Hourglass: Enabling Efficient Split Federated Learning with Data Parallelism

Qiang He
Huazhong University of Science and Technology, Wuhan, China
Swinburne University of Technology
Melbourne, Australia
hqiang@hust.edu.cn

Kaibin Wang
Swinburne University of Technology
Melbourne, Australia
kaibinwang@swin.edu.au

Zeqian Dong
Swinburne University of Technology
Melbourne, Australia
zdong@swin.edu.au

Liang Yuan
University of Adelaide, Adelaide
Australia
liang.yuan@adelaide.edu.au

Feifei Chen
Deakin University, Melbourne
Australia
feifei.chen@deakin.edu.au

Hai Jin
Huazhong University of Science and Technology, Wuhan, China
hjin@hust.edu.cn

Yun Yang
Swinburne University of Technology
Melbourne, Australia
yyang@swin.edu.au

Abstract

Federated learning (FL) has emerged as a promising solution for training machine learning (ML) models with privacy preservation. One of the key challenges is the computational burden on clients caused by training large-sized models. To tackle this challenge, researchers are trying to incorporate split learning into federated learning so that an ML model can be partitioned into two parts, one for training on clients and the other on a cloud server or an edge server. In current split FL systems, each client's server-side model partition is trained with an individual GPU on the fed server before model aggregation. This demands massive GPU resources and does not scale in real-world scenarios. This paper presents Hourglass, a new split FL system that trains clients' server-side model partitions on multiple GPUs with data parallelism. Unlike existing systems that maintain one model partition for each client and pass clients' intermediate features through corresponding model partitions, Hourglass maintains model partitions shared by clients and passes their intermediate features through GPUs in groups based on their differences. In this way, Hourglass prevents the overhead incurred by swapping model partitions in and out of GPUs and improves knowledge sharing between clients. Extensive experiments are conducted on four widely-used public datasets to evaluate the performance of Hourglass. The results demonstrate that, compared to state-of-the-art systems, Hourglass accelerates model convergence by up to 35.2x, and improves model accuracy by up to 9.28%.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Parallel computing methodologies**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

Keywords: Split federated learning, data parallelism, machine learning system

ACM Reference Format:
Qiang He, Kaibin Wang, Zeqian Dong, Liang Yuan, Feifei Chen, Hai Jin, and Yun Yang. 2025. Hourglass: Enabling Efficient Split Federated Learning with Data Parallelism. In *Twentieth European Conference on Computer Systems (EuroSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3689031.3717467

Feifei Chen is the corresponding author of this paper. Qiang He and Hai Jin are also with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster, and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China.

EuroSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1196-1/2025/03
https://doi.org/10.1145/3689031.3717467

1 Introduction

Federated learning (FL) has emerged as a promising solution for privacy-preserving model training [48]. It enables edge devices to collaboratively train a shared global model without revealing users' private data. This is achieved through the utilization of a fed server, i.e., a cloud server [53] or an edge server [47]. Federated learning [3, 53] is usually a multi-round process that involves multiple participating edge devices (often referred to as *clients*) training local models on their own data. In each training round, clients transmit

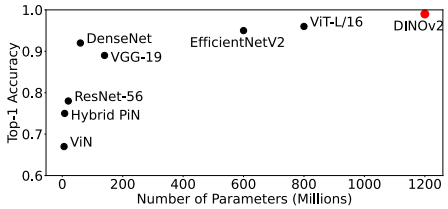


Figure 1. Size vs. top-1 accuracy on CIFAR-10 [34] for recent state-of-the-art image classification models [6, 23, 27, 30, 63, 65, 68].

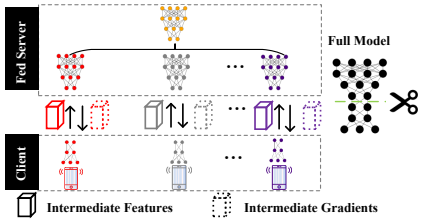


Figure 2. Split FL system: An overview.

their locally-trained models to the fed server. Based on these models, the fed server produces or updates the global model with an aggregation method such as FedAvg [43, 53]. The global model is then distributed to the clients for updating their local models. This process allows the fed server to aggregate knowledge from the clients' local models into the global model while maintaining data privacy.

In recent years, researchers have proposed various high-performance ML models, such as VGG [63], ResNet [23], CondenseNet [26], EfficientNetV2 [65], etc. These models often come in large sizes. As illustrated in Fig. 1 [58], ViN [30] reaches 65.1% top-1 accuracy with 0.53M parameters while DINOv2 [6] reaches 99.5% top-1 accuracy with 1,100M parameters on the CIFAR-10 dataset [34]. In an FL system, clients often struggle to train such large models with their limited computation resources [2, 4, 72, 79]. To overcome this limitation, researchers are starting to explore the potential of split learning (SL) to alleviate the computational burden on clients participating in FL [20, 38, 45, 67]. SplitFed [67] was the first attempt to facilitate *split federated learning* that combines split learning and federated learning. As depicted in Fig. 2, a model is sliced into two partitions: a client-side model and a server-side model partition. During each training round, the clients first train their client-side models on their local data and transmit the intermediate features to the fed server. The fed server then trains clients' corresponding server-side model partitions, one for each client, based on their intermediate features, and transmits intermediate gradients back to corresponding clients to update their client-side model partitions. Finally, the fed server aggregates server-side model partitions to produce a global server-side model partition for the next training round.

Split FL reduces the computational burden on clients by shifting the majority of the training workload to the fed server. However, it introduces two new challenges to the fed server. (1) **Computational Overhead.** In a conventional FL system, the clients train their local models, and the fed server aggregates their local model to produce the global model. There is no computational pressure on the fed server. However, in split FL, the fed server needs to allocate an individual GPU to train each client's server-side model partition. In a system that involves massive clients [28, 36], split FL would demand enormous GPU resources from the fed server.

(2) **Storage Overhead.** Deploying a large-sized model in a split FL system incurs significant storage overhead on the fed server, especially when there are many participating clients. The fed server must maintain a server-side model partition for each client during the training process. When there is a large number of clients, in some cases up to millions [28, 36], the fed server needs massive storage capacity to store their server-side model partitions. The issues of computational and storage overheads become even more critical in edge-assisted FL [73, 75]. In an edge-assisted FL system, instead of a cloud server, an edge server takes the responsibility of model training and aggregation. Unlike the cloud server, edge servers often suffer from constrained computation and storage resources due to their limited physical sizes [2, 62].

A possible solution to mitigate the computational and storage overheads is to reduce the sizes of server-side model partitions with model compression techniques [1, 54, 60]. For example, by representing model parameters with low-precision data types like 8-bit integer (`int8`) instead of high-precision data types like 32-bit floating point (`float32`) [14], the sizes of server-side model partitions can be reduced proportionally. As a result, the fed server can store clients' server-side model partitions with much lower storage demands. In the meantime, training these model partitions also takes much less time [24, 33, 50]. However, model compression often results in a substantial loss in model accuracy [13, 22, 29, 74]. For example, AlexNet suffers a 57.5% accuracy drop when they are quantized from 32-bit precision to 8-bit precision [74]. MobileNet also suffers a 10% accuracy drop when it is quantized from 32-bit precision to 8-bit precision [29]. In addition, when there is a large number of clients in the system, it is still a luxury for the fed server to train and store all their server-side model partitions.

This paper presents Hourglass, a new system designed to enable efficient split FL by reducing the computational and storage overheads with data parallelism. Hourglass can facilitate split FL when the fed server has access to one or multiple available GPUs for model training. Given only one GPU, a conventional split FL system like SplitFed [67] has to train clients' server-side model partitions in sequence with their intermediate features. This involves switching these model partitions in and out of the GPU, which takes time [25, 57, 83]. These model-switching overheads significantly impact the overall training time for each round and

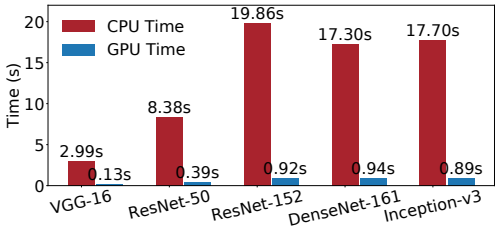


Figure 3. Average time taken per round to train ML models on CIFAR-10 [34] with a Nvidia RTX 3080 GPU (batch sizes = 32).

delays model convergence. To eliminate the model-switching overheads, Hourglass maintains only a single server-side model partition in the GPU shared by all the clients. In each training round, the fed server runs clients' intermediate features in sequence through this shared server-side model partition. This eliminates the need for model switching and reduces model-switching overhead during the training process. In the meantime, Hourglass does not need to store all clients' server-side model partitions for aggregation. This minimizes the storage overhead incurred. In addition, compared to aggregating multiple server-side model partitions using methods like FedAvg [53], Hourglass accelerates knowledge fusion by up to 23.21x through training a shared server-side model partition (§4.1).

