ImPACT: Importance-Informed Prefetching and Caching for I/O-Bound DNN Training

Weijian Chen, Shuibing He, *Member, IEEE*, Ruidong Zhang, Xuechen Zhang, *Member, IEEE*, Ping Chen, Siling Yang, *Graduate Student Member, IEEE*, Haoyang Qu, and Xuan Zhan

***Abstract*—Fetching large amounts of DNN training data from storage systems causes high I/O latency and GPU stalls. Importance sampling can reduce data processing on GPUs while maintaining model accuracy, but current frameworks lack a prefetching and caching layer to optimize data fetches and cache management based on sample importance. This leads to unnecessary fetches, poor cache hit ratios, and random I/Os. We present ImPACT, an importance-informed prefetching and caching system, to accelerate I/O-bound DNN training. First, we propose an importance-informed prefetching technique to reduce the prefetching of unimportant data. Then, we introduce an importance-aware caching layer, partitioned into two regions: H-cache and L-cache, which store samples of high importance and low importance respectively. Rather than using recency or frequency, we manage data items in H-cache according to their corresponding sample importance. When there is a cache miss in L-cache, we use sample substitutability and dynamic packaging to improve the cache hit ratio and reduce the number of random I/Os. Our experimental results show that ImPACT has a negligible impact on training accuracy while speeding up DNN training by up to 3.5× compared to state-of-the-art prefetching and caching systems.**

***Index Terms*—Prefetching, caching, importance-sampling, DNN training.**

I. Introduction

DEEP neural networks (DNNs) have been attracting attention in computer vision [1], natural language processing [2] and many other fields [3], [4], [5]. DNN training needs to fetch data from I/O systems and compute them for updating parameters [6], [7], [8]. Recent research shows that I/O has become the bottleneck in DNN training [9], [10]. This is because AI accelerators, such as GPUs and ASICs, have evolved at a faster pace than storage devices. Another reason is that DNN model training needs to access ever-increasing datasets. For example, the Google OpenImages dataset used in the Open Images Challenge is about 18 TB [11]. And training data items (referred to as samples) are shuffled every epoch to ensure that they are read in a random order, leading to poor I/O efficiency of storage systems.

Importance sampling (IS) [12], [13], [14] is an approach to accelerate DNN training by skipping the calculation of some items while maintaining a similar accuracy. It assigns each data item an importance value to reflect its influence on DNN model accuracy. We refer to samples of high and low importance values as H-samples and L-samples respectively. When importance sampling is used, H-samples are computed in a higher probability while L-samples are computed in a lower probability. However, all existing sampling approaches are *computing-oriented IS (CIS)* algorithms because they only focus on reducing computing on GPUs instead of I/O. They still need to fetch all items to memory and thus perform poorly for I/O-bound DNN Training. For example, when training ResNet18 with CIFAR10 on a parallel file system, a history-based CIS algorithm [14] can only accelerate the overall training by 1.02× though the computing time is reduced by 1.3× (Section II-B).

Prefetching and caching are widely used methods to alleviate I/O bottlenecks [9], [10], [15]. Specifically, prefetching conceals I/O latency by proactively reading samples that will be used in the future, integrating I/O time into computation time. Caching reuses data in fast memory, reducing slow I/O accesses. However, current deep learning caching systems process the entire dataset for each epoch, missing opportunities to optimize I/O by serving fewer samples.

Moreover, they do not consider sample importance. For example, CoorDL [10] never replaces samples in its MinIO cache. Therefore, it is possible that the MinIO cache does not have space for H-samples after it is full. Quiver [9] exploits substitutability to avoid memory thrashing. However, it is likely H-samples are substituted by L-samples leading to poor accuracy of DNN models after training.

None of the existing approaches can reduce the amount of data fetched and consider sample importance in the I/O of DNN training. In this paper, we propose the idea of *I/O-oriented importance sampling (IIS)* and apply it to data prefetching and caching. IIS only fetches a subset of samples instead of all the

Received 29 October 2024; revised 21 March 2025; accepted 3 May 2025. Date of publication 9 May 2025; date of current version 11 July 2025. This work was supported in part by the National Science Foundation of China under Grant 62172361, in part by the Major Projects of Zhejiang Province under Grant LD24F020012, in part by the Open Project Program of Wuhan National Laboratory for Optoelectronics under Grant 2023WNLOKF005, and in part by the Pioneer and Leading Goose R&D Program of Zhejiang Province under Grant 2024SSYS0002. Recommended for acceptance by T. Adegbija. *(Corresponding author: Shuibing He.)*

Weijian Chen, Shuibing He, Ruidong Zhang, Ping Chen, Siling Yang, Haoyang Qu, and Xuan Zhan are with the State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou 310027, China, also with Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou 310027, China, and also with Zhejiang Key Laboratory of Big Data Intelligent Computing, Hangzhou 310027, China (e-mail: weijianchen@zju.edu.cn; heshuibing@zju.edu.cn; 22321348@zju.edu.cn; zjuchenping@zju.edu.cn; slingzjunet@zju.edu.cn; haoyangqu@zju.edu.cn; zhanxuan@zju.edu.cn).

Xuechen Zhang is with the School of Engineering and Computer Science, Washington State University Vancouver, Vancouver WA 98686, USA (e-mail: xuechen.zhang@wsu.edu).

Digital Object Identifier 10.1109/TC.2025.3569126

original samples from the cache or storage. Simply caching H-samples for DNN training does not work well in the context of importance sampling. The existing cache replacement algorithms are designed to explore temporal locality based on recency or frequency. However, importance sampling accesses samples randomly and based on their impact on the model accuracy. Therefore, we need a new importance-sampling-informed cache replacement algorithm to achieve a higher hit ratio in the cache.

There are three challenges in the design of the new cache system. First, a dilemma arises when trying to reduce the prefetching of unimportant samples, as the importance values of the samples are unknown and can only be accurately calculated after they have been loaded into the CPU for computation. Second, the importance values of samples may change across epochs, so statically caching the initially most important samples is inefficient. We need an efficient algorithm to keep a maximum number of H-samples in the cache when the importance values of samples are changed. Third, caches have limited space. If we only cache H-samples, data loaders still need to access L-samples randomly with poor I/O efficiency.

To address these issues, we design and implement ImPACT, new importance-informed prefetching and caching systems to accelerate DNN training when I/O is its performance bottleneck.

To reduce the prefetching of unimportant samples while avoiding large I/O overhead, we propose a hybrid importance evaluation method based on the observation that the importance rankings of most training samples are stable, while a small portion of the samples exhibits significant fluctuations. Then, ImPACT generates a sequence of sample IDs that includes only a subset of the entire dataset for the current epoch prefetching by reducing the probability of selecting relatively less important samples.

The cache space is partitioned into two regions: H-cache and L-cache which store H-samples and L-samples respectively. We use a small-top heap (H-heap) for cache management. When H-cache is full, the data item corresponding to the node at the top of the heap will be evicted if its importance value is smaller than that of the incoming one. To efficiently refill the cache when importance values are changed, we manage a shadow heap for H-heap. After the importance values are updated, the H-heap becomes read-only and is used only for item eviction from the cache. The changes (i.e., insertions/evictions and value updates) to the H-heap are recorded in the shadow heap. To reduce the amount of random I/Os for L-samples, ImPACT uses dynamic packaging to load L-samples to L-cache in batch. When L-samples to be accessed are not in the L-cache, we apply substitutability to replace the missing L-samples with those already in the cache, thus reducing the number of small random I/Os and keeping a high training accuracy.

In summary, this paper offers the following contributions:

- We are the first to propose the idea of I/O-oriented importance sampling (IIS) and apply it into both the prefetching and caching system ImPACT.
- To make importance-informed prefetching truly effective, we propose a hybrid importance evaluation method, which accurately assesses sample importance while introducing minimal I/O overhead.
- We design a cache replacement algorithm based on sample importance, a dynamic packaging technique to further boost the I/O system performance.
- We implement ImPACT in PyTorch [16] and evaluate it with eight DNN models on two datasets. The evaluation shows that ImPACT outperforms the state-of-the-art DNN prefetching and caching systems Quiver [9], CoorDL [10], and iCACHE [17] by up to 3.9×, 3.5× and 2.5× on the model training time and 5.6×, 4.8×, and 2.6 on the I/O time while achieving an equivalent training accuracy.

II. Background and Motivation

A. I/O-Bound DNN Model Training

DNN training is iterative, with model accuracy converging gradually. One training *epoch* in a conventional training scheme involves reading all training samples once, and each epoch consists of multiple *iterations* that process mini *batches* of training data. Specifically, DNN training involves three steps: (1) loading a mini batch from remote storage to host memory, (2) pre-processing data using CPUs, and (3) training the DNN with GPUs.

As dataset sizes grow [9], training data must be accessed from remote high-capacity storage or parallel file systems, a common setup in modern HPC-AI data centers [18], [19], [20]. By default, data loaders must randomly feed all samples exactly once per epoch to maintain model accuracy, leading to frequent random reads over slow network transfers and causing I/O-bound data stalls during DNN training.

