embedding data for the next batch is relatively small, and combined with the prior DSF operation, only unique embedding vectors need to be transferred. The prefetch operation coordinates with the cache management method to determine whether a feature is inserted, hit, or evicted. The move operation ensures that the prefetched features of the current batch will be used in the next batch for training. This significantly reduces the amount of data transferred and provides theoretical support for parallel prefetching. The prefetching process will be completed before the current batch training finishes to ensure a seamless transition to the next batch, making the training process appear as if it were completed at GPU memory speed.

During the training of the current batch, only the features of the next batch that are not in the current cache are prefetched. As a result, the transfer time is reduced and can be more easily covered by the training module.

### 5.6. Cache Structure Design

In the previous sections, we discussed the prefetching process in detail. In fact, prefetching and cache strategy are closely intertwined. According to the Figure 5, the cache



**Figure 5.** The Embedding cache consists of a hash table and an embedding parameter table.

structure primarily consists of a hash table and an embedding table stored in GPU.

	2080Ti	4090	A100
CPU	AMD Ryzen 7 3700X 8-Core Processor	12th Gen Intel(R) Core(TM) i5-12400F	Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz
GPU	NVIDIA GeForce RTX 2080 Ti-11GB	NVIDIA GeForce RTX 4090-24GB	NVIDIA A100-PCIE-40GB

**Table 1.** CPU and GPU information used in the experiment.

In this cache structure of HECA, the keys in the hash table store sparse features, while the values store the row indices of their embeddings entries. Since different keys may map to the same index position after hashing, conflict issues arise in cache management. The specific solution to these conflicts is described in Algorithm 1. By using the hash method, the required keys can be quickly located with an  $O(1)$  time complexity, allowing retrieval of the row index corresponding to the feature and subsequently the associated embedding vector. Additionally, the hash table performs parallel lookup of all features in the current batch on GPU, a process that does not introduce much time overhead.

### 5.7. Cache Insertion, Eviction and Hit

Before training begins, hot features and their corresponding embedding parameters are preloaded into GPU cache. This portion of the cache is static and will not be evicted. As previously mentioned, these hot features account for 90% of the total feature set. But is it sufficient to use them as the entire cache? The answer is clearly no. Less frequent features also need to be stored, requiring extra space for storing their embedding parameters to minimize communication overhead. This design keeps the cache area dynamic and active.

If all the features in a batch are hot features, the embeddings can be quickly retrieved. However, if there are cold features, it is necessary to check whether they are present in the cache. The specific search, insertion, and replacement operations are detailed in Algorithm 1.

#### Algorithm 1 Cache Algorithm

```

1: Input: key, Cache Hash Table (CHT), GPU Cache Embedding Table (T), Cache Size (CS), Current Batch (CB), Next Batch (NB)
2: index  $\leftarrow$  hash(key)
3: while CHT[index] is not empty and CHT[index]  $\neq$  key do
4:   index  $\leftarrow$  (index + 1) mod CS
5: end while
6: if CHT[index] == key then
7:   Retrieve embedding from T
8: else
9:   Insert key into CHT at index
10:  Retrieve embedding from CPU
11: end if
12: if CHT is full and CHT[index]  $\neq$  key then
13:   Search from index for a replacement candidate key1
14:   if key1 is not in CB, NB, and not a hot feature then
15:     Replace key1 with key in CHT, retrieve key's embedding from CPU, and replace key1's embedding in T
16:   end if
17: end if