When Hourglass has access to multiple available GPUs, say M GPUs, it can further optimize the training process with data parallelism. Similar to the single-GPU solution discussed above, Hourglass maintains a server-side model partition in each of the M GPUs, sharing a total of M server-side model partitions across all clients. During the training process, the fed server can leverage data parallelism to train these server-side model partitions simultaneously. In each training round, it distributes clients' intermediate features to the M GPUs for training these model partitions. In this way, Hourglass can process clients' intermediate features promptly as they arrive at the fed server and avoids model-switching overhead during the training process. In addition, we discovered that clients' intermediate features need to be distributed to GPUs strategically to ensure training performance - distributing clients' intermediate features randomly or based on their similarity can slow down model convergence and undermine model accuracy. Hourglass employs a new strategy to distribute clients' dissimilar intermediate features to the same GPU for processing. In this way, Hourglass accelerates model convergence by up to 35.2x, and improves model accuracy by up to 9.28%, compared to state-of-the-art split FL systems. The main contributions of this paper include:

- To the best of our knowledge, Hourglass is the first split FL system to systematically mitigate computational and storage overheads for the fed server.
- Instead of one server-side model partition for each client, Hourglass maintains server-side model partitions based on the number of available GPUs, i.e., one model partition per GPU. This reduces storage overhead on the fed server by up to 96.67% and eliminates model-switching overhead, which contributed to 43.15% of original training time.
- Instead of distributing similar intermediate features to GPUs for processing, Hourglass distributes dissimilar intermediate features to GPUs for processing. This accelerates model convergence and improves model accuracy by exploiting clients' data diversity.
- Instead of classic k-means clustering, Hourglass employs locality-sensitive hashing (LSH)-based clustering to group similar clients' intermediate features into clusters for distribution. This reduces the clustering time and allows Hourglass to process clients' intermediate features asynchronously as they arrive.
- Extensive experiments are performed using five ML models on four widely-used public datasets. The results show that Hourglass improves model accuracy by 2.18%-9.28%, and accelerates model convergence by 1.8x-35.2x.

2 Background and Motivation

Federated Learning. In recent years, FL has gained tremendous attention as a promising distributed ML approach with the ability to privacy-preserving model training [53]. It enables multiple clients to train their local models independently, while a fed server located in the cloud aggregates these local models to produce a global model. The objective of FL can be generally formulated as follows:

$$\min_{W} f(W) = \sum_{k=1}^{N} \frac{n_k}{n} F(W_k) \tag{1}$$

where N denotes the number of clients, n_k is the number of client k' data samples, and $n = \sum_{k=1}^{N} n_k$ denotes the total number of data samples across all clients. The local objective function $F(W_k)$ measures the empirical risk over client k's data distribution D_k, where W_k denotes its local model.

Split Federated Learning. In the FL system, clients train local models on their own data. Large-sized models have enormous parameters and demand massive computational resources for training. Training these models on clients with limited processing power, memory capacity, and energy is impractical if not completely infeasible. Fig. 3 illustrates the computation time needed for training various ML models on an Intel i7-12700 CPU and a Nvidia RTX 3080 GPU with a batch size of 32 on the CIFAR-10 dataset [34]. It shows that training models on the CPU consumes 18x to 23x more time than on the GPU. While the majority of mobile phones are equipped with CPUs only, training ML models on edge devices is impractical, if not impossible, in most real-world FL systems. To overcome this challenge, a promising solution is to integrate SL [59, 78] into FL systems. SplitFed [67], as the first split FL system, partitions the ML model for training into two parts: a client-side model partition and a server-side model partition. As illustrated in Fig. 2, the client-side model

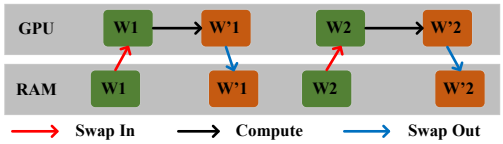


Figure 4. Swapping clients' models W_1 and W_2 in and out between RAM and GPU for training.

partition is trained on clients' devices and the server-side model partition is trained on the fed server. SplitFed offloads the majority of training workloads to the fed server and reduces the computation burden on resource-constrained clients like mobile phones. It allows large-sized models to be trained across clients in a federated manner.

Table 1. Computation time v.s. model-switching time during split FL under SplitFed [67], where models are trained across 300 clients with the support of a fed server equipped with one GPU, training one server-side model partition for each client.

Model	Dataset	Target Accuracy	Switching Time (%)	Computation Time (%)	Overall Time (s)
VGG-16 [63]	CIFAR-10 [34]	0.716	79.9	20.1	15,957
ResNet-50[23]	CINIC-10 [10]	0.453	39.9	60.1	42,109
CharCNN [81]	AG News [81]	0.872	13.1	86.9	2,524
VGG-16 [63]	Google Speech [76]	0.780	39.7	60.3	23,565

Computational Overhead. As illustrated in Fig. 2, in a split FL system, the fed server maintains a server-side model partition in a GPU for each client. When a client's intermediate features arrive, the fed server runs the intermediate features through the corresponding server-side model partition with a forward pass and a backward pass. When there are a lot of clients in the system, the fed server may not be able to allocate a dedicated GPU to train each client's server-side model partition. It needs to share available GPUs across the clients. After training a client's server-side model partition, the fed server can switch it out of the GPU and load the next client's server-side model partition into the GPU for training. Sharing GPUs across many clients requires switching their server-side model partitions in and out of the GPUs constantly. This can easily incur significant computational overhead because switching an ML model out of a GPU takes time. As illustrated in Fig. 4, the model-switching overhead is attributed to three main factors. First, the GPU environment for the current model must be cleaned up, such as freeing the GPU memory. Next, the GPU environment for the next model must be created and initialized. This includes launching processes and loading the PyTorch CUDA runtime. Finally, GPU memory must be allocated for the new model and the new model must be transmitted from system memory to GPU memory. To measure this overhead, we conducted an experiment with three popular DNN models trained on four public datasets under SplitFed, i.e., the state-of-the-art split FL scheme. As shown in Table 1, a significant proportion of the overall time, about 13.1%-79.9% (with an average of 43.15% across all datasets), is spent on switching

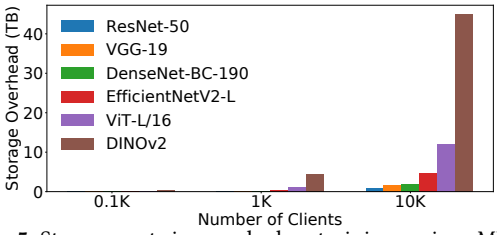


Figure 5. Storage costs incurred when training various ML models across different numbers of clients in conventional split FL systems.

server-side model partitions in and out of the GPU. These model-switching overheads can cause massive delays in the overall training process, which are even more pronounced when training larger ML models.

Storage Overhead. In a split FL system, the fed server must store a (usually very large) server-side model partition for each of the clients in the system. In this way, when it completes training clients' server-side model partitions with their intermediate features, it can aggregate all the server-side model partitions to produce a global server-side model partition shared across all the clients. Storing a large number of clients' server-side model partitions incurs prohibitive storage overhead. Fig. 5 illustrates the storage overhead incurred on the fed server when different ML models are trained on the CIFAR-10 dataset [34] in split FL systems of different sizes. We can see that the storage overhead increases notably with the number of clients. When training a DINOv2 [55] model for 10K clients, the fed server needs 40TB+ storage capacity to store their server-side model partitions. As the sizes of ML models continue to increase with the popularity of Transformer [70], the overwhelming demand for storage capacity makes it impractical, if not impossible, to facilitate split FL. Model compression techniques [1, 54, 60] can reduce the sizes of clients' server-side model partitions. However, it often undermines model accuracy [17, 50, 84]. When there is a large number of clients in the system, the fed server still requires massive storage resources.

Hourglass Solution. The root cause for the computational and storage overheads incurred for conventional split FL systems is the maintenance of N server-side model partitions, one for each of the N clients. The solution employed by Hourglass is to maintain server-side model partitions based on the number of available GPUs, i.e., M model partitions in total, one for each available GPU, rather than the number of clients (§4).