Although data prefetching, data caching, batch size adjustment, and multi-GPU training are widely used to accelerate DNN training, they are inefficient for I/O-bound tasks for the following reasons. (1) Prefetching is effective only when computing time is longer than I/O time. With powerful GPUs like A100 and H100, the computing time can be less than I/O time [10]. (2) Caching policies are usually based on traditional temporal/spatial locality, thus they are insufficient for DNN workloads with strong randomness [9]. (3) Batch size adjustment and multi-GPU training are mainly used to boost computing performance instead of I/O time.

To verify this, we train four DNN models on a server with four A100 GPUs with various batch sizes. The dataset is CIFAR10 and placed in a remote OrangeFS file system. The system configuration is described in section V-A. We enable the built-in prefetching technique in PyTorch and implement an LRU-based cache system. The cache size is 20% of the training dataset, as Quiver [9] does. We measure the average I/O time, which is the time the training process waits to acquire a batch of training data. Specifically, this is the time on the critical path to fetch each batch of data from remote OrangeFS to local CPU memory, after excluding the time that can be hidden by asynchronous computation. We decoupled the data preprocessing operations (e.g., transformation) from the Dataloader and add timestamps in the Dataloader of the PyTorch program to obtain the I/O time. Fig. 1 shows that even with existing performance optimizing techniques, I/O is still a bottleneck for multi-GPU training cases. For example, the I/O bottleneck becomes more prominent because the ratio of the I/O time to the total training

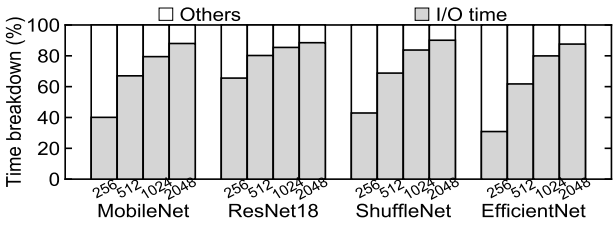
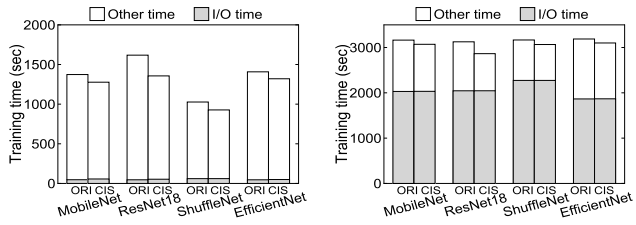


Fig. 1. Training time with varying batch sizes on four A100 GPUs.



(a) Computing-bound training. (b) I/O-bound training.

Fig. 2. Training time with and without CIS algorithm in computing. "ORI" means original training system without CIS.

time increases from 44% to 89% on average while the batch size increases from 256 to 2048.

B. Computing-Oriented IS Approaches are Inefficient for I/O Bound Training

Importance sampling accelerates DNN model training by feeding fewer samples on GPUs for updating parameters. When it is applied in model training, the random order of sample computed may be changed to reflect the principle that H-samples should be computed more frequently than L-samples. A lot of works have used importance sampling to effectively accelerate deep learning applications [14], [21], [22], [23], [24], [25].

However, all existing importance sampling algorithms for reducing training samples are designed for computing-bound tasks (which we refer to as CIS), which reduce computation time but still load *all* samples, rendering them inefficient for I/O-bound tasks. We validate this with four DNN models on CIFAR10 using a CIS algorithm [14]. We use a single A100 GPU for training with a batch size of 256 and enable PyTorch's prefetching. Fig. 2(a) shows that with local DRAM storage (i.e., DRAM-based tmpfs without a cache), CIS reduced total training time by 1.2×. However, with a LRU cache holding 20% of data from a remote OrangeFS, CIS only reduced total training time by 1.02×, as I/O became the bottleneck in Fig. 2(b). This bottleneck is due to factors like large datasets, low remote storage IOPS, and significant CPU resources for data loading.

Inspired by the idea of CIS, it is feasible to apply I/O-oriented importance sampling (IIS) to fetch fewer samples from the cache or storage to accelerate I/O bound DNN model training with acceptable accuracy degradation.

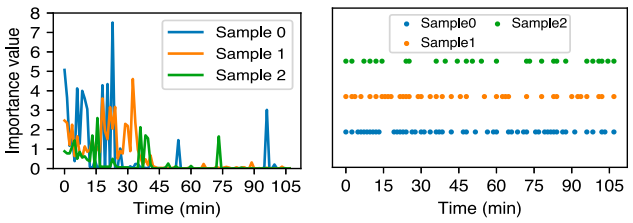
C. Importance-Informed Prefetching, Caching and Challenges

None of the existing prefetching and caching systems are designed for DNN model training based on importance sampling. (1) While prefetching systems like PyTorch [16], TensorFlow [26], and NoFS [27] parallelize the time spent prefetching next batch data with the current batch's computation time, they always prefetch the entire dataset for each epoch of training. This results in a significant prefetching time overhead, making it difficult to completely hide this overhead. (2) OS page cache explores temporal locality and uses recency or frequency for managing the samples when it is full [28]. When samples are randomly accessed and the time of revisiting the same samples in the page cache is long, the page cache will have a high miss ratio. CoorDL [10] keeps all data samples in the cache with no eviction to avoid thrashing. The cache is used to store both H-samples and L-samples. When it is full, H-samples will not be stored in the cache, leading to a higher miss ratio of H-samples, which are accessed more frequently than L-samples. When Quiver [9] is used, an H-sample which is not in the cache may be substituted by an L-sample, reducing the model accuracy.

There are three challenges in the design of the importance-informed prefetching and caching systems.

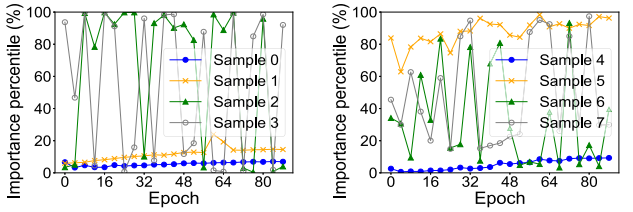
Importance values are unknown before prefetching. Most existing methods [14], [21], [22] for determining the importance of training data need to read training data first and then use a lightweight method to calculate the current importance. To simplify, we refer to these methods as reassessment-based methods. However, although these methods can accurately measure data importance, they are not suitable for I/O bottleneck scenarios. This is because they introduce significant I/O overheads. Some methods, such as iCache [17], SHADE [24], use historical importance values to predict sample importance for the next training epoch. To simplify, we refer to these methods as history-based methods. Although they avoid I/O overheads by not needing to read and recalculate importance values, they may lead to inaccurate assessments of data importance due to the gap between historical importance values and the current actual importance values. This could potentially require more training epochs and time to achieve the target model accuracy as shown in section V-E.

Varying importance values. The importance value of one sample changes across epochs during training [14], [21]. To verify this, we record the importance values of three samples (i.e., Sample 0 to Sample 2) with a model loss-based importance sampling algorithm [14] when training ResNet18 on CIFAR10. As Fig. 3 shows, the same sample is selected from time to time with varying importance values. The importance value of the same sample changes because it is determined by the sample content and the model's parameters (e.g., weights) which are updated by the SGD algorithm iteratively [29]. Therefore, the H-samples in the previous epoch may become L-samples. Since samples are selected according to their relative importance values, it is not practical to set an importance threshold to decide whether to place a sample in the cache or not. We need a judicious cache management algorithm to keep a maximum number of H-samples in the cache without significantly affecting model accuracy when the importance values of samples are changed.



(a) Varying importance values. (b) Sample selection in training.

Fig. 3. Varying importance values and the selected samples by the importance sampling algorithm during training. One dot in (b) means the sample is selected once.



(a) ResNet18 on CIFAR10. (b) ResNet50 on Tiny-ImageNet.

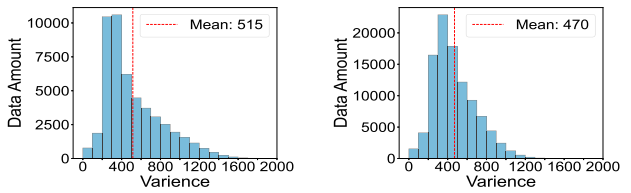
Fig. 4. Importance rankings of eight training samples.

Random I/Os after cache misses. A cache has limited capacity. It cannot always store all H-samples. When cache misses happen, it is required to read H-samples randomly. Furthermore, although H-samples are accessed more frequently than L-samples, training frameworks still need to access L-samples to improve sample diversity for high training accuracy. The performance of loading L-samples from storage systems may become the I/O bottleneck causing data stalls. A widely used method of mitigating this problem is packaging the training dataset into many large files, each of which contains a certain number of L-samples [26]. However, it is not practical in this case because importance sampling specifies the order of samples to be trained. They are probably distributed in different data packages and will cause a serious read amplification problem if we directly use the existing packaging algorithms.

D. Observation

Although the absolute importance values of training samples fluctuate irregularly, we have found that *the importance rankings of most samples are stable, while only a few samples exhibit significant fluctuations.* Specifically, a large number of samples are consistently more or less important compared to others throughout the entire training process, while a smaller subset of training samples is sometimes more important than the majority of others, and other times the opposite is true.