```

## 6. Experiment

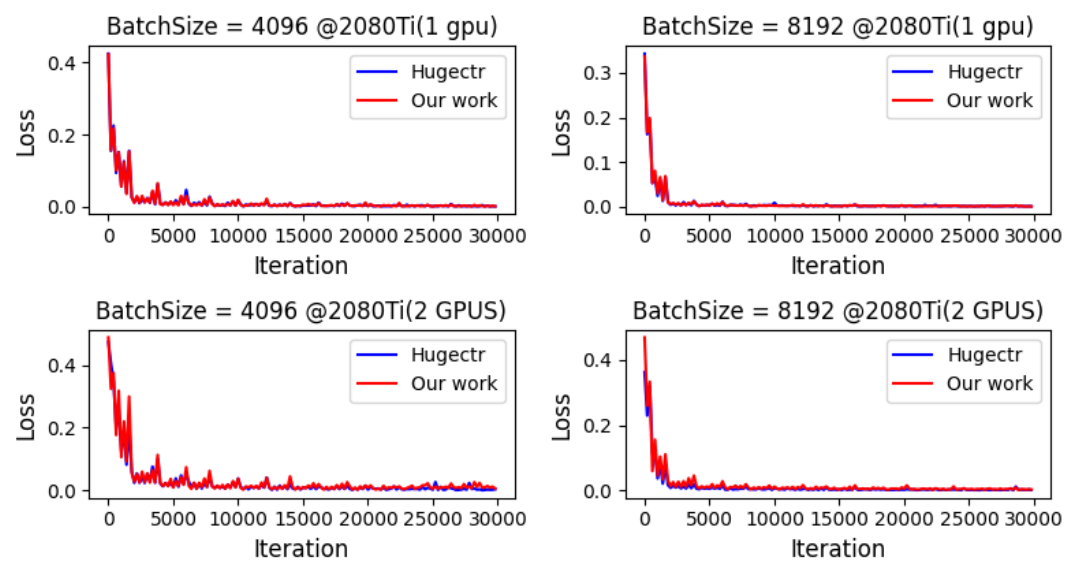
Table 1 shows the experimental platform and environment. The publicly available Criteo Kaggle dataset is used to evaluate the performance of the experiments. This dataset, generated from 24 consecutive days of user click logs by Criteo, contains a total of 45



million samples. Each sample includes 13 numerical features, 26 categorical features, and a label (0 or 1) indicating whether an advertisement is clicked. To prevent data reading from becoming a bottleneck during training, the dataset used in the experiments is preprocessed to enable faster program access.

Each sample includes 13 numerical features, 26 categorical features, and a label (0 or 1) indicating whether an advertisement is clicked. To prevent data reading from becoming a bottleneck during training, the dataset used in the experiments is preprocessed to enable faster program access.

The benchmark used for comparison is HugeCTR, an open-source framework developed by NVIDIA specifically for training and inference of large-scale deep learning recommendation systems models. HugeCTR supports various models, including the classic Wide & Deep model and the DeepFM model and so on. It is designed to fully leverage the powerful parallel computing capability of GPUs to accelerate the training processes of recommendation systems models.



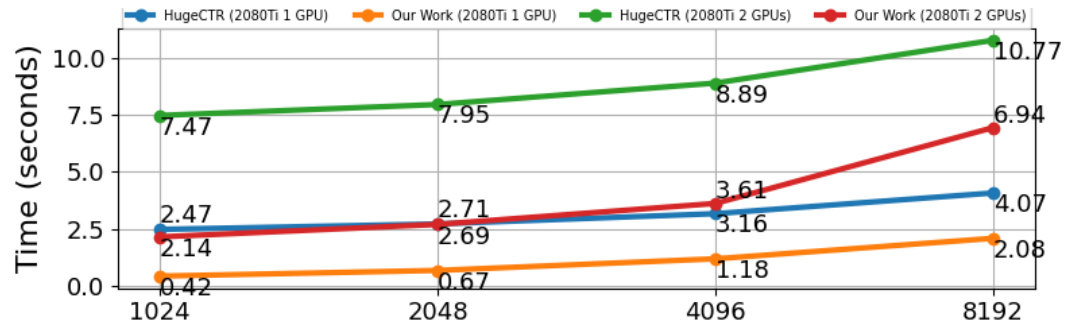
**Figure 6.** Loss reduction curve of 2080Ti single card and 2080Ti dual cards when batchsize = 4096 and 8192.