When the fed server has access to one available GPU, Hourglass maintains a single server-side model partition in the GPU shared by all clients. In each training round, it runs all clients' intermediate features through this GPU to train this shared server-side model partition without model switching (§4.1). Our experiment shows that this single-GPU solution reduces the storage overhead by up to 96.67% and the computational overhead by up to 88.07% (§4.1). Surprisingly,

it also reveals that the single GPU solution improves knowledge fusion between clients, evidenced by faster model convergence that requires up to 80.1% less training time (§4.1).

When the fed sever has access to $M(M > 1)$ available GPUs, it can accelerate the training process with data parallelism. Specifically, it maintains M server-side model partitions, one in each GPU, to be shared by all clients. In each training round, it distributes clients' intermediate features to these GPUs to train the M shared server-side model partitions in parallel without model switching. Through this data parallelism, Hourglass accelerates model convergence up to 31.6x compared to conventional split FL systems (§6.2). Taking a step further, instead of random and conventional distribution strategies, Hourglass employs a new strategy that distributes similar intermediate features to different GPUs for model training and accelerates model convergence further by up to 35.2x (§6.2).

3 Hourglass Overview

Hourglass is designed to accelerate split FL with minimal computational and storage overheads. Similar to conventional FL systems, it involves two fundamental operations in each training round: training client-side model partitions on the clients and training server-side model partitions on the fed server. Fig. 6 illustrates the architectural overview of Hourglass. In each training round, Hourglass goes through five steps. ① Client Forward: Clients run forward passes on their own client-side model partitions with their own datasets and then transmit the intermediate features produced by the last layers to the fed server. ② Fed Scheduling: The fed server allocates clients' intermediate features to its GPUs for forward passes. ③ Fed Training: The fed server trains server-side model partitions by running clients' intermediate features through the GPUs with forward and backward passes. When a backward pass is completed, the fed server transmits the model gradients to corresponding clients. ④ Fed Aggregation: The aggregator aggregates the server-side model partitions with the FedAvg method to update the global server-side model partition for the next training round. ⑤ Client Backward: After receiving model gradients, clients complete their backward process to update their client-side models. Here, ④ Fed Aggregation and ⑤ Client Backward are performed concurrently, which reduces the overall training time. In this paper, the ML model is denoted by w=$\{w^c; w^e\}$, where w^c is the client-side model partition and w^e is the server-side model partition.

4 Scheduling and Training

Hourglass maintains and trains a server-side model partition in each available GPU and adapts its client feature scheduling strategy based on whether the system operates in a single-GPU configuration (§4.1) or multi-GPU configuration (§4.2).

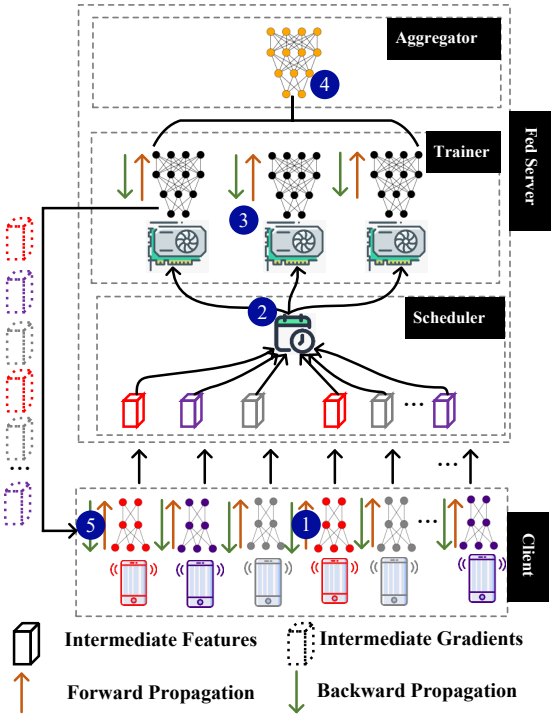


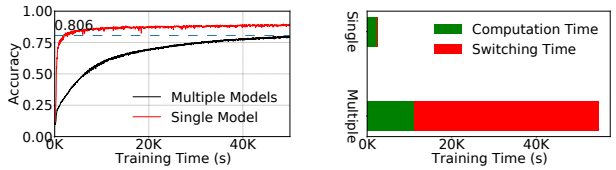
Figure 6. Hourglass Overview. This figure illustrates the workflow of Hourglass in a multi-GPU configuration. In single-GPU configurations, the Scheduler strategically runs clients' intermediate features through the sole GPU (§4.1). Such configurations are omitted from the figure for clarity.

4.1 Single-GPU Configuration

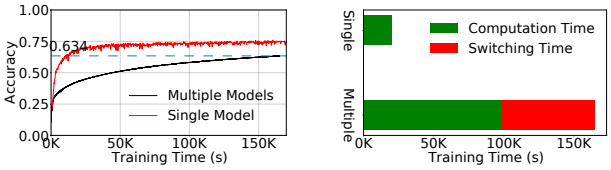
In a single-GPU configuration, e.g., when the fed server is deployed on an edge server whose GPU resources are limited, an intuitive solution is to maintain one server-side model partition in the GPU shared by all the clients. Training a shared model partition significantly reduces the storage demands on the fed server, as it eliminates the need to store and maintain server-side model partitions for individual clients.

Maintaining a shared server-side model partition for all clients, the fed server can simply run clients' intermediate features through the model partition in the same order as they arrive, following an FCFS (First Come, First Serve) strategy. When processing a client c_i's intermediate features, the fed server runs the intermediate features through the server-side model partition with a forward pass followed by a backward pass. At the end of the forward pass, it computes the predicted outputs. Next, it runs a backward pass to calculate gradients and update the parameters of the server-side model partition. Finally, it transmits the intermediate gradients to c_i to finalize the backward pass. The training iteration completes when the fed server completes processing all intermediate features received from the clients.

Fig. 7 presents the performance of training a shared server-side model partition for 300 clients. It shows that, when training a shared server-side model partition, the fed server



(a) VGG-16 on CIFAR-10



(b) ResNet-50 on CINIC-10

Figure 7. Comparison in time-to-accuracy between multiple and single server-side model partition in single-GPU configuration.

requires only 4.31% and 11.93% of the training time to achieve the same model accuracy for VGG-16 [63] and ResNet-50 [23] as training distinct model partitions for individual clients. This is expected because training a shared model partition does not incur model-switching overhead. However, the experimental results revealed something exciting, as can be observed in Fig. 7. When comparing only the model training time (excluding model-switching overhead), training a shared model partition takes less time to converge the model to the same accuracy as training multiple model partitions, i.e., 80.1% and 78.2% less time for VGG-16 and ResNet-50. This indicates that running clients' intermediate features through a shared model partition is more training-efficient than through their individual model partitions. In other words, Hourglass is more efficient in fusing the knowledge extracted from clients' training data than conventional split FL systems. Another exciting observation from the experiment is that training a shared model partition can converge the model to a higher accuracy than training multiple model partitions in conventional split FL systems. In the case of VGG-16 on CIFAR-10, as shown in Fig. 7, Hourglass can converge the model to an accuracy of 86.82% with a 6.76% accuracy advantage over conventional split FL systems. In the case of ResNet-50 on CINIC-10, the numbers are 75.92% vs 63.40%. This tells us that by training a shared server-side model partition, Hourglass achieves more effective knowledge fusion compared to conventional split FL systems. In other words, processing clients' intermediate features through forward and backward passes better exploits their data diversity - the core objective of FL - compared to model aggregation, which is typically employed in split FL systems and FL systems.

Data Heterogeneity. To further validate the reason for the advantages of Hourglass in fusing clients' knowledge, we conducted another experiment where Hourglass employs a different client feature scheduling strategy. In FL, clients' training data often differ in quantity, quality, and distribution,

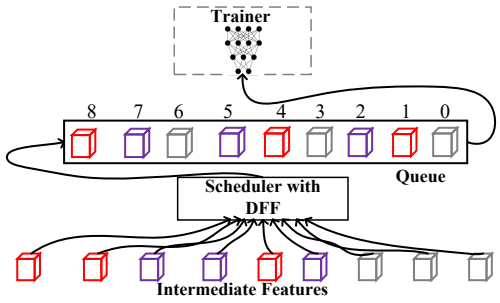
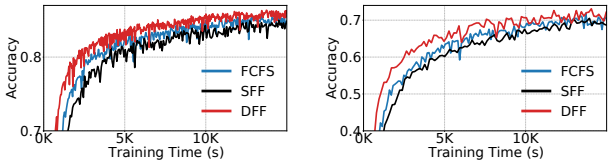


Figure 8. Overview of DFF (Dissimilar Feature First) strategy in a single-GPU configuration. Similar intermediate features are represented in the same color.