To verify the observation, we train two popular DNN models, ResNet18 and ResNet50, on the CIFAR10 [30] and Tiny-ImageNet [31] datasets for 90 epochs. We evaluate the importance of each data item and sort them in ascending order at the end of each epoch. The importance percentiles are used to measure the importance rankings of each training sample, with a higher percentile indicating greater importance. We randomly



(a) ResNet18 on CIFAR10. (b) ResNet50 on Tiny-ImageNet.

Fig. 5. Density distribution of the importance ranking variance.

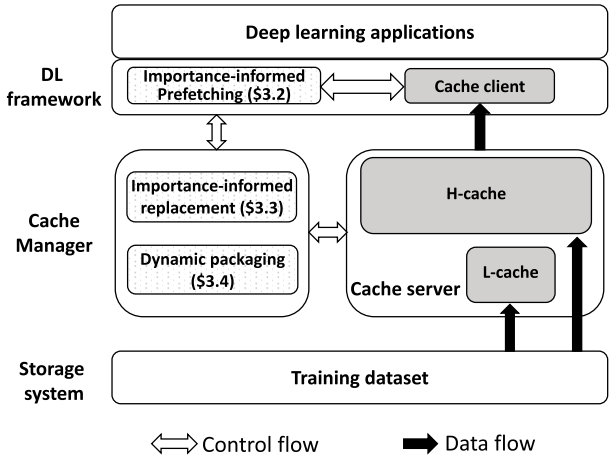


Fig. 6. Single-node architecture overview of ImPACT.

selected eight samples, and the results are shown in Fig. 4. We observe that the importance ranking of samples 0, 1, 4, and 5 remain consistently stable, while the others exhibit large fluctuations. To further quantitatively measure the degree of variation in importance rankings, we calculate the ranking variance of each sample throughout the entire training process. We plot a data amount distribution curve, as shown in Fig. 5. We find that 62% and 59% of the samples have variances below the mean value on CIFAR10 and Tiny-ImageNet, respectively.

III. Design of ImPACT

In this section, we present the design of ImPACT. We first introduce the system overview of ImPACT and then elaborate on its three key components.

A. System Overview

ImPACT is an intelligent prefetching and caching system for accelerating DNN model training. It supports both single-node with multiple GPUs training and multi-node distributed training, both of which are popular deep learning training configurations [9], [10], [32], [33]. To illustrate the details, we first show the single-node architecture in Fig. 6. Overall, it consists of an importance-informed prefetching module and an importance-informed cache system. The cache component consists of client modules, cache managers, and servers.

Importance-informed prefetching. To prefetch important samples, we devise a two-phase prefetching technique. In the first phase, the prefetcher evaluates the importance values of all

training data using a *hybrid importance evaluation* method at the beginning of each epoch. The core idea is to re-evaluate the actual importance of only a small number of samples that exhibit the greatest fluctuations in importance rankings, while directly using historical importance for the remaining samples; it can be seen as a combination of two existing methods (i.e., the historical-based methods [17], [24] and the reassessment-based methods [14], [21], [22]) to improve evaluation precision while reducing I/O overhead (section III-B). In the second phase, the prefetcher invokes an importance sampling algorithm to select samples that will be trained in the current epoch and initiates prefetch requests for these selected training data. Although many algorithms exist, we choose a recent loss-based importance sampling algorithm [22] in our current design for its simplicity and efficiency. We plan to study other algorithms in future work. Consequently, samples that are not selected do not need to be loaded. This approach reduces both I/O and computation time.

Cache client and server. The client modules are integrated into the deep learning frameworks (e.g., PyTorch and TensorFlow). It mainly plays the role of request forwarding. When data loaders of DNN applications start to randomly select samples to read, the clients will forward the request to ImPACT servers by calling the RPC interface. One client belongs to a unique DNN training job and is transparent to the users.

A client module maintains an H-list to record H-samples for the training job. H-list is generated by the importance sampling algorithm. It is a list of vectors $< ID, IV >$, where ID corresponds to a sample's identity and IV is its importance value. Both the ID and IV are 64 bits (8B), thus the space overhead of H-list is trivial. We take a cache for ImageNet-1K (with 1281167 samples in 140GB) as an example. Assume the cache holds 20% samples, then the cached data size is 140GB*20%=28GB, and the importance mapping overhead is 1281167*20%*16B=3.9MB, which is just 0.014% (3.9MB/(28GB + 3.9MB)) of the whole cache space. The IVs are computed according to the importance sampling algorithm. Since the importance value of one sample changes across epochs during training, we periodically update importance values (section III-C).

The functionality of the ImPACT server is to provide a user-level cache, which stores training datasets in memory to accelerate the I/O-bound DNN training.

Cache manager. It is designed to manage the cache based on the importance values of samples. Since the importance values referenced by the sampling process are periodically updated by the hybrid importance evaluation, the cache manager periodically pulls the H-list from clients to achieve a decent trade-off between cache performance and its management overhead. Additionally, the manager needs to dynamically pack samples which are later loaded to L-Cache (section III-D). The minimum I/O unit is a package of samples.

H-cache. It stores H-samples recorded in H-list. Its capacity determines how many samples can be stored in H-cache. When its capacity is not large enough to cache all H-samples, an importance-informed cache replacement algorithm is applied to manage it. The general idea is that the samples of higher importance value have a lower chance of being evicted from the cache. And the importance value is provided by H-list. If the importance values of H-samples are changed leading to a lower hit ratio of H-cache, it needs to refill the cache with new H-samples. When importance sampling is used for training, the importance-informed cache replacement achieves a higher cache hit ratio than the commonly used LRU-based cache replacement algorithm and its variants.

The size of H-cache is determined based on the following equation: $Size_{hcache} = Size_{cache} * (Frequence_{HI}/Frequence_{LI} + Frequence_{HI})$. $Frequence_{HI}$ and $Frequence_{LI}$ are the frequency of accesses to H-samples and L-samples respectively. $Size_{cache}$ is the cache size. A higher $Frequence_{HI}/Frequence_{LI}$ automatically increases the cache space allocated for H-samples and reduces the space for L-samples. The ImPACT manager tracks the number of accesses to H-samples and L-samples. The minimum size of L-cache is equal to the number of samples in one package.

L-cache. The purpose of L-cache is to further reduce the number of small random I/Os for accessing L-samples. It is designed to cache only the L-samples, which are not in the H-list. Another benefit of L-cache is to maintain a high model accuracy. Basically, we manage L-cache with *substitutability*, a unique characteristic of the DNN I/O process [9]; it means when a read request is missed in the cache, it can be served by another randomly picked sample in the cache. While the missed samples from L-cache can be substituted with the one in H-cache, serving them with another one in L-cache can keep a high degree of sample diversity and yields better training accuracy (See section V-F). The L-samples are packaged in advance by an asynchronous thread. In addition, because the importance values of samples are constantly changing across epochs, it also needs to re-packing the L-samples accordingly. The size of L-cache is equal to $Size_{cache} - Size_{hcache}$.

B. Importance-Informed Prefetching

As stated in section III-A, the goal of this module is to reduce the number of unimportant samples prefetched during training.

Once a DNN training task begins, it first enters a warm-up phase to identify training samples that exhibit high variance in importance rankings. Specifically, all samples undergoes training in each epoch during the initial k epochs, and their importance values are sorted. Once k epochs have been completed, the variance of importance rankings for each training sample can be calculated and samples with significant fluctuations in importance rankings are identified. For clarity, we denote the set of IDs of samples with high importance ranking variances as S. During the warm-up phase, all prefetched data are managed by a simple and efficient static caching strategy proposed by CoorDL [10], which means no replacement occurs.

Before the start of each subsequent training epoch, ImPACT uses a hybrid importance evaluation method to assess the importance values of all training data. It combines existing reaccessment-based method and history-based method to get an accurate evaluation of importance values with minimal additional I/O overhead, thereby reducing the amount of training data and time required to achieve the

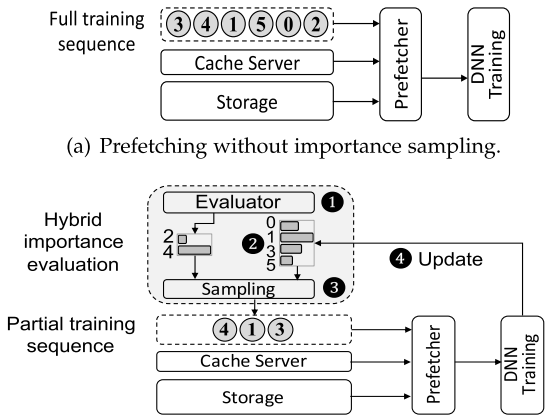


Fig. 7. Comparison of prefetching systems for one epoch training.