The embedding dimension is set to 96. A dimension that is too small may indicate insufficient capacity, while a dimension that is too large can lead to significant memory consumption. The number of iterations is set to 30000. In this study, we evaluate the performance of our optimized work on four different hardware platforms: a single NVIDIA GeForce RTX 2080 Ti, dual NVIDIA GeForce RTX 2080 Ti, a single NVIDIA A100, and a single NVIDIA GeForce RTX 4090. The aim is to assess the improvement in training time after optimization when compared to HugeCTR. We test the model under batch sizes of 1024, 2048, 4096, and 8192. For each combination of batch size and hardware platform, we record the training time in seconds for both the HugeCTR and our work. Figure 6 shows that the loss curves of our proposed method closely match those of HugeCTR, with both exhibiting similar convergence speeds. It fully demonstrates the effectiveness and correctness of HECA.

At the same time, we record the training time under different batch sizes and hardware platforms, as shown in Table 2 and Figure 7. On the single 2080Ti GPU, HugeCTR training time increases from 2.47 seconds for a batch size of 1024 to 4.07 seconds for a batch size of 8192. With our optimized work, the training time is significantly reduced, ranging from 0.42 seconds to 2.08 seconds across different batch sizes. This represents a marked improvement, especially at larger batch sizes. For the dual 2080Ti setup, HugeCTR performs more slowly, with training time ranging from 7.47 seconds to 10.77 seconds. While, HECA reduces the training time to a range between 2.14 seconds and 6.94 seconds, demonstrating effective scalability across multiple GPUs. On the A100, HugeCTR is faster than on the 2080Ti, taking

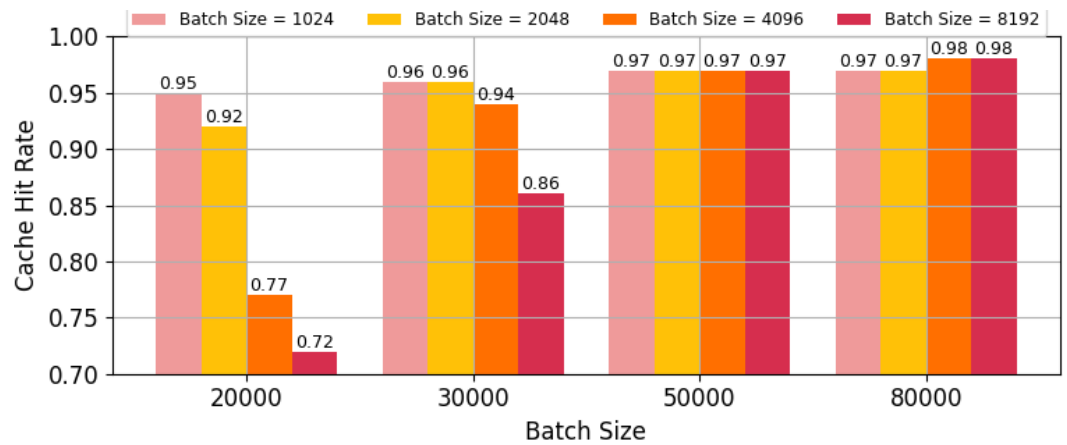
	HugeCTR	Our work	HugeCTR	Our work	HugeCTR	Our work	HugeCTR	Our work
Batch_size	1024		2048		4096		8192	
2080Ti(1 GPU)	2.47	0.42	2.71	0.67	3.16	1.18	4.07	2.08
2080Ti(2 GPUs)	7.47	2.14	7.95	2.69	8.89	3.61	10.77	6.94
4090(1 GPU)	1.12	0.36	1.27	0.66	1.61	0.88	2.01	1.68
A100(1 GPU)	0.86	0.21	0.95	0.28	1.17	0.47	1.63	0.86