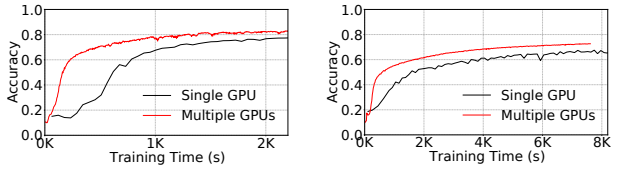


(a) CIFAR-10 (b) CINIC-10

Figure 9. Comparison in training time-to-accuracy when Hourglass trains a shared server-side model partition in a single-GPU configuration with the FCFS (First Come, First Served) strategy, the Similar Feature First (SFF) strategy, and the Dissimilar Feature First (DFF) strategy.

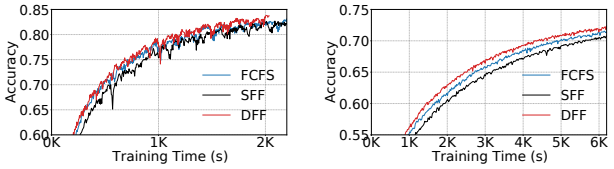
referred to as data heterogeneity [8, 36, 37]. Many studies have demonstrated its impact on the performance of FL systems [36, 39, 51]. In a split FL system, the heterogeneity of clients' training data translates directly into the heterogeneity of their intermediate features. In recent years, a series of methods have been proposed to exploit data heterogeneity [5, 56, 61, 82]. These methods employ centralized clustering algorithms, such as k-means, to group clients based on the similarity of their local models, sharing their knowledge first within the same group (with less data heterogeneity) and then between different groups (with more data heterogeneity). Following the same idea, we devise a new similarity-oriented client feature scheduling strategy named SFF (Similar Feature First). Its core idea is to always select the intermediate features most similar to the ones being processed for processing. However, this strategy contradicts our earlier findings, that is, the heterogeneity of the data should be exploited rather than minimized during the training process. Thus, we devised another new client feature scheduling strategy named DFF (Dissimilar Feature First) that always selects the intermediate features most different from the ones being processed. Fig. 8 provides an overview of this strategy.

Fig. 9 compares the SFF, DFF, and the FCFS strategy. We can see that Hourglass takes the most time to converge the models with the SFF strategy and the least time with the DFF strategy. This is consistent with our earlier findings on exploiting clients' data heterogeneity. In addition, the figure shows that Hourglass converges both models to the



(a) CIFAR-10 (b) CINIC-10

Figure 10. Comparison in time-to-accuracy between single-GPU configuration and 10-GPU configuration with the FCFS (First Come, First Serve) strategy. In the experiment, when a client's intermediate features arrive, Hourglass randomly allocates them to one of the ideal GPUs. If all GPUs are busy, the intermediate features are randomly allocated to one of the 10 GPUs for queued processing.



(a) CIFAR-10 (b) CINIC-10

Figure 11. Comparison in training time-to-accuracy between FCFS, SFF, and DFF for Hourglass in multi-GPU configuration.

highest accuracy with the DFF strategy. The reason behind these phenomena is that when training similar intermediate features consecutively, Hourglass adjusts the server-side model partition in a similar direction due to the close resemblance of these intermediate features. This can lead to an over-adjustment in a specific direction, potentially trapping the model in a local optimum and reducing its generalization capability. By feeding the model partition with dissimilar intermediate features, Hourglass can prevent the model from being stuck in a local optimum and allow it to learn more generalized features, and improve accuracy. Based on the results, Hourglass employs the DFF strategy to process clients' intermediate features in single-GPU configurations.

4.2 Multi-GPU Configuration

When multiple GPUs are available, say, M, Hourglass maintains and trains M server-side model partitions, one in each GPU. It distributes clients' intermediate features to these GPUs for parallel processing. After that, it aggregates the M server-side model partitions to produce or update the global server-side model partition.

By processing clients' intermediate features with multiple GPUs in parallel, Hourglass can train multiple server-side model partitions at the same time to reduce the training time in each training round. In addition, it prevents situations where some GPUs are overwhelmed while others remain underutilized. To validate the performance gains, we implemented an experiment where Hourglass trains a VGG-16 model and a ResNet-50 model across 300 clients, and compare the overall training time in a single-CPU configuration and 10-GPU configuration. In the 10-GPU configuration, the scheduler allocates clients' intermediate features to idle GPUs randomly for processing. Fig. 10 presents the results. We can see that training the models in a 10-GPU configuration can significantly accelerate model convergence.

Data Heterogeneity. In a multi-GPU configuration, Hourglass also needs to handle data heterogeneity properly, similar to a single-GPU configuration (§4.1). Similar to the SFF client feature scheduling strategy for single-GPU configuration (§4.1), in a M-GPU configuration, Hourglass can cluster clients' intermediate features into M groups with k-means based on their Euclidean distance. After that, each cluster is allocated to one of the M GPUs for processing. This SFF (Similar Feature First) strategy aims to process similar intermediate features with the same GPU. Another potential strategy is DFF (Dissimilar Feature First), which feeds a set of dissimilar intermediate features to the same GPU for training the server-side model partition in that GPU, similar to the DFF strategy for single-GPU configurations. Fig. 11 compares the model convergence when Hourglass trains server-side model partitions with three different client feature scheduling strategies, i.e., FCFS, SFF, and DFF. The results illustrate that the DFF strategy is the winner, outperforming both FCFS and SFF in model convergence and model accuracy. With the DFF strategy, Hourglass allocates clients' intermediate features with varying label distributions to each GPU. Recently, several approaches have leveraged clustering techniques to partition clients into distinct groups for knowledge sharing. IFCA [18] assumes that clients in an FL system can be easily partitioned into k disjoint clusters but does not provide the rationale behind this assumption or a specific clustering method. In contrast, Auxo [46] and Fed-CAM [52] employ k-means clustering to group clients based on their data characteristics. However, the use of k-means clustering introduces substantial computational overhead on the fed server. This is primarily due to the high-dimensional nature of ML models and the potentially large number of clients in the split FL system. Table 2 compares the computation time and the k-means clustering time for the DFF scheduling strategy when Hourglass trains different models on different datasets. We can see that the k-means clustering time accounts for approximately 8.1%-24.1% of total training time.

Table 2. Computation time v.s. clustering time during split FL.

Model	Dataset	Target Accuracy	Clustering Time (%)	Training Time (%)	Overall Time (s)
VGG-16 [63]	CIFAR-10 [34]	0.837	24.1	75.9	2,680
ResNet-50[23]	CINIC-10 [10]	0.691	14.3	85.7	5,182
CharCNN [81]	AG News [81]	0.872	14.2	85.8	1,679
VGG-16	Google Speech [76]	0.766	8.1	91.9	2,484

Client Heterogeneity. The clients in a split FL system can vary significantly in terms of their computational capabilities, communication bandwidth, and energy resources.

Some clients may have limited resources, while others possess ample resources. In each training round, clients require varying amounts of time to train their models. As a result, their intermediate features would arrive at the fed server at different times. Hourglass must wait for the arrival of all clients' intermediate features before it can run k-means clustering on these features. When there is a straggler in the system, the training performance would be profoundly affected [7, 40, 71]. These additional delays inevitably increase the overall convergence time of the FL system.

To address data heterogeneity and client heterogeneity systematically, Hourglass employs LSH (locality-sensitive hashing) [11] instead of k-means to cluster clients' intermediate features. LSH is a technique widely used in various data processing applications, e.g., nearest-neighbor search [12, 66], de-duplication [35], anomaly detection [80], and clustering [9, 11]. LSH has two unique features. First, it is designed to efficiently approximate the similarity between high-dimensional data points. It maps similar data points to the same or nearby hash buckets while minimizing the number of dissimilar data points assigned to the same bucket. This property enables Hourglass to identify similar intermediate features across clients. Second, in clustering applications, LSH is capable of assigning data points to buckets individually without the need for the entire dataset. Taking advantage of these features, Hourglass can cluster clients' features and assign them to GPUs for processing immediately when they arrive at the fed server without waiting for all clients' intermediate features. Fig. 12 illustrates the workflow of scheduling clients' intermediate features in 3-GPU configuration. It goes through three steps. First, the scheduler utilizes Euclidean distance-based LSH [11] to assign intermediate features to hash buckets as they arrive. Next, the scheduler allocates intermediate features within the same hash bucket to available GPUs for training, prioritizing those with greater computational capacities. Finally, the trainer runs clients' intermediate features through the server-side model partition for each GPU with forward and backward passes. In each training round, the scheduler generates a projection vector P and an offset value b for clients' intermediate features. Then, as a client's intermediate features arrive, a hash value is computed and stored in a hash table H. Next, based on the hash value, the scheduler assigns s_i to the corresponding GPU for processing. This process repeats until all features have been distributed across GPUs.