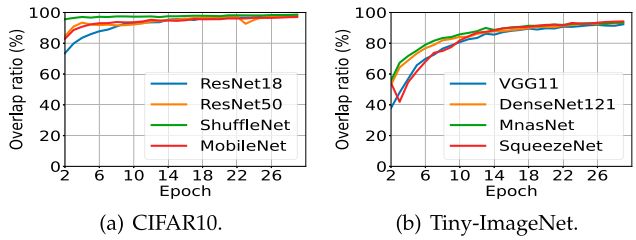


Fig. 8. Warmup phases on two training datasets with various models.

target model training accuracy. It unfolds in two steps. (1) Re-evaluation of set S. ImPACT re-evaluates the importance values of all data that belongs to the set S using an importance evaluator. This is a critical step because the importance values of data in S are known to be highly variable. To avoid repeatedly fetching training samples belonging to S from underlying storage, we use an additional static buffer to retain them in memory after their initial read. This ensures that these samples are preserved without undergoing cache replacement, thereby expediting the subsequent re-evaluation process. (2) Estimation for remaining data. ImPACT utilizes the most recent historical importance values of the remaining samples as their estimated importance for the current epoch. The rationale of this step is that since the importance rankings of these samples exhibit less variability compared to those in set S, re-evaluating them every epoch would be redundant and unnecessary. After the above two steps, an established importance sampling algorithm [22] is applied to select a subset of the training data for the current epoch's training. Besides, these data are placed into either the H-cache or the L-cache depending on whether they are H-samples or L-samples. The admission and eviction decisions are made by the method in section III-C for H-samples and section III-D for L-sample.

We use an example to illustrate the benefits of the importance-informed prefetching. Suppose we have six training samples, numbered 0-5. Assume the warm-up phase identifies the set $S = \{2,4\}$, indicating that samples 2 and 4 have the most unstable importance rankings. Fig. 7(a) shows the existing prefetching for one epoch training [16]. First, a random permutation of the six training samples is generated. Then, the prefetcher loads the data in batches from the cache or storage and sends them to the GPU for training. In contrast, with our prefetching method enabled, as shown in Fig. 7(b), samples $\{2,4\}$ are first loaded for importance re-evaluation (❶). The remaining four samples (i.e., $\{0,1,3,5\}$) use their historical importance values as estimates for the current epoch (❷). Then a loss-based sampling algorithm [22] is applied based on the estimated importance values of each sample, resulting in the selection of three samples $\{4,1,3\}$ (❸). The prefetcher will only prefetch three samples for the current epoch training, thereby reducing the amount of data that needs to be prefetched. During training, the loss values of samples not in S (i.e., $\{1,3\}$) are updated in CPU memory for the next epoch's training (❹).

To make importance-informed prefetching truly effective, we need to carefully adjust two hyperparameters: the length of the warm-up stage (k) and the size of the sample set (S), to balance benefits with additional time and space overhead. Adjusting these parameters is not trivial. A larger k allows for more accurate identification of samples with high variability in importance rankings, while a larger S enables more samples' importance to be accurately reevaluated, reducing the training of unimportant samples. However, increasing both k and S leads to greater additional time overhead due to the longer warm-up and reevaluation times. Similarly, smaller values of k and sizes of S will result in opposite benefits and drawbacks. Therefore, a trade-off needs to be made between the accuracy of importance evaluation and the additional time and space overhead.

We first determine the size of the set S, and then set the k value based on S. Considering that the samples in set S need to be statically cached, we empirically establish the size of S based on 0% to 100% of the cache space and observe the end-to-end time. Based on our observation, we empirically set the size of set S to be 25% of the total memory space to achieve a good tradeoff in our experiments (section V-I). Once the samples in S are stored in the buffer, they are excluded from the cache server to avoid redundant storage. With this S size and a static buffering strategy, the average re-evaluation time overhead is less than 6% of the total training time in our experiments.

To determine k, we generate a temporary set S' which contains samples IDs with the highest variability in importance rankings at the end of each epoch during the warm-up stage, matching the size of S. If two successive S' sets share over 80% of their sample IDs, we consider the last S' stable and end the warm-up phase. Additionally, users can set an upper limit on the number of epochs to avoid excessive duration. Fig. 8 shows the overlap ratios of adjacent S' sets during 30 epochs on CIFAR-10 and Tiny-ImageNet with eight popular DNN models. The X-axis begins at epoch 2, as overlap ratio is available only from this point. On average, the warm-up stage lasts 6.4 epochs, about 7% of the typical 90 total epochs for training [9]. The additional space needed to record importance values and variances during this stage is negligible; for large datasets like ImageNet, it requires only around 100MB, less than 0.4% of the cache space in our experiments (section V-A).

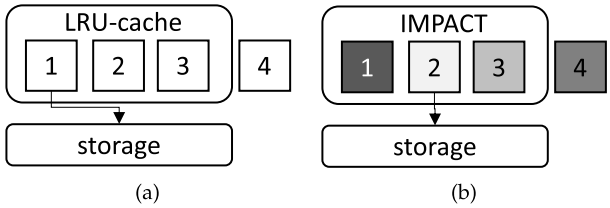


Fig. 9. Comparison of LRU and ImPACT. The squares in darker color denote samples with higher importance values.

C. Importance-Informed Cache Algorithm

We use a key-value store to manage data items in H-cache. The key denotes sample ID and the value stores the data item of a sample. ImPACT also manages a small-top-heap (H-heap) for cache management. The heap objects are also key-value pairs, whose key is the importance value of a data item and value is a reference to the item in the key-value store. The objects in H-heap are sorted based on their importance values. The object at the top of the heap is called *top-node*. The size of a heap object is 16 B. The space usage of H-heap is correlated to the number of H-samples and is generally less than 0.5% of H-cache capacity.

When the requested data items are H-samples but do not exist in H-cache, the server needs to read them from storage systems and return them to clients. It also needs to decide whether to cache the sample when H-cache is full. One challenge is that LRU-based cache replacement algorithms do not work effectively in training with importance sampling because they do not consider sample importance. Our observation shows that H-samples are accessed more frequently than L-samples in each epoch and across multiple epochs. Therefore, we need a new importance-informed cache replacement to improve the cache hit ratio in training.

Specifically, when H-cache is not full, the H-sample read from storage systems will be inserted into H-cache directly. ImPACT then creates a heap object corresponding to the H-sample and inserts it into the heap. When it is full, the importance value of an incoming H-sample is compared to that of top-node. The top-node will be evicted if its importance value is smaller than that of the incoming sample. Otherwise, the incoming sample will not be admitted. If the top-node is evicted, ImPACT will create a new heap object corresponding to the incoming H-sample and insert it into the heap.

Importance-informed cache replacement algorithm performs better than traditional LRU-like algorithms exploiting temporal and spatial locality. Let's take Fig. 9 as an example. We assume that the capacity of H-cache is three and three data items (#1, #2, #3) have been cached consecutively. When item #4 is accessed, the LRU-like replacement algorithm will evict item #1 because it is least recently used (shown in Fig. 9(a)). If we further assume item #2 has the least importance value thus is the top-node in the heap, we will have a higher probability of accessing #1 than accessing #2 in future references. Hence, a higher cache hit ratio will be achieved by evicting sample #2 (shown in Fig. 9(b)). We will experimentally demonstrate the effectiveness of importance-informed cache over LRU-based cache in section V-C.

The second challenge in the design of ImPACT is the importance values of data items change as the model is trained, leading to variation of a cache hit ratio. To solve this issue, ImPACT periodically updates H-list by pulling it from cache clients. Because of the overhead of building a small-top-heap, we do not update H-heap in place. Instead, ImPACT manages a shadow heap, which has the same structure as H-heap. After H-list is updated, the current H-heap becomes read-only and we continue using it for eviction purposes. But new heap objects are inserted into the shadow heap with their updated importance value. After the shadow heap is rebuilt completely against the updated H-list, it can be directly used as a new H-heap. Then the original H-heap is released. In this way, the update of importance values in the heap can be performed asynchronously and does not affect the critical I/O path of the training process.

D. Dynamic Packaging

Because ImPACT only stores H-samples in H-cache, L-samples that are not in H-list may still incur small random I/Os. We design a new approach, named dynamic packaging, to reduce data stall time caused by accessing L-samples. The idea is to maintain a small L-cache for storing L-samples. The L-samples are loaded in the memory in the unit of a package to improve I/O efficiency. The package size is at least 1 MB exploiting the spatial locality of storage systems. When an L-sample is requested, ImPACT returns the data item from L-cache if it is a hit. Otherwise, instead of reading the requested L-sample from the storage system, we apply substitutability and return a cached L-sample that has not been accessed at the current epoch. The IDs of L-samples that are missed in the cache will be recorded and later loaded from storage systems by the loading thread. Because we replace any missed L-samples with other L-samples in L-cache, we can achieve a hit ratio of 100% in L-cache. Consequently, L-heap is not needed for L-samples.

Although the previous work [9] has shown that replacing the cache missed requests with any samples in the cache will not affect the model accuracy, in our importance sampling scenario, it is different. We can only use the L-samples in the cache to replace the cache missed L-sample requests to avoid a significant drop in accuracy. This is because if we use the cached H-samples to replace the cache missed L-samples, it will lead to the H-samples being trained too many times while some L-samples are never trained. This disrupts the balance of the importance sampling algorithm when selecting samples, resulting in reduced training data diversity and model accuracy. Our results show that our method has a very minor impact on the model accuracy while significantly reducing data stall time (section V-B and section V-F).