**Table 2.** Training time on different hardware platforms and different batchsizes.



**Figure 7.** Training time on single 2080Ti and dual 2080Ti GPUs for our work and HugeCTR.

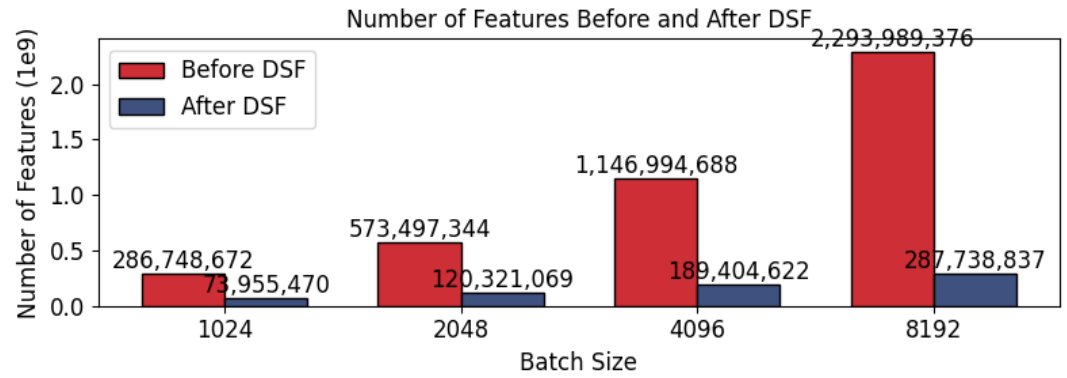
between 1.12 and 2 seconds for the full range of batch sizes. HECA further reduces these times to between 0.36 and 1.68 seconds, highlighting the efficiency of our method on high-end AI hardware. Finally, the 4090 provides the best performance overall, with the original training time ranging from 0.86 to 1.63 seconds and the optimized time ranging from 0.21 to 0.86 seconds. This demonstrates the substantial performance gains that can be achieved on modern consumer-grade hardware when optimizations are applied. HECA achieves a maximum speedup of 5.9x and an average speedup of 2.8x across these platforms. As



**Figure 8.** Cache hit rate under four different cache sizes and four different batch sizes.

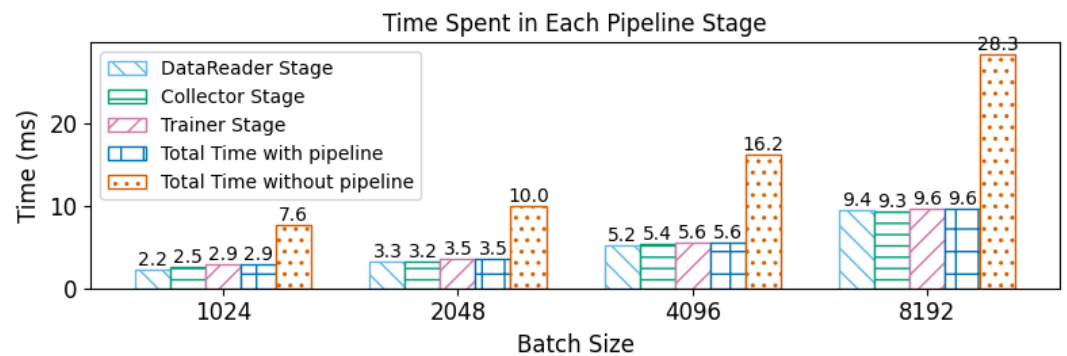
shown in Figure 8, we test the cache hit rate of HECA across four batch sizes: 1024, 2048, 4096, and 8192, and various cache sizes. With a cache size of 2000, the hit rates are 72%, 77%, 92%, and 95%, respectively. The hit rates for the first two batch sizes are relatively low. Increasing the cache size to 5000 improved the hit rates, and the hit rate stabilize when the cache size exceed 5000. At a cache size of 8000, the increasement in hit rate is negligible.

The purpose of DSF operation is to minimize data transfer volume between CPU and GPU, thereby reducing latency. As shown in Figure 10, With a batch size of 1024, the DSF operation retains only 26% of the original features before DSF operation. As the batch size increases to 2048, 4096, and 8192, the proportion decreases to 21%, 17%, and 13%, respectively.