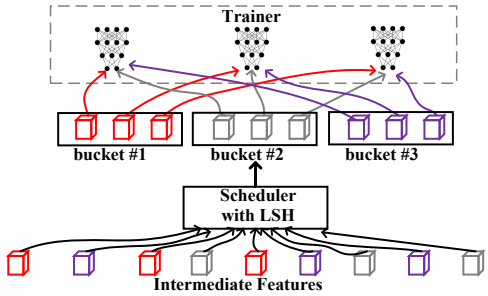


Figure 12. Scheduling 9 clients' intermediate features for processing with LSH in 3-GPU configuration.

5 Theoretical Analysis

Assumption 1. Each client k's loss function F is μ-strongly convex. That is, for all w_{k_1}, w_{k_2},

$$F(w_{k_2}) \geq F(w_{k_1}) + \langle \nabla F(w_{k_1}), w_{k_2} - w_{k_1} \rangle + \frac{\mu}{2} \|w_{k_2} - w_{k_1}\|^2 \quad (2)$$

Assumption 2. Each client k's loss function F is L-smooth.

$$\|\nabla F(w_{k_1}) - \nabla F(w_{k_2})\| \leq L \|w_{k_1} - w_{k_2}\| \quad (3)$$
$$F(w_{k_2}) \leq F(w_{k_1}) + \langle \nabla F(w_{k_1}), w_{k_2} - w_{k_1} \rangle + \frac{L}{2} \|w_{k_2} - w_{k_1}\|^2$$

Assumption 3. Stochastic gradients $g(\cdot)$ of $F(\cdot)$ are unbiased with the variance bounded by σ^2.

$$\mathbb{E}[g(w_k)] = \nabla F(w_k), \quad \mathbb{E}[\|g(w_k) - \nabla F(w_k)\|^2] \leq \delta_k^2. \quad (4)$$

Assumption 4. The expected squared norm of stochastic gradients is bounded by G^2.

$$\mathbb{E}[\|g(w_k)\|^2] \leq G^2 \quad (5)$$

Proposition 1. $w_*=\{w_*^c; w_*^e\}$ is the optimal global model, and $w_T=\{w_T^c; w_T^e\}$ is the global model obtained by Hourglass after T training rounds. From Proposition 3.5 of [21], we get

$$\mathbb{E}[F(w_T) - F(w_*)] \leq \frac{L}{2} \left(\mathbb{E}\|w_T^e - w_*^e\|^2 + \mathbb{E}\|w_T^c - w_*^c\|^2 \right) \quad (6)$$

Proposition 2. Under Assumptions 1 - 4, following the Proposition C.4 from [21], we get

$$\begin{aligned} &\mathbb{E}\left[F(w_{t+1}) - F(w_t)\right] \leq \mathbb{E}\left[\langle \nabla_{w^c} F(w_t), w_{t+1}^c - w_t^c \rangle\right] \\ &+ \frac{L}{2} \mathbb{E}\left[\|w_{t+1}^c - w_t^c\|^2\right] + \mathbb{E}\left[\langle \nabla_{w^e} F(w_t), w_{t+1}^e - w_t^e \rangle\right] \\ &+ \frac{L}{2} \mathbb{E}\left[\|w_{t+1}^e - w_t^e\|^2\right]. \end{aligned} \quad (7)$$

Before we give the convergence analysis, we first define η_t, which represents the learning rate in round t, and $\gamma = \frac{8L}{\mu} - 1$.

Theorem 1 (μ-Strongly Convex): Under Assumptions 1 - 4, $\eta_t = \frac{4}{\mu(\gamma+t)}$, it holds that:

$$\begin{aligned} \mathbb{E}[F(w_T) - F(w_*)] \leq & \frac{8LN \sum_{m=1}^{M} \sum_{k=1}^{N/M} (\frac{M^2}{N^2} + 1)(2\delta_{k,m}^2 + G^2)}{\mu^2(\gamma + T)} \\ &+ \frac{768L^2 \sum_{m=1}^{M} \sum_{k=1}^{N/M} (\frac{M}{N} + 1)(2\delta_{k,m}^2 + G^2)}{\mu^3(\gamma + T)(\gamma + 1)} \\ &+ \frac{L(\gamma + 1)\|w_0 - w_*\|^2}{2(\gamma + T)}. \end{aligned} \quad (8)$$

Proof. **Update Server-side Model.** Under Assumptions 1 - 4, if $\eta_t \leq \frac{1}{2L}$

$$\mathbb{E}\left[\|w^e_{t+1}-w^e_*\|^2\right] \le \mathbb{E}\left[\left\|w^e_t-w^e_*-\eta_t\nabla_{w^e}F\left(\left\{w^c_t,w^e_t\right\}\right)\right\|^2\right] + (\eta_t)^2\,\mathbb{E}\left[\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left\|g^e_{t,k,m}-\nabla_{w^e}F\left(\left\{w^c_t,w^e_t\right\}\right)\right\|^2\right] \tag{9}$$

$$\begin{aligned}\mathbb{E}\left[\left\|w^e_t-w^e_*-\eta_t\nabla_{w^e}F\left(\left\{w^c_t,w^e_t\right\}\right)\right\|^2\right] &\le \mathbb{E}\left[\|w^e_t-w^e_*\|^2\right]\\ &+(\eta_t)^2N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\mathbb{E}\left[\left\|\nabla_{w^e}F\left(\left\{w^c_{t,k,m},w^e_{t,k,m}\right\}\right)\right\|^2\right]\\ &-2\eta_t\mathbb{E}\left[\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left\langle w^e_t-w^e_*,\nabla_{w^e}F\left(\left\{w^c_{t,k,m},w^e_{t,k,m}\right\}\right)\right\rangle\right]\\ &\le \mathbb{E}\left[\|w^e_t-w^e_*\|^2\right]+(\eta_t)^2N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(\delta^2_{k,m}+G^2\right)\\ &-2\eta_t\sum_{m=1}^{M}\sum_{k=1}^{N/M}\mathbb{E}\left[F(w_{t,k,m})-F(w_*)+\frac{\mu}{4}\|x^e_t-x^e_*\|^2\right]\\ &-2\eta_t\sum_{m=1}^{M}\sum_{k=1}^{N/M}\mathbb{E}\left[L\|w^e_{t,k,m}-w^e_t\|^2\right]\end{aligned} \tag{10}$$

$$\mathbb{E}\left\|\sum_{m=1}^{M}\sum_{k=1}^{N/M}g^e_{t,k,m}-\nabla_{w^e}f\left(\left\{w^c_t,w^e_t\right\}\right)\right\|^2 \le N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\delta^2_{k,m}. \tag{11}$$

Hence, combining (10) and (11), there is

$$\mathbb{E}\left[\|w^e_{t+1}-x^e_*\|^2\right] \le \left(1-\frac{\eta_tN\mu}{2}\right)\Delta_t+\frac{(\eta_t)^2}{4}B_1+\frac{(\eta_t)^3}{8}B_2. \tag{12}$$

where $\Delta_{t+1}=\mathbb{E}\left[\|w^e_{t+1}-w^e_*\|^2\right]$,
$B_1=4N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right)$
and $B_2=192L\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right)$.