Specifically, ImPACT uses two concurrent threads (i.e., packaging thread and loading thread) working together to achieve dynamic packaging, as shown in Fig. 10. At the beginning of epoch 1, there are no packages in the storage system. Once H-list is generated, the packaging thread randomly selects L-samples. It then packs them into large file packages and stores them in the storage system. Then the loading thread chooses one package and caches all data items in the package in L-cache.

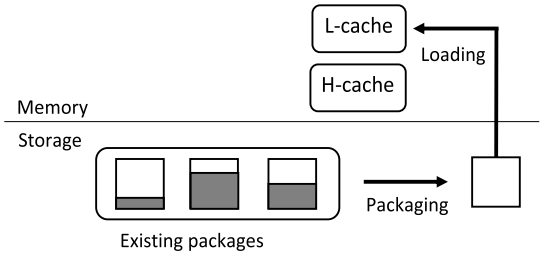


Fig. 10. Illustration of packing thread and loading thread. The white area in the package represents samples with low importance values and the dark area denotes samples with high importance values.

Next, when a requested sample is not in H-list and missed in L-cache, ImPACT directly randomly chooses a cached L-sample which has not been accessed to replace the requested one. When all data items in L-cache have been accessed once, new packages will be read into L-cache by the loading thread.

After H-list is updated, the ratio of L-samples in the existing packages is changed in the following training epochs. We need to periodically repack the samples to make sure that a package consists of a large number of L-samples to fill L-cache quickly in fewer I/Os. To achieve this goal, the samples that are previously missed in L-cache will be re-packed in the packages to increase sample diversity. Then the rest of space will be filled with L-samples that are randomly selected from the existing packages. At the same time, the loading thread will load the reorganized packages into memory. Then it stores L-samples in L-cache.

Both the loading thread and packaging thread run asynchronously. The package reorganization introduces three benefits. First, L-samples can be read in the form of large packages, which alleviates random small I/Os. Second, because of the dynamic repackaging, we can guarantee that a large number of L-samples are stored into L-cache for every I/O, thus improving the effective storage bandwidth. Third, compared to the existing fixed packaging strategy [26], package reorganization may improve model accuracy because it can increase the randomness of samples being trained.

E. Distributed ImPACT

We extend ImPACT to a distributed cache for multi-node training using data parallelism. Each node has an importance-informed prefetching module, a local cache client, a cache server, and a cache manager (detailed in section III-A). For distributed prefetching, each node statically caches non-overlapping samples with high variability in importance rankings. Nodes receive unique sample ID subsets via PyTorch's distributed sampler and prefetch using the process in section III-B. For distributed cache management, a key-value store tracks sample IDs and their cached node locations. The cache server prevents sample duplication across nodes to maximize data caching. When a cache client requests data, the cache manager checks the local cache first; if unavailable, it queries the key-value store. If found, data is retrieved from the respective node; otherwise, the request goes to the shared storage system.

IV. Implementation

We implement the client of ImPACT in Python based on PyTorch 1.8.0 [16]. We develop *ImpactRandomSampler*, a sampler for implementing importance-informed prefetching. We add a new *iCacheImageFolder* interface in the original *torch.utils.Dataset* class, which uses the gRPC framework to communicate with the ImPACT server. The client gets samples from the server through the *rpc_loader* interface and sends the H-list of samples to the server through the *update_ivpersample* interface. For the server, we implement it in Go language. We use the key-value structure to organize the samples in H-cache and L-cache. In addition to providing the usual functions of lookup/access/insert, the server also provides an interface to receive importance values and modules to handle dynamic packaging in cache management. The client consists of approximately 2,500 lines of code, while the server has about 3,500 lines. ImPACT is easy to deploy.

V. Evaluation

A. Experimental Setup

System configurations. We conduct the experiments on a training server with 2× AMD EPYC 7742 CPUs (64 cores), 512 GB DRAM, 10Gbps Ethernet, 8× NVIDIA A100 GPUs, one Intel SSD of 1 TB. The operating system is 64-bit Ubuntu 18.04.5. We train the DNN models using PyTorch 1.8.0 on the server and place the training datasets in an OrangeFS parallel file system [34], [35] in the same data center.

Workloads and datasets. We use a small dataset CIFAR10 [30] and a large dataset ImageNet [36]. With CIFAR10, we train ShuffleNet, ResNet18, MobileNet, and ResNet50. With ImageNet, we train VGG11, MnasNet, SqueezeNet, and DenseNet121. These eight models are widely used in the prior work [9], [10]. We do not consider natural language processing models (e.g., BERT) because they are typically computation-bound [10].

Compared systems. We compare ImPACT with Default, Base, Quiver [9], CoorDL [10], iLFU, and iCache [17]. *Default* is the PyTorch framework with an LRU cache and a built-in prefetching. *Base* adds computing-oriented importance sampling (CIS) to Default to only reduce computation. *Quiver* enhances cache management through sample substitutability, while *CoorDL* uses a static policy that does not evict cached data. We implement *iLFU* which uses the same prefetching as ImPACT but with an LFU cache replacement strategy. *iCache* uses the same cache management as ImPACT but relies on a history-based importance evaluation during prefetching. We also include *Oracle*, which assumes all cache accesses are hits, to show the lower bound of training time. We re-implement Quiver based on the paper descriptions, as it is not open-source. For CoorDL, we evaluate it by referring to its open-source. By default, the cache space is set at 20% of the dataset for all comparisons, consistent with previous work [9]. The memory used by ImPACT to buffer samples with unstable importance rankings is included in the total cache space for fairness. The initial ratio of $Size_{hcache}$ to $Size_{lcache}$ is 9:1. We use 6 workers to fetch data, a batch size of

TABLE I
MODEL ACCURACY ON CIFAR10

Models	Top-1 Acc. (%)			Top-5 Acc. (%)		
	Default	Base	ImPACT	Default	Base	ImPACT
ShuffleNet	87.76	87.10	86.96	99.59	99.57	99.57
ResNet18	92.70	92.17	92.14	99.81	99.83	99.80
MobileNet	92.37	92.06	92.01	99.87	99.81	99.77
ResNet50	89.91	89.40	89.36	99.68	99.68	99.69

TABLE II
MODEL ACCURACY ON IMAGENET

Models	Top-1 Acc. (%)			Top-5 Acc. (%)		
	Default	Base	ImPACT	Default	Base	ImPACT
VGG11	67.06	65.74	65.67	87.46	86.18	86.13
MnasNet	58.59	57.27	57.25	81.78	80.20	80.16
SqueezeNet	54.69	53.88	53.83	77.72	77.75	77.91
DenseNet121	75.35	74.78	74.79	92.57	92.42	92.39

256, and stripe the training datasets over four servers with a 64 KB stripe size in OrangeFS.

B. Accuracy Results

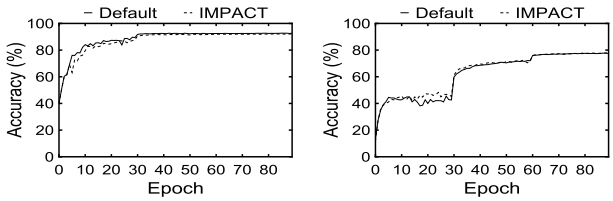
We first present the accuracy comparison for the four models on CIFAR10 with different cache schemes. Table I shows ImPACT achieves the comparable Top-1 and Top-5 accuracy compared to Default for all models. More specifically, ImPACT has 0.80%, 0.56%, 0.36%, and 0.55% accuracy losses on Top-1 accuracy and 0.02%, 0.01%, 0.10%, and -0.01% accuracy losses on Top-5 accuracy for ShuffleNet, ResNet18, MobileNet, and ResNet50, respectively. The accuracy loss is constrained within 1%. Table II shows the model accuracy on ImageNet. The accuracy loss of ImPACT yields satisfactory accuracy with less than 2% losses compared to Default.

We also observe that the average accuracy of ImPACT decreases by 0.03% compared with Base; the average accuracy of Base decreases by 0.56% compared with Default. This indicates that the accuracy drop caused by caching management in ImPACT (i.e., substitutability on cache missed L-samples) is less than the accuracy drop caused by the importance sampling algorithm. Additionally, sometimes the accuracy of ImPACT or Base is higher than that of Default, which indicates that focusing more on important samples can sometimes improve model accuracy [24].

Fig. 11(a) and 11(b) plot the Top-5 accuracy convergence curves for ResNet18 on CIFAR10 and SqueezeNet on ImageNet in 90 epochs, respectively. We can see that the curves of ImPACT are closely matched with the curves of Default, which provides the highest accuracy.

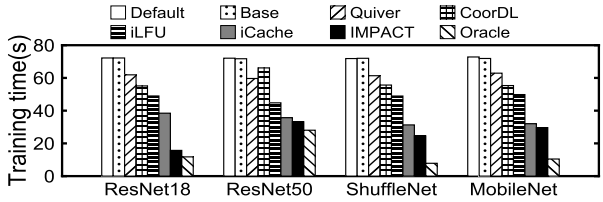
C. Performance Results

Training time. Fig. 12 shows the average training time per epoch for the DNN models. The average training time is defined as the total training time divided by the number of epochs. We have four observations.