**Figure 9.** Comparison of feature quantity before and after DSF operation.

HECA uses pipeline parallelism, dividing the overall process into three stages: dataLoader, collector, and trainer. The Figure 10 reveals that, with our design, the training time of the trainer module is slightly longer than the other two modules, but in the parallel



**Figure 10.** The fifth bar indicates that there is no pipeline, and the total training time is the sum of the time taken in the three stages.

design, it effectively masks the execution time of the other two modules. This ensures that the next iteration can proceed without delay.

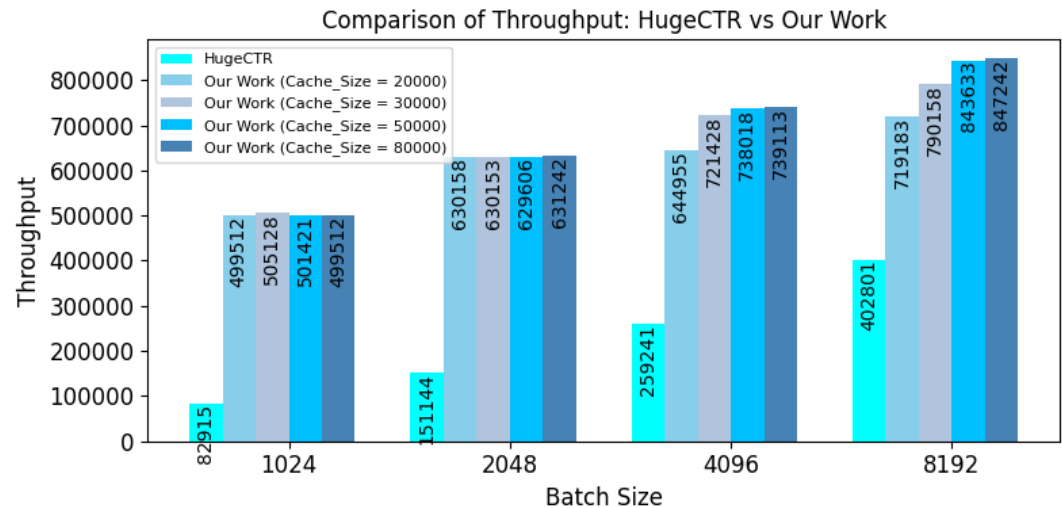
In Figure 11, HECA consistently outperforms HugeCTR in terms of throughput under various batch and cache sizes. For batch sizes of 1024 and 2048, cache size has little effect on throughput. However, for batch sizes of 4096 and 8192, the throughput is significantly lower when the cache size is 20000 or 30000 compared to 50000 or 80000. This is due to the lower cache hit rate in scenarios with smaller cache sizes and larger batch sizes, as illustrated in Figure 9, which negatively impacts the overall system throughput.

## 7. Future work

In future work, we will expand the scale of dataset, and try to introduce SSD transfer method. At the same time, we will also study the compression of embedding tables and explore methods with Mixed embedding precision to further optimize system performance. In addition, we plan to conduct experiments on other recommendation datasets and models to verify the versatility and scalability of our solution.

## 8. Conclusion

This paper presents an optimized solution, HECA, for efficiently training recommendation models with hundreds of GB to TB of embedding parameters using a single GPU. HECA offers a cost-effective method grounded in several key insights. First, features and embeddings exhibit a skewed access pattern. Second, CPU-GPU communication latency is high. These observations not only highlight challenges but also reveal opportunities for optimization.



**Figure 11.** Throughput under four different cache sizes and four different batch sizes.

To address these challenges, HECA is designed to reduce data transfer volume between CPU and GPU, enabling the training of recommendation models with hundreds of GB or even TB of parameters. Additionally, a three-stage pipeline method is introduced to effectively hide latency. During the training of the current batch, a prefetching strategy is employed to parallelly fetch the embedding parameters needed for the next batch, thereby covering data transfer time. Furthermore, a DSF operation is implemented to eliminate duplicate features within a batch, reducing the number of accesses to the CPU global embedding table and minimizing data transfer.