Consider a diminishing stepsize $\frac{N\eta_t}{2}=\frac{\beta}{\gamma_e+t}$, $\beta=\frac{2}{\mu}$, $\gamma_e=\frac{8L}{N\mu}-1$. $v=\max\left\{\frac{4B_1}{\mu^2}+\frac{8B_2}{\mu^3(\gamma_e+1)},(\gamma_e+1)\Delta^0\right\}$. From Theorem 1 in [43], we get

$$\Delta_{t+1}\le\frac{v}{\gamma_e+t+1}. \tag{13}$$

Hence, we have

$$\begin{aligned}\mathbb{E}\left[\|w^e_t-w^e_*\|^2\right]=\Delta^t &\le \frac{v}{\gamma_e+t}\le\frac{(\gamma_e+1)\mathbb{E}\left[\|w^e_0-w^e_*\|^2\right]}{\gamma_e+t}\\ &+\frac{16N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right)}{\mu^2(\gamma_e+t)}\\ &+\frac{1536L\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right)}{\mu^3(\gamma_e+t)(\gamma_e+1)}\end{aligned} \tag{14}$$

Update for Client-side Model. Under Assumptions 1 - 4, same as (12), if $\eta_t\le\frac{1}{2L}$, there is

$$\mathbb{E}\left[\|w^c_{t+1}-w^c_*\|^2\right]\le\left(1-\frac{\eta_t\mu}{2}\right)\Delta^t+\frac{(\eta^t)^2}{4}B_1+\frac{(\eta_t)^3}{8}B_2. \tag{15}$$

where $\Delta^{t+1}=\mathbb{E}\left[\|w^c_{t+1}-w^c_*\|^2\right]$,
$B_1=4N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M^2}{N^2}\left(2\delta^2_{k,m}+G^2\right)$
and $B_2=192L\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M}{N}\left(2\sigma^2_{m,k}+G^2\right)$.

$$\begin{aligned}\mathbb{E}\left[\|w^c_t-w^c_*\|^2\right]=\Delta^t &\le \frac{v}{\gamma_c+t}\le\frac{(\gamma_c+1)\mathbb{E}\left[\|w^c_0-w^c_*\|^2\right]}{\gamma_c+t}\\ &+\frac{16N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M^2}{N^2}\left(2\delta^2_{k,m}+G^2\right)}{\mu^2(\gamma_c+t)}\\ &+\frac{1536L\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M}{N}\left(2\delta^2_{k,m}+G^2\right)}{\mu^3(\gamma_c+t)(\gamma_c+1)}\end{aligned} \tag{16}$$

Merge the Server-side Models and Client-side Models. Under Proposition 1, we have

$$\begin{aligned}\mathbb{E}\left[F(w_T)-F(w_*)\right] &\le \frac{L(\gamma+1)\mathbb{E}\|w_0-w_*\|^2}{2(\gamma+T)}\\ &+\frac{8LN\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)(2\delta^2_{k,m}+G^2)}{\mu^2(\gamma+T)}\\ &+\frac{768L^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M}{N}+1)(2\delta^2_{k,m}+G^2)}{\mu^3(\gamma+T)(\gamma+1)}.\end{aligned} \tag{17}$$

□

Theorem 2 (General Convex): Under Assumptions 1 - 4, and $\eta_t\le\frac{1}{2L}$, there is

$$\begin{aligned}\mathbb{E}\left[F(w_T)-F(w_*)\right] &\le \frac{L\|w_0-w_*\|^2}{2(T+1)}\\ &+\frac{1}{2}\left(\frac{N\|w_0-w_*\|^2}{T+1}\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)(2\delta^2_{k,m}+G^2)\right)^{\frac{1}{2}}\\ &+\frac{1}{2}\left(\frac{24L\|w_0-w_*\|^2}{T+1}\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M}{N}+1)(2\delta^2_{k,m}+G^2)\right)^{\frac{1}{3}}\end{aligned} \tag{18}$$

Proof. **Update for Server-side Model.**
By the strong convexity assumption with $\mu=0$ and $\eta_t\le\frac{1}{2L}$, there is

$$\begin{aligned}\mathbb{E}\left[\|w^e_{t+1}-w^e_*\|^2\right] &\le \mathbb{E}\left[\|w^e_t-w^e_*\|^2\right]\\ &-2\eta_t\mathbb{E}\left[F(w_t)-F(w_*)\right]+(\eta_t)^2N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right)\\ &+24L(\eta_t)^3\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(2\delta^2_{k,m}+G^2\right).\end{aligned} \tag{19}$$

Update for Client-side Model

$$\begin{aligned}\mathbb{E}\left[\|w^c_{t+1}-w^c_*\|^2\right] &\le \mathbb{E}\left[\|w^c_t-w^c_*\|^2\right] \\ &- 2\eta_t\mathbb{E}\left[F(w_t)-F(w_*)\right] \\ &+ (\eta_t)^2 N\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M^2}{N^2}\left(2\delta^2_{k,m}+G^2\right) \\ &+ 24L(\eta_t)^3\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M}{N}\left(2\delta^2_{k,m}+G^2\right)\end{aligned} \tag{20}$$

Merge the Server-side Model and Client-side model

$$\begin{aligned}&\mathbb{E}\left[\|w_{t+1}-w_*\|^2\right] \le \mathbb{E}\left[\|w^e_{t+1}-w^e_*\|^2\right]+\mathbb{E}\left[\|w^c_{t+1}-w^c_*\|^2\right] \\ &= \mathbb{E}\left[\|w_t-w_*\|^2\right]-4\eta_t\mathbb{E}\left[F(w_t)-F(w_*)\right] \\ &+ (\eta_t)^2 N\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)\left(2\delta^2_{k,m}+G^2\right) \\ &+ 24L(\eta_t)^3\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M}{N}+1)\left(2\delta^2_{k,m}+G^2\right)\end{aligned} \tag{21}$$

From Lemma 8 in [44] we get

$$\begin{aligned}\mathbb{E}\left[F(w_T)-F(w_*)\right] &\le \frac{L\|w_0-w_*\|^2}{2(T+1)} \\ &+ \frac{1}{2}\left(N\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)\left(2\delta^2_{k,m}+G^2\right)\right)^{\frac{1}{2}}\left(\frac{\|w_0-w_*\|^2}{T+1}\right)^{\frac{1}{2}} \\ &+ \frac{1}{2}\left(24L\sum_{m=1}^{M}\sum_{k=1}^{N/M}(M/N+1)\left(2\delta^2_{k,m}+G^2\right)\right)^{\frac{1}{3}}\left(\frac{\|w_0-w_*\|^2}{T+1}\right)^{\frac{1}{3}}\end{aligned} \tag{22}$$

□

Theorem 3 (Non-convex). Let Assumptions 2 - 4 hold and $\eta_t \le \min\left\{\frac{1}{16L}, \frac{1}{8LN^2}\right\}$, we have

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}\left[\|\nabla_w F(w_t)\|^2\right] &\le \frac{4}{T}\left(F(w_0)-F(w_*)\right) \\ &+ \frac{8NL}{T}\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)\delta^2_{k,m}\sum_{t=0}^{T-1}(\eta_t)^2\end{aligned} \tag{23}$$

Proof. **Update for Server-side Model.** By Lemma E.3 in [21], we have

$$\begin{aligned}\mathbb{E}\left[\langle\nabla_{w^e}F(w_t), w^e_{t+1}-w^e_t\rangle\right] &\le 8N\eta_t L^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\eta_t\delta_{k,m})^2 \\ &+ \left(-\frac{\eta_t}{2}+8N^2(\eta_t)^3L^2\right)\|\nabla_{w^e}f(w_t)\|^2\end{aligned} \tag{24}$$

$$\begin{aligned}\mathbb{E}\left[\|w^e_{t+1}-w^e_t\|^2\right] &= N(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}\mathbb{E}\left\|g^e_{t,k,m}\right\|^2 \\ &\le N(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(\mathbb{E}\left[\|g^e_{t,k,m}-g^e_t\|^2\right]+\mathbb{E}\left[\|g^e_t\|^2\right]\right) \\ &\le N(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(1+8(\eta_t L)^2\right)\left(2\|\nabla_{w^e}F(w_t)\|^2+\sigma^2_{m,k}\right)\end{aligned} \tag{25}$$

Update for Client-side model.

$$\begin{aligned}&\mathbb{E}\left[\langle\nabla_{w^c}F(w_t), w^c_{t+1}-w^c_t\rangle\right] \le 8N\eta_t L^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M}{N}\eta_t\sigma_{m,k})^2 \\ &+ \left(-\frac{\eta_t}{2}+8N(\eta_t)^3L^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(M/N)^2\right)\|\nabla_{w^c}f(w_t)\|^2\end{aligned} \tag{26}$$

$$\begin{aligned}&\mathbb{E}\left[\|w^c_{t+1}-w^c_t\|^2\right] \\ &\le N(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}\frac{M^2}{N^2}\left(1+8(\eta_t L)^2\right)\left(2\|\nabla_{w^c}F(w_t)\|^2+\delta^2_{k,m}\right)\end{aligned} \tag{27}$$