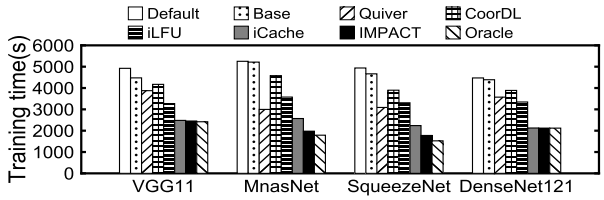


(a) ResNet18 on CIFAR10 (b) SqueezeNet on ImageNet

Fig. 11. Top-5 accuracy comparison.



(a) Training time of one epoch on CIFAR10



(b) Training time of one epoch on ImageNet

Fig. 12. Training time per epoch with various systems.

First, ImPACT outperforms all other six cache systems. On CIFAR10, it achieves maximum speedups of 4.6×, 4.6×, 3.9×, 3.5×, 3.1×, 2.5× over Default, Base, Quiver, CoorDL, iLFU, and iCACHE for the four models, respectively. On ImageNet, the maximum speedups are 1.3×-2.7× for the other four models. We also observe that on VGG11 and DenseNet121, ImPACT performs almost the same as Oracle. This demonstrates the effectiveness of the importance-informed prefetching and cache management methods in ImPACT. Second, different DNN models yield varied performance improvements. For example, ImPACT accelerates ResNet18 on CIFAR10 by 4.6×, but achieves a smaller acceleration (i.e., 2.2×) for ResNet50. This is because ResNet18 requires less GPU computation than ResNet50, leading to a larger I/O bottleneck and thus more space for performance improvement with ImPACT. Third, ImPACT reduces up to 59% training time compared to iCache. This is because ImPACT statically buffers samples with unstable importance rankings, which accelerates access to this data.

I/O time. Fig. 13 presents the I/O time per epoch in DNN training. Similar trends can be observed on ImageNet. ImPACT reduces I/O time by an average of 4.3× compared to Default, while other systems achieve speedups of 1.2×-2.4×. This explains why ImPACT has the shortest training time.

D. Impact of Individual Techniques

Fig. 14 shows the impact of each optimization in ImPACT on the total training time. *Base* is the system with the computing-oriented importance sampling (CIS) and an LRU

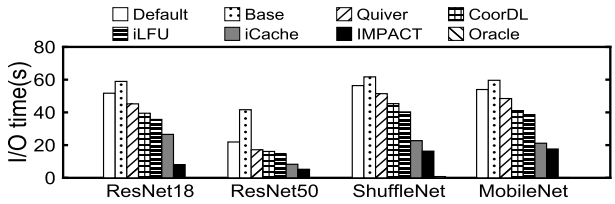
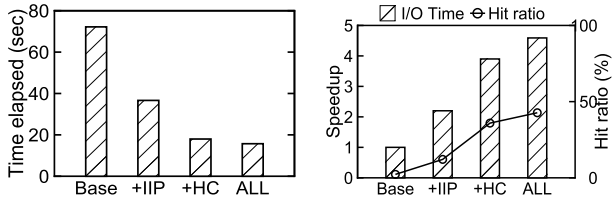
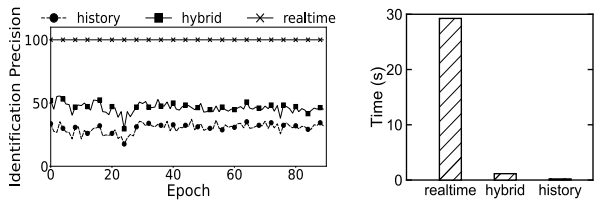


Fig. 13. Average I/O time of one epoch training on CIFAR10.



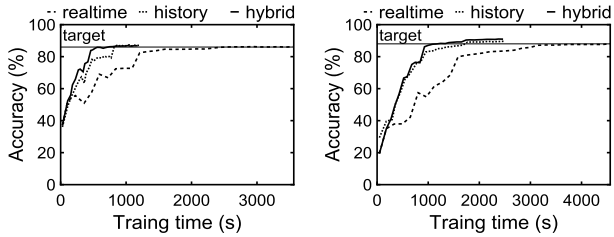
(a) Total training time. (b) I/O time and hit ratio.

Fig. 14. Performance impact of individual techniques.



(a) Identification precision. (b) Time overhead.

Fig. 15. Comparison of identification ability and time overhead across various methods.



(a) ShuffleNet (b) ResNet50

Fig. 16. Comparison of training time to achieve target model accuracy using various methods.

cache (CIS+LRU). *+IIP* denotes the version where importance-informed prefetching module is enable to reduce the number of prefetching requests for training data. *+HC* denotes the variant where H-cache is managed according to sample importance. *All* denotes the version with all optimizations including L-cache is enabled.

Fig. 14 shows the end-to-end training time, I/O time, and cache hit ratio after enabling each technique. *+IIP* achieves a 1.9× training time speedup and a 2.2× I/O time speedup over the Base for ResNet18. The primary reason is that while the *Base* method only reduces computation time, *+IIP* further reduces I/O time. When H-cache is enabled, the importance-informed algorithm caches more H-samples, significantly increasing the hit ratio to 37% in ImPACT. This results in a 3.9× training time speedup compared to the *Base*. With L-cache further enabled, *All* achieves a 4.6× training time speedup and increases the hit ratio to 43%. This is because missed L-samples are substituted by others in the L-cache, further reducing data loading time.

E. The Advantage of Hybrid Importance Evaluation

ImPACT utilizes the hybrid importance evaluation method as described in §III-B (denoted as *hybrid*) to effectively evaluate the importance of all training data. We compare it with two other methods: the history-based method [17], [24] (denoted as *history*) and the reaccessment-based method [22] (denoted as *realtime*). The *history* method chooses training samples based on historical importance values before each epoch starts, while the *realtime* method recalculates the precise and realtime importance values of all training samples by fetching the entire dataset for each epoch. We compare these three methods across three aspects: (1) their ability to identify important samples, (2) the additional time required for importance evaluation, and (3) the total training time needed to achieve the same target model accuracy.

Fig. 15 shows the ability and time overhead for identifying the top 1% most important samples for each epoch. We have the following two observations. First, *Hybrid* outperforms *history* in the ability of identifying the most important samples, with identification precision improved by 9% to 28%. Second, although *realtime* has the highest identification precision, its significant time overhead restricts training efficiency. Specifically, *realtime* completes the identification process 25× slower than *hybrid*. This is because the *hybrid* method only needs to fetch samples with unstable importance rankings to re-evaluate their importance for each epoch, whereas the *realtime* method requires fetching all the samples from memory or storage to re-evaluate all their importance values.

Fig. 16 shows the time taken to reach the same target model accuracy using the three methods. We observe that the *hybrid* method achieved the target accuracy the fastest. Compared to the *history* method, the *hybrid* method accelerates the time to reach the same accuracy by an average of 1.3×. This is because the *hybrid* method can evaluate the importance of training data more accurately than the *history* method, thereby reducing the time spent loading unimportant data. Compared to the *realtime* method, the *hybrid* method is up to 3.7× faster than the *realtime* method. This is because, although the *realtime* method can more accurately assess the importance of all samples in each epoch, its significant I/O time overhead leads to a considerable increase in training time.

F. Impact of Sample Substitution on Model Accuracy

To accelerate model training without significant accuracy degradation, ImPACT does not substitute H-samples with other samples when they are missed in the cache. However, when L-samples are missed, we can substitute them with samples in either H-cache or L-cache. Since either case shows the same I/O performance, we study its impact on model accuracy.

Table III shows the model accuracy with different sample substitution policies. Def denotes the policy without sample

TABLE III
MODEL ACCURACY ON CIFAR10

Models	Top-1 Acc. (%)			Top-5 Acc. (%)		
	Def	ST_{HC}	ST_{LC}	Def	ST_{HC}	ST_{LC}
ResNet18	92.70	91.89	92.14	99.81	99.77	99.80
ShuffleNet	87.76	86.73	86.96	99.59	99.52	99.57

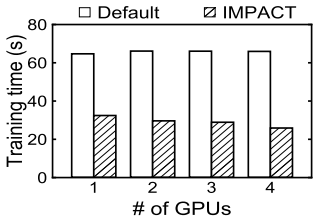


Fig. 17. Performance of ImPACT on a single machine using one to four GPUs.