Evaluation results demonstrate that HECA not only maintains model accuracy but also enhances training speed, achieving a speedup of up to 2.8x on average and improving system throughput.

**Funding:** This work was supported by China University of Petroleum-Beijing research funding program (Grant No. HX20200792).

## References

1. Zhang, S.A.; Yao, L.N.; Sun, A.X.; Tay, Y. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 2019, 52, 38, doi:10.1145/3285029.
2. Gomez-Urbe, C.A.; Hunt, N.J.A.T.o.M.I.S. The netflix recommender system: Algorithms, business value, and innovation. 2015, 6, 1-19.
3. Desai, A.; Chou, L.; Shrivastava, A.J.a.p.a. Random Offset Block Embedding Array (ROBE) for CriteoTB Benchmark MLPerf DLRM Model: 1000× Compression and 3.1× Faster Inference. 2021.
4. Zhao, S.; Li, F.; Chen, X.; Guan, X.; Jiang, J.; Huang, D.; Qing, Y.; Wang, S.; Wang, P.; Zhang, G.J.I.T.o.P.; et al. v pipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel dnn training. 2021, 33, 489-506.
5. Sergeev, A.; Del Balso, M.J.a.p.a. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR abs/1802.05799* (2018). 2018.
6. Naumov, M.; Mudigere, D.; Shi, H.-J.M.; Huang, J.; Sundaraman, N.; Park, J.; Wang, X.; Gupta, U.; Wu, C.-J.; Azzolini, A.G.J.a.p.a. Deep learning recommendation model for personalization and recommendation systems. 2019.
7. Gai, K.; Zhu, X.; Li, H.; Liu, K.; Wang, Z.J.a.p.a. Learning piece-wise linear models from large scale data for ad click prediction. 2017.
8. Lyu, F.Y.; Tang, X.; Zhu, H.; Guo, H.F.; Zhang, Y.X.; Tang, R.M.; Liu, X.; Acm. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM)*, Atlanta, GA, Oct 17-21, 2022; pp. 1399-1409.
9. Theocharous, G.; Thomas, P.S.; Ghavamzadeh, M.; Acm. Ad Recommendation Systems for Life-Time Value Optimization. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, Florence, ITALY, May 18-22, 2015; pp. 1305-1310.
10. Guo, H.F.; Guo, W.; Gao, Y.; Tang, R.M.; He, X.Q.; Liu, W.Z.; Assoc Comp, M. ScaleFreeCTR: MixCache-based Distributed Training System for CTR Models with Huge Embedding Table. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Electr Network*, Jul 11-15, 2021; pp. 1269-1278.
11. Ge, T.Z.; Zhao, L.Q.; Zhou, G.R.; Chen, K.Y.; Liu, S.Y.; Yi, H.M.; Hu, Z.L.; Liu, B.C.; Sun, P.; Liu, H.Y.; et al. Image Matters: Visually Modeling User Behaviors Using Advanced Model Server. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*, Torino, ITALY, Oct 22-26, 2018; pp. 2087-2095.
12. Wilkening, M.; Gupta, U.; Hsia, S.; Trippel, C.; Wu, C.J.; Brooks, D.; Wei, G.Y.; Assoc Comp, M. RecSSD: Near Data Processing for Solid State Drive Based Recommendation Inference. In *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Electr Network, Apr 12-23, 2021; pp. 717-729.