Merge Server-side Model and Client-side Model. Under Proposition 2, $\eta_t \le \frac{1}{16L}$ and then $\eta_t \le \frac{1}{8LN^2}$, we have

$$\begin{aligned}&\mathbb{E}\left[F(w_{t+1})-F(w_t)\right] \le \mathbb{E}\left[\langle\nabla_{w^c}f(w_t), w^c_{t+1}-w^c_t\rangle\right] \\ &+ \frac{L}{2}\mathbb{E}\left[\|w^c_{t+1}-w^c_t\|^2\right] \\ &+ \mathbb{E}\left[\langle\nabla_{w^e}F(w_t), w^e_{t+1}-w^e_t\rangle\right]+\frac{L}{2}\mathbb{E}\left[\|w^e_{t+1}-w^e_t\|^2\right] \\ &\le \left(-\frac{\eta_t}{2}+8N(\eta_t)^3L^2\right)\|\nabla_w F(w_t)\|^2 \\ &+ 8N\eta_t L^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\eta_t\delta_{k,m})^2(\frac{M^2}{N^2}+1) \\ &+ LN(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}\left(1+8(\eta_t L)^2\right)\|\nabla F(w_t)\|^2 \\ &+ \frac{LN(\eta_t)^2}{2}\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)\left(1+8(\eta_t L)^2\right)\delta^2_{k,m} \\ &\le -\frac{\eta_t}{4}\|\nabla_w F(w_t)\|^2+2NL(\eta_t)^2\sum_{m=1}^{M}\sum_{k=1}^{N/M}(\frac{M^2}{N^2}+1)\delta^2_{k,m}\end{aligned} \tag{28}$$

Hence, we have

$$\begin{aligned}\eta_t\|\nabla_w f(w_t)\|^2 &\le 8NL(\eta_t)^2\sum_{m=1}^{M}\sum_{K=1}^{N/M}(\frac{M^2}{N^2}+1)\delta^2_{k,m} \\ &+ 4\mathbb{E}\left[F(w_{t+1})-F(w_t)\right]\end{aligned} \tag{29}$$

Finally, we get

$$\frac{1}{T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}\left[\|\nabla_w F(w_t)\|^2\right] \leq \frac{4}{T}\left(F(w_0)-F(w_*)\right) + \frac{8NL}{T}\sum_{m=1}^{M}\sum_{K=1}^{N/M}\left(\frac{M^2}{N^2}+1\right)(\delta_{m,k}^2)\sum_{t=0}^{T-1}(\eta_t)^2. \quad (30)$$

□

6 Experiments

To validate the usefulness of Hourglass and evaluate its performance, a series of experiments are conducted using five ML models across three popular mobile AI applications: image classification, text classification, and speech recognition., they are trained on four widely-used datasets.

6.1 Experimental Setup

System Setup. The fed server is equipped with an Intel Xeon Gold 6248R CPU, 1TB memory, with access to 10 RTX 3080 GPUs and 5 RTX 2060 GPUs. Each client has an Intel i7-12700 CPU and 16GB memory.

Models and Datasets. Five ML models covering three popular AI applications are trained on four public datasets.

1. **Image Classification**. VGG-16 [63], ResNet-50 [23], and VIT [69] models are trained on the CIFAR-10 [34] and CINIC-10 [10] datasets, both of which are widely used in FL research. The CIFAR-10 dataset comprises 50,000 training images and 10,000 test images of color images divided into 10 classes. Similarly, CINIC-10 contains 90,000 training images and 90,000 test images across 10 classes.
2. **Text Classification**. The CharCNN [81] and LSTM [15] models are trained on the AG News dataset [81], a popular dataset for text classification. This dataset consists of 120,000 training samples and 7,600 test samples from four classes derived from the original corpus.
3. **Speech Recognition**. The VGG-16 [63] model is trained on the Speech Commands dataset [76]. This dataset comprises 65,000 one-second-long recordings of 30 short keywords, which are commonly used in voice-activated systems and speech recognition tasks. Each keyword contains 80% training samples and 20% testing samples.

Parameters. All models were trained with stochastic gradient descent (SGD). For the CIFAR-10 and CINIC-10 datasets, the learning rate (lr) is set to 0.01, with a momentum of 0.9 and a weight_decay (wd) of $5e^{-4}$. A lr of 5 is applied for the AG News dataset, while the Speech Commands dataset is trained with a lr of 0.001 and a wd of e^{-4}.

Baselines. FL and SplitFed are implemented as baselines for comparison with Hourglass.

1. **FL** [53]. In the system, clients train their entire local models on their own datasets in each training round. After that, they transfer the updated local models to the fed server for aggregation. The fed server then integrates clients' local models and updates the global model.
2. **SplitFed** [67]. SplitFed is the state-of-the-art split FL system. Its training process goes through three main steps. (1) Clients train the client-side model partitions on their local data and then transfer the intermediate features produced to the fed server. (2) The fed server maintains a server-side model partition for each of the clients. It trains these server-side model partitions using corresponding clients' intermediate features. (3) Next, it aggregates all the server-side model partitions to produce a global server-side model partition for the next training round.

Table 3. Performance comparison of speedup over FL and SplitFed, where FCFS, SFF, and DFF represent First Come First Serve feature scheduling strategy, Similar Feature First strategy, and Dissimilar Feature First strategy in Hourglass.

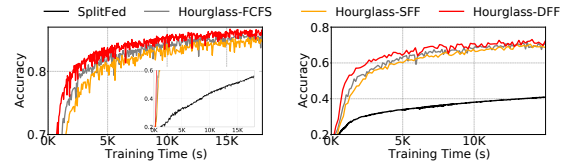
Dataset	Model	Target Accuracy	Speedup over FL			Speedup over SplitFed		
			FCFS	SFF	DFF	FCFS	SFF	DFF
CIFAR-10	VGG-16	0.806	41.3×	38.8×	52.7×	8.1×	7.1×	10.2×
	ResNet-50	0.742	73.3×	72.3×	78.8×	31.6×	26.5×	35.2×
	VIT	0.767	32.7×	31.8×	38.2×	7.1×	6.9×	8.3×
CINIC-10	VGG-16	0.725	22.9×	20.3×	26.8×	14.6×	11.5×	16.3×
	ResNet-50	0.633	53.8×	46.4×	66.2×	23.6×	19.1×	26.4×
	VIT	0.677	21.9×	21.1×	29.1×	5.2×	5.0×	6.9×
AG News	CharCNN	0.872	9.5×	7.5×	13.8×	3.6×	2.1×	4.2×
	LSTM	0.901	6.5×	5.8×	8.9×	2.1×	1.8×	2.7×
Speech Commands	VGG-16	0.766	6.8×	6.2×	9.6×	2.2×	1.9×	3.1×

6.2 Overall Evaluation

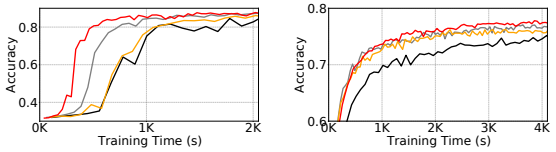
Table 3 presents a summary of the time-to-accuracy performance for all datasets. Fig. 13 and Fig. 14 show the convergence time of the models.

Training Speedups. Table 3 compares the training speedups achieved by Hourglass (with FCFS, SFF, and DFF) over FL [53] and SplitFed [67] across four datasets in a 10-GPU configuration. These schemes are represented as Hourglass-FCFS (with First Come First Serve feature scheduling strategy), Hourglass-SFF (with Similar Feature First strategy), and Hourglass-DFF (with Dissimilar Feature First strategy). There are three key observations. 1) Compared with FL, SplitFed reduces the training time required to converge models to target accuracies. This is expected because SplitFed shifts the majority of the training workload to the fed server. 2) Hourglass outperforms both FL and SplitFed significantly, despite its feature scheduling strategy. Compared with FL and SplitFed, Hourglass achieves 5.8x-73.3x, and 1.8x-35.2x speedup gains, respectively. This indicates its superiority in accelerating model convergence. 3) Among the three feature scheduling strategies, DFF achieves the greatest speedups. This shows the ability of DFF to leverage data heterogeneity in clients' intermediate features.