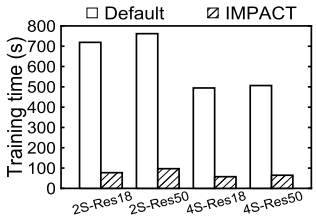


Fig. 18. Distributed training on CIFAR10. The label "nS" means the system with "n" servers.

substitution. ST_{HC} and ST_{LC} represent the policy to substitute the missed sample with H-sample and L-sample respectively. For ResNet18, the Top-1 model accuracy drops 0.81% and 0.56% with ST_{HC} and ST_{LC} compared to Def. For ShuffleNet, ST_{HC} and ST_{LC} yield a 1.03% and 0.80% Top-1 accuracy drop. These results show ST_{LC} has less impact on model accuracy. Similar trends can be observed in Top-5 accuracy for other models on CIFAR10 and ImageNet. We believe this is because ST_{HC} prevents L-samples from being trained. Although L-samples have a smaller impact on model gradient updates, completely excluding them from training still reduces the diversity of the training data, which consequently leads to a decrease in the model's generalization ability. In contrast, ST_{LC} only changes the order in which L-samples are trained, while maintaining the diversity of the training data, thus achieving higher accuracy. Thus, ImPACT takes this substituting method.

G. Single-Job Multi-GPU Training

Fig. 18 shows the per-epoch training time of ResNet50 on multi-GPUs with CIFAR10. We can find ImPACT always takes less training time than *Default* for all GPU configurations. Compared to *Default*, ImPACT achieves an average speedup of 2.3× with different numbers of GPUs. This shows ImPACT is effective in multi-GPU training. Another observation is that the total training time of *Default* remains similar as the number of GPUs increases. This is because increasing GPUs reduces a small amount of computing time but also incurs the communication overhead among multiple GPUs. In contrast, ImPACT has a slight performance improvement because it can reduce a lot of I/O time with the IIS and importance-aware cache management techniques, making the training process less I/O bound.

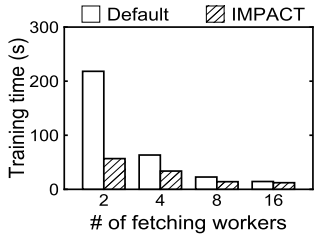


Fig. 19. Performance impact of number of workers.

H. Multi-Server Distributed Training

Fig. 18 shows the system performance of ImPACT in a distributed cloud platform with two and four nodes. Each server is equipped with one GPU and a cache space of 20% of the whole training data. As we do not have the permission to install the kernel module of OrangeFS on the cloud platform, we store the training data in an NFS server, similar to the approach in other cloud deep learning systems [37], [38]. The maximum read bandwidth of the NFS is about 10Gb/s. Although our scale is small, we argue that it is sufficient to demonstrate the efficiency of our system. Due to space limitations, we only show the results on CIFAR10 for ResNet18 and ResNet50. The results of other models lead to similar observations as described below.

ImPACT outperforms Default in distributed training, achieving at least 8.6× and 7.6× speedups on 2-server and 4-server configurations, respectively. The reasons are similar to those in single-server training (section V-C). Additionally, 4-server training time is about 1.5× lower than 2-server time. However, ImPACT's speedup is lower on 4-server than on 2-server configurations. For example, the speedup for ResNet18 drops from 9.3× to 8.5×. This is due to diminishing returns in cache hit ratio improvement with larger joint cache spaces: the cache hit ratio increases by 42% on 2-server and 23% on 4-server setups.

I. Parameter Sensitivity Analysis

Number of prefetching workers. PyTorch employs multiple workers to prefetch training data from storage systems. Fig. 19 shows the training time per epoch with various number of workers. We train ResNet18 on CIFAR10. As shown, ImPACT achieves a speedup over Default from 3.9× to 1.2× while the number of workers is increased from 2 to 16. This is because the proportion of data stall time decreases from 96.7% to 28.9% when the number of workers increases. Thus, the I/O benefits brought by ImPACT diminishes. However, since NVIDIA's AI-optimized servers (i.e., DGX-2) or general commercial cloud servers typically provide users with 3-4 CPU cores (6-8 vCPUs) per GPU [10], the number of workers set by users are usually limited to eight. Therefore, the prefetching effect is limited and iCache is still useful.

Cache size. Fig. 20 shows the training performance with various cache sizes for ShuffleNet on CIFAR10. First, we observe that with ImPACT, the speedup of training throughput ranges from

Cache	Speedup	Hit ratio	
		Ours	Default
20%	2.9×	43%	2%
40%	3.1×	61%	9%
60%	3.3×	74%	23%
80%	3.6×	88%	48%

Fig. 20. Performance impact of cache size.

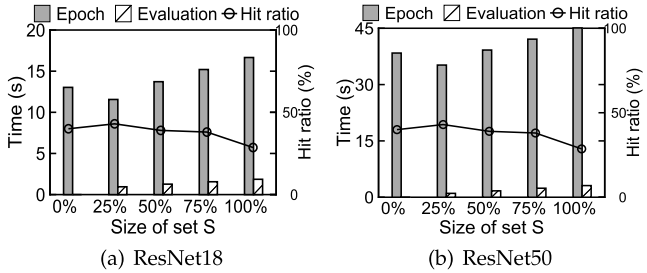


Fig. 21. Performance impact of various sizes of set *S*.

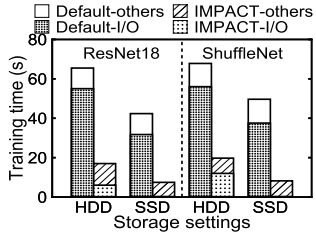


Fig. 22. Performance impact of underlying storage.

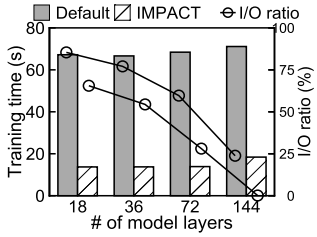


Fig. 23. Performance impact of model size.

2.9× to 3.6× as the cache size increases from 20% to 80% compared to the Default. Second, the cache hit ratio increases with cache size for both ImPACT and Default. Notably, even when the cache size is 80% of the dataset size, ImPACT still achieves a cache hit ratio 1.8× higher than of Default, explaining its superior performance.

The size of set *S*. The size of set *S* represents the number of samples whose importance values need to be re-evaluated before each epoch as described in §III-B. It affects the importance evaluation time (denoted as 'Evaluation'), cache hit ratios, and training time per epoch (denoted as 'Epoch'). We adjust the size of set *S* from 0% to 100% of the total cache space. As shown in Fig. 21, selecting a size of 25% for set *S* results in the highest cache hit rate and the shortest training time. Hence, we emparically chose 25% as the default value. Furthermore, it shows that regardless of the size of *S*, the epoch time outperforms the best counterparts (55s for ResNet18 and 59s for ResNet50). This demonstrates the robustness of the ImPACT system. Similar observations were made with other datasets and models.

Underlying storage. We evaluate ImPACT using two local storage media: SSD and HDD. Fig. 22 shows that ImPACT speeds up the training by 3.4× to 6× on the ResNet18 and ShuffleNet models compared to Default. We also observe that the average speedup on SSD (i.e., 5.8×) is greater than that on HDD (i.e., 3.6×), as ImPACT can completely eliminate the I/O time on the critical path when using SSDs.

Model size. To evaluate the efficiency of ImPACT on models with different scales, we create variants of the ResNet model by stacking different numbers of convolutional layers, ranging from 18 to 144 layers. Fig. 23 shows that ImPACT consistently outperforms the Default by 4.2× to 5.3× due to the reduced computation and I/O time. Although the I/O time ratio of the Default model decreases as the model size increases, ImPACT consistently achieves a lower I/O time ratio, even completely eliminating the I/O bottleneck on the 144-layer large model. This demonstrates that ImPACT is still efficient on large models. Additionally, due to the I/O-bound training of the system, the total time does not change significantly as the model size increases.

VI. Related Work

Prefetching Optimization. Popular deep learning frameworks (such as PyTorch [16] and TensorFlow [26]), along with numerous studies [27], [39], [40], [41], use multithreading to parallelize prefetching and computation, placing pre-fetched training data into a buffer to reduce prefetching time stalls. However, because prefetching can be time-consuming and the number of prefetching threads is limited [10], prefetching time cannot be fully covered by computation in scenarios where I/O is a bottleneck. Unlike these approaches, ImPACT reduces the prefetching of unimportant data.

Cache Management for DNN Training. Existing cache optimizations for DNN include aggregate the cache capacity from multiple nodes [42], using static cache or substitutability to improve the cache hit ratio [9], [10], [15], [20], and sharing the data in the cache between training jobs [10]. In contrast, ImPACT retains important samples in the H-cache and only substitutes cache missed unimportant samples (i.e., L-samples) because replacing important samples changes the distribution of H-samples decided by importance sampling algorithms, which may impact the final model accuracy.

Storage Optimizations for DNN Training. Existing methods use static packing (e.g., TFRecord [26], Webdataset [32], DIESEL [33], DLFS [43]) to address random reads. In contrast, ImPACT dynamically packages unimportant samples at runtime to enhance data access randomness during training. Google alleviates storage I/O bottleneck via data reusing [44], [45] which is orthogonal to our work and can be integrated.

VII. Conclusion

In this paper, we present ImPACT, a novel prefetching and caching system that reduces data stall time for I/O-bound DNN

training jobs. We introduce I/O-oriented importance sampling, applied to both prefetching and caching layers. Our importance-informed prefetching minimizes access to unimportant data, while importance-informed cache server further enhances cache hit ratios. Experimental results demonstrate that ImPACT has a negligible impact on training accuracy and accelerates DNN training time by up to 3.5× compared to state-of-the-art systems. As deep learning applications grow, we hope ImPACT will inspire future memory and storage systems for artificial intelligence.