13. Jiang, Y.M.; Zhu, Y.B.; Lan, C.; Yi, B.R.; Cui, Y.; Guo, C.X.; Assoc, U. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Electr Network, Nov 04-06, 2020; pp. 463-479.
14. Ren, J.K.; Yu, G.D.; Ding, G.Y. Accelerating DNN Training in Wireless Federated Edge Learning Systems. *IEEE J. Sel. Areas Commun.* 2021, 39, 219-232, doi:10.1109/jsac.2020.3036971
15. Peng, Y.H.; Zhu, Y.B.; Chen, Y.R.; Bao, Y.X.; Yi, B.R.; Lan, C.; Wu, C.; Guo, C.X.; Acm. A Generic Communication Scheduler for Distributed DNN Training Acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, Huntsville, CANADA, Oct 27-30, 2019; pp. 16-29.
16. Ye, H.C.; Zhang, X.F.; Huang, Z.Z.; Chen, G.S.; Chen, D.M.; Ieee. HybridDNN: A Framework for High-Performance Hybrid DNN Accelerator Design and Implementation. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, Electr Network, Jul 20-24, 2020.
17. Garofalo, A.; Perotti, M.; Valente, L.; Tortorella, Y.; Nadalini, A.; Benini, L.; Rossi, D.; Conti, F.; Ieee. DARKSIDE: 2.6GFLOPS, 8.7mW Heterogeneous RISC-V Cluster for Extreme-Edge On-Chip DNN Inference and Training. In *Proceedings of the 48th IEEE European Solid State Circuits Conference (ESSCIRC)*, Milan, ITALY, Sep 19-22, 2022; pp. 273-276.
18. Shen, H.C.; Chen, L.Q.; Jin, Y.C.; Zhao, L.Y.; Kong, B.Y.; Philipose, M.; Krishnamurthy, A.; Sundaram, R.; Acm. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, Huntsville, CANADA, Oct 27-30, 2019; pp. 322-337.
19. Fowers, J.; Ovtcharov, K.; Papamichael, M.; Massengill, T.; Liu, M.; Lo, D.; Alkalay, S.; Haselman, M.; Adams, L.; Ghandi, M.; et al. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, Los Angeles, CA, Jun 01-06, 2018; pp. 1-14.
20. Capra, M.; Bussolino, B.; Marchisio, A.; Masera, G.; Martina, M.; Shafique, M. Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead. *IEEE Access* 2020, 8, 225134-225180, doi:10.1109/access.2020.3039858.
21. Chen, Y.; He, J.; Zhang, X.F.; Hao, C.; Chen, D.M.; Acm. Cloud-DNN: An Open Framework for Mapping DNN Models to Cloud FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, Seaside, CA, Feb 24-26, 2019; pp. 73-82.
22. Yao, W.; Hou, Q.; Wang, J.; Lin, H.; Li, X.; Wang, X. A personalized recommendation system based on user portrait. In *Proceedings of the Proceedings of the 2019 international conference on artificial intelligence and computer science*, 2019; pp. 341-347.
23. Rajarajeswari, S.; Naik, S.; Srikant, S.; Sai Prakash, M.; Uday, P. Movie recommendation system. In *Proceedings of the Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018*, Volume 1, 2019; pp. 329-340.

24. Crespo, R.G.; Martínez, O.S.; Lovelle, J.M.C.; García-Bustelo, B.C.P.; Gayo, J.E.L.; De Pablos, P.O.J.C.i.h.b. Recommendation system based on user interaction data applied to intelligent electronic books. 2011, 27, 1445-1449. 446
25. Yuan, Z.; Yuan, F.; Song, Y.; Li, Y.; Fu, J.; Yang, F.; Pan, Y.; Ni, Y. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. In Proceedings of the Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2023; pp. 2639-2649. 447
26. Lee, Y. Recommendation system using collaborative filtering. 2015. 448
27. Goyani, M.; Chaurasiya, N.J.E.e.l.o.c.v.; analysis, i. A review of movie recommendation system: Limitations, Survey and Challenges. 2020, 19, 0018-0037. 449
28. Lian, X.R.; Yuan, B.H.; Zhu, X.F.; Wang, Y.L.; He, Y.J.; Wu, H.H.; Sun, L.; Lyu, H.D.; Liu, C.J.; Dong, X.; et al. PERSIA: An Open, Hybrid System Scaling Deep Learning-based Recommenders up to 100 Trillion Parameters. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Washington, DC, Aug 14-18, 2022; pp. 3288-3298. 450
29. Miao, X.P.; Zhang, H.L.; Shi, Y.N.; Nie, X.N.; Yang, Z.; Tao, Y.Y.; Cui, B. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *Proc. VLDB Endow.* 2021, 15, 312-320, doi:10.14778/3489496.3489511. 451
30. Jiang, B.Y.; Deng, C.; Yi, H.M.; Hu, Z.L.; Zhou, G.R.; Zheng, Y.; Huang, S.; Guo, X.Y.; Wang, D.Y.; 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 with KDD (DLP KDD), Anchorage, AK, Aug 05, 2019. 452
31. Rong, H.D.; Wang, Y.Z.H.; Zhou, F.H.; Zhai, J.J.; Wu, H.Y.; Lan, R.; Li, F.; Zhang, H.; Yang, Y.K.; Guo, Z.Y.; et al. Distributed Equivalent Substitution Training for Large-Scale Recommender Systems. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), Electr Network, Jul 25-30, 2020; pp. 911-920. 453
32. Zhao, W.; Xie, D.; Jia, R.; Qian, Y.; Ding, R.; Sun, M.; Li, P.J.P.o.M.L.; Systems. Distributed hierarchical gpu parameter server for massive scale deep learning ads systems. 2020, 2, 412-428. 454
33. Yang, J.A.; Huang, J.; Park, J.; Tang, P.T.P.; Tulloch, A.J.a.p.a. Mixed-precision embedding using a cache. 2020. 455
34. Balasubramanian, K.; Alshabanah, A.; Choe, J.; Annaram, M.; Acm. cDLRM: Look Ahead Caching for Scalable Training of Recommendation Models. In Proceedings of the 15th ACM Conference on Recommender Systems (RECSYS), Amsterdam, NETHERLANDS, Sep 27-Oct 01, 2021; pp. 263-272. 456
35. Kwon, Y.; Rhu, M. Training personalized recommendation systems from (GPU) scratch: Look forward not backwards. In Proceedings of the Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022; pp. 860-873. 457
36. Shan, Y.; Hoens, T.R.; Jiao, J.; Wang, H.J.; Yu, D.; Mao, J.C.; Assoc Comp, M. Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Francisco, CA, Aug 13-17, 2016; pp. 255-262. 458
37. Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M. Wide deep learning for recommender systems. In Proceedings of the Proceedings of the 1st workshop on deep learning for recommender systems, 2016; pp. 7-10. 459
38. Guo, H.; Tang, R.; Ye, Y.; Li, Z.; He, X.J.a.p.a. DeepFM: a factorization-machine based neural network for CTR prediction. 2017. 460
39. Wang, R.X.; Fu, B.; Fu, G.; Wang, M.L.; Acm. Deep Cross Network for Ad Click Predictions. In Proceedings of the 23rd ACM-SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Halifax, CANADA, Aug 13-17, 2017. 461
40. Xiao, J.; Ye, H.; He, X.N.; Zhang, H.W.; Wu, F.; Chua, T.S. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), Melbourne, AUSTRALIA, Aug 19-25, 2017; pp. 3119-3125. 462
41. Owens, J.D.; Houston, M.; Luebke, D.; Green, S.; Stone, J.E.; Phillips, J.C.J.P.o.t.I. GPU computing. 2008, 96, 879-899. 463
42. Mittal, S.; Vetter, J.S. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.* 2015, 47, 35, doi:10.1145/2788396. 464
43. Clauset, A.; Shalizi, C.R.; Newman, M.E.J.s.r. Power-law distributions in empirical data. 2009, 51, 661-703. 465
44. Tomkins, A.; Patterson, R.H.; Gibson, G.J.A.S.P.E.R. Informed multi-process prefetching and caching. 1997, 25, 100-114. 466

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content. 467