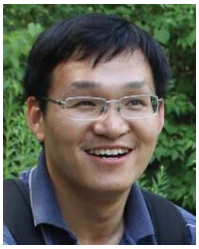
ACKNOWLEDGMENT

We sincerely thank the anonymous reviewers for their constructive suggestions.

REFERENCES

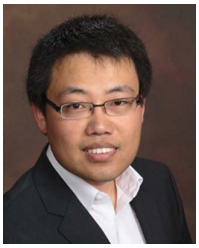
[1] C. Seifert et al., "Visualizations of deep neural networks in computer vision: A survey," in *Proc. Transparent Data Mining Big Small Data*, 2017, pp. 123–144.
[2] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," 2019, *arXiv:1901.11504*.
[3] W. Chen, S. He, H. Qu, and X. Zhang, "LeapGNN: Accelerating distributed GNN training leveraging feature-centric model migration," in *Proc. 23rd USENIX Conf. File Storage Technol.*, 2025, pp. 255–270.
[4] S. Yang et al., "GOPIM: GCN-oriented pipeline optimization for PIM accelerators," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2025.
[5] T. Wu et al., "AUTOHET: An automated heterogeneous reram-based accelerator for DNN Inference," in *Proc. 53rd Int. Conf. Parallel Process.*, 2024, pp. 1052–1061.
[6] P. Chen et al., "CSWAP: A self-tuning compression framework for accelerating tensor swapping in GPUs," in *Proc. IEEE Int. Conf. Cluster Comput.* Piscataway, NJ, USA: IEEE Press, 2021, pp. 271–282.
[7] P. Chen et al., "Accelerating tensor swapping in GPUs with self-tuning compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4484–4498, Dec. 2022.
[8] S. He et al., "HOME: A holistic GPU memory management framework for deep learning," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 826–838, Mar. 2022.
[9] A. V. Kumar and M. Sivathanu, "Quiver: An informed storage cache for deep learning," in *Proc. 18th USENIX Conf. on File and Storage Technologies*, 2020, pp. 283–296.
[10] J. Mohan, A. Phanishayee, A. Raniwala, and V. Chidambaram, "Analyzing and mitigating data stalls in DNN training," in *Proc. VLDB Endowment*.
[11] Google, "Open images dataset," 2018. [Online]. Available: https://github.com/cvdfoundation/open-images-dataset
[12] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2525–2534.
[13] T. B. Johnson and C. Guestrin, "Training deep models faster with robust, approximate importance sampling," *Adv. Neural Inf. Process. Syst.*, 2018.
[14] A. H. Jiang et al., "Accelerating deep learning by focusing on the biggest losers," 2019, *arXiv:1910.00762*.
[15] Y. Zhu et al., "Entropy-aware I/O pipelining for large-scale deep learning on HPC systems," in *Proc. 26th Int. Symp. Model., Anal., Simul. Comput. Telecommunication Syst.*, 2018, pp. 145–156.
[16] PyTorch, "PyTorch/Vision," 2021. [Online]. Available: https://github.com/pytorch/vision/tree/master/torchvision
[17] W. Chen et al., "iCache: An importance-sampling-informed cache for accelerating I/O-bound DNN model training," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2023, pp. 220–232.
[18] S. Pumma, M. Si, W. C. Feng, and P. Balaji, "Scalable deep learning via I/O analysis and optimization," *ACM Trans. Parallel Comput.*, vol. 6, no. 2, pp. 1–34, 2019.
[19] T. Kurth et al., "Exascale deep learning for climate analytics," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage, Anal.*, 2018, pp. 649–660.
[20] M. Dantas, D. Leitao, C. Correia, R. Macedo, W. Xu, and J. Paulo, "MONARCH: Hierarchical storage management for deep learning frameworks," in *Proc. IEEE Int. Conf. Cluster Comput.* Piscataway, NJ, USA: IEEE Press, 2021, pp. 657–663.
[21] I. Loshchilov and F. Hutter, "Online batch selection for faster training of neural networks," 2015, *arXiv:1511.06343*.
[22] S. Mindermann et al., "Prioritized training on points that are learnable, worth learning, and not yet learnt," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 15630–15649.
[23] S. Hu, W. Chen, Y. Yin, and S. He, "IOWA: An I/O-aware adaptive sampling framework for deep learning," in *Proc. 17th Int. Conf. Netw., Archit., Storage*, 2024.
[24] R. I. S. Khan et al., "SHADE: Enable fundamental cacheability for distributed deep learning training," in *Proc. 21st USENIX Conf. File Storage Technol.*, 2023, pp. 135–152.
[25] W. Chen et al., "IMPRESS: An importance-informed multi-tier prefix KV storage system for large language model inference," in *Proc. 23rd USENIX Conf. File Storage Technol.*, 2025, pp. 187–201.
[26] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implement.*, 2016, pp. 265–283.
[27] N. Dryden, R. Böhringer, T. Ben-Nun, and T. Hoefler, "Clairvoyant prefetching for distributed machine learning I/O," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–15.
[28] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 31–42, 2002.
[29] J. Chung, K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Ubershuffle: Communication-efficient data shuffling for SGD via coding theory," *Adv. Neur. Inf. Process. Syst.*, vol. 7, pp. 131–145, 2017.
[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Tech. Rep.*, 2009.
[31] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231n*, vol. 7, no. 7, 2015, Art. no. 3.
[32] A. Aizman, G. Maltby, and T. Breuel, "High performance I/O for large scale deep learning," in *Proc. Int. Conf. Big Data*, 2019, pp. 5965–5967.
[33] L. Wang et al., "DIESEL: A dataset-based distributed storage and caching system for large-scale deep learning training," in *Proc. 49th Int. Conf. Parallel Process.*, 2020, pp. 1–11.
[34] Orange, "Orange File System," http://www.orangefs.org/, 2021.
[35] S. He, X.-H. Sun, and B. Feng, "S4D-cache: Smart selective SSD cache for parallel I/O systems," in *Proc. 34th Int. Conf. Distrib. Comput. Syst.* Piscataway, NJ, USA: IEEE Press, 2014, pp. 514–523.
[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[37] S. Shi et al., "Towards scalable distributed training of deep learning on public cloud clusters," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 401–412, 2021.
[38] K. R. Jayaram et al., "FfDL: A flexible multi-tenant deep learning platform," in *Proc. 20th Int. Middleware Conf.*, 2019, pp. 82–95.
[39] X. Ruan and H. Chen, "Informed prefetching in I/O bounded distributed deep learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*. Piscataway, NJ, USA: IEEE Press, 2021, pp. 850–857.
[40] S. Lee et al., "Asynchronous I/O strategy for large-scale deep learning applications," in *Proc. 28th Int. Conf. High Perform. Comput., Data, Anal*. Piscataway, NJ, USA: IEEE Press, 2021, pp. 322–331.
[41] R. Macedo et al., "The case for storage optimization decoupling in deep learning frameworks," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2021, pp. 649–656.
[42] Z. Zhang, L. Huang, J. G. Pauloski, and I. T. Foster, "Efficient I/O for neural network training with compressed data," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 409–418.
[43] Y. Zhu, W. Yu, B. Jiao, K. Mohror, A. Moody, and F. Chowdhury, "Efficient user-level storage disaggregation for deep learning," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2019, pp. 1–12.
[44] D. Choi, A. Passos, C. J. Shallue, and G. E. Dahl, "Faster neural network training with data echoing," 2019, *arXiv:1907.05550*.
[45] C. Xu, S. Bhattacharya, M. Foltin, S. Byna, and P. Faraboschi, "Data-aware storage tiering for deep learning," in *Proc. IEEE/ACM Sixth Int. Parallel Data Syst. Workshop*, 2021, pp. 23–28.

Weijian Chen is currently working toward the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, China. His research interests include memory and storage systems for AI.



Shuibing He (Member, IEEE) received the Ph.D. degree in computer science and technology from Huazhong University of Science and Technology, in 2009. Currently, he is a ZJU100 Young Professor with the College of Computer Science and Technology, Zhejiang University, China. His research interests include intelligent computing, memory and storage systems, and processing-in-memory. He is a member of ACM.

Ruidong Zhang is currently working toward the master's degree with the College of Computer Science and Technology, Zhejiang University. His research interests include system for AI and storage systems.



Xuechen Zhang (Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from Wayne State University. Currently, he is an Associate Professor with the School of Engineering and Computer Science, Washington State University Vancouver. His research interests include the areas of storage systems and high-performance computing. He is a member of ACM.

Ping Chen is currently working toward the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, China. His research interests include intelligent computing and memory management for AI systems.

Siling Yang (Graduate Student Member, IEEE) is currently working toward the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, China. Her research interests include intelligent computing and processing-in-memory.

Haoyang Qu is currently working toward the master's degree with the College of Computer Science and Technology, Zhejiang University. His research interests include system for AI and storage systems.

Xuan Zhan is currently working toward the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, China. Her research interests include systems for AI and cluster resource scheduling.