Model Convergence: Single-GPU Configuration. Fig. 13 illustrates the model convergence and accuracy in a single-GPU configuration. Compared to SplitFed, Hourglass takes



(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10

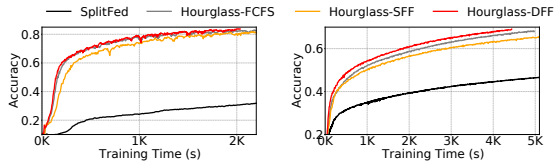


(c) CharCNN on AG News (d) VGG-16 on Speech

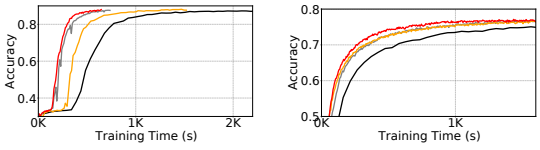
Figure 13. Model convergence and accuracy with a single GPU processing all clients' intermediate features. In FL, clients have to train entire ML models with their CPUs. Model convergence is extremely slow. Thus, FL is not included in this figure.

much less time to converge a model, which attributes to its ability to avoid model-switching overhead and accelerate knowledge fusion. Among the three feature distribution strategies, DFF enables Hourglass to achieve the maximum training speedup. If we take a look at the final model accuracy, we can see that Hourglass can also converge a model to a higher accuracy than SplitFed. Take VGG-16 on CIFAR-10 for example. As shown in Fig. 13(a), Hourglass-DFF can converge the model to 86.82%, while SplitFed, FCFS, and SFF can only reach 80.6%, 86.11%, and 85.69%, respectively. The advantage of DFF can also be observed when Hourglass trains other models, as demonstrated in Fig. 13(b) - Fig. 13(d). Across all four models, Hourglass-DFF achieves an average accuracy advantage of 6.89%, 0.77%, and 1.12% over SplitFed, FCFS, and SFF, respectively. This experiment validates the ability of Hourglass to utilize limited GPU resources, in particular, DFF which leverages data heterogeneity.

Model Convergence: Multi-GPU Configuration. Fig. 14 illustrates the convergence of the models in a 10-GPU configuration when they are trained in different systems except FL. It shows that Hourglass takes much less time to converge a model to a high accuracy than SplitFed. When we compare FCFS, SFF, and DFF, we can see that Hourglass takes the least time to converge a model with the DFF strategy. This is consistent with Table 3. We can also see that DFF can converge a model to a higher accuracy than FCFS and SFF. For example, Fig. 14(c), DFF is capable of converging CharCNN on AG News to an accuracy of 88.17% when FCFS and SFF can only reach 87.47% and 87.14%, respectively. The same can be observed in Fig. 14(d) with VGG-16 on Speech Commands. DFF can converge the model to an accuracy of 78.2% when the numbers for FCFS and SFF are 77.13% and 76.77%, respectively. In particular, DFF achieves a remarkable 9.28% accuracy improvement over SplitFed, as shown in Fig. 14(b). These validate the advantages of Hourglass in both model

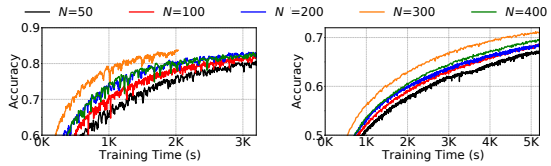


(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10

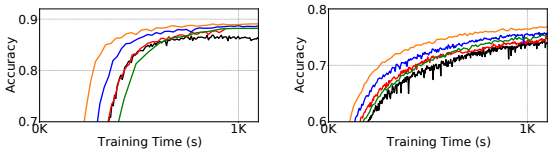


(c) CharCNN on AG News (d) VGG-16 on Speech

Figure 14. Model convergence and final accuracy with 300 clients in a 10-GPU configuration. Similar to Fig. 13, FL is not included in this figure for its poor performance.



(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10



(c) CharCNN on AG News (d) VGG-16 on Speech

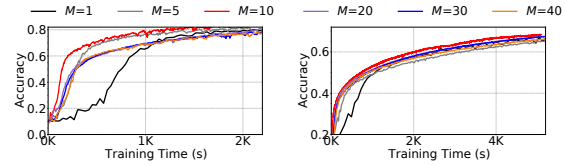
Figure 15. Model convergence and accuracy with different numbers of clients in 10-GPU configuration.

convergence and model accuracy, especially when powered by the DFF strategy.

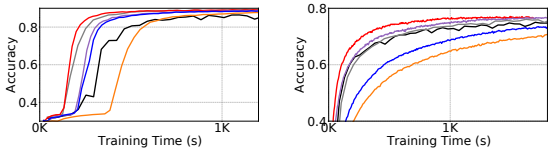
6.3 In-depth Evaluation

Hourglass-DFF demonstrates its advantages over FL, SplitFed, as well as Hourglass-FCFS and Hourglass-SFF. This section evaluates its performance in various split FL scenarios through a series of experiments. In this section, we simply refer to Hourglass-DFF as Hourglass for simplicity.

Impact of Number of Clients (N). In this experiment, the number of participating clients (N) varied between 50 and 400 to evaluate the time-to-accuracy performance of Hourglass. The fed server is equipped with 10 GPUs. Fig. 15 displays the results. When there are more clients in the system, Hourglass can fuse their knowledge more efficiently by leveraging their data heterogeneity when training their shared server-side model partitions. In each training round, Hourglass needs more time to process clients' intermediate features with the available GPUs, but it is worth the time. Thus, when N increases from 50 to 300, we observe a

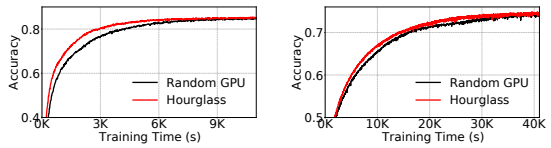


(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10

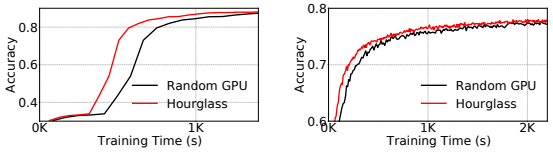


(c) CharCNN on AG News (d) VGG-16 on Speech

Figure 16. Model convergence and accuracy with different numbers of GPUs across 300 clients.



(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10



(c) CharCNN on AG News (d) VGG-16 on Speech

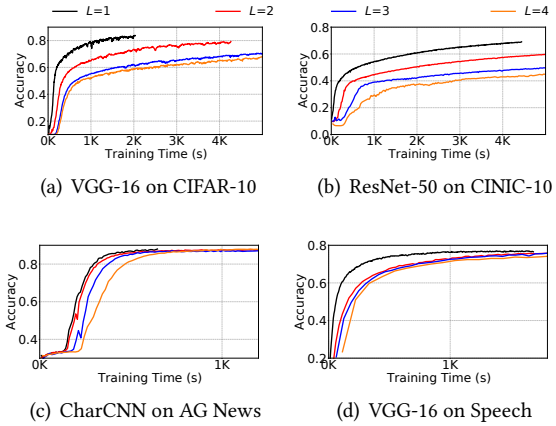
Figure 17. Model convergence with heterogeneous GPUs.

speedup in model convergence, i.e., 2.6x, 1.7x, 2.1x, and 2.2x for VGG-16 on CIFAR-10, ResNet-50 on CINIC-10, CharCNN on AG News, and VGG-16 on Speech. However, as N continues to increase from 300, the extra time Hourglass needs to process clients' intermediate features starts to outweigh the speedup of knowledge fusion. As a result, model convergence slows down. This experiment reveals that Hourglass needs a proper number of GPUs to achieve its top performance when training models for a large number of clients.

Impact of Number of GPUs (M). In a multi-GPU configuration, the fed server is equipped with multiple GPUs for training clients' shared server-side model partitions. In this experiment, we vary the number of GPUs (M) from 1 to 40 to evaluate the time-to-accuracy performance of Hourglass training models for 300 clients. Fig. 16 presents the results. Interestingly, increasing the number of GPUs does not always lead to faster model convergence and higher model accuracy. In this experiment, Hourglass always takes the least time to converge a model to the target accuracy with $M = 10$. We investigated the system and found the reason. More GPUs enable greater data parallelization, which reduces the overall time Hourglass needs to process all intermediate features in each training round. However, spreading clients' intermediate features across more GPUs for process slows down the fusion of the knowledge contained in the intermediate features processed by each individual GPU, taking Hourglass more training rounds to converge models. Consequently, model convergence slows down when M exceeds 10 and increases further. It tells us that Hourglass can strike a proper balance between model convergence and GPU resource demands.

Hourglass with Heterogeneous GPUs. In this experiment, the fed server is equipped with heterogeneous GPUs, including five RTX 3080 and five RTX 2060. During each training round, Hourglass utilizes LSH to assign clients' intermediate features in the same bucket to available GPUs for processing, prioritizing those with high computation capacity (§4.2). For comparison purposes, we implement a random feature distribution strategy named Random GPU that allocates the clients' intermediate features in the same bucket to *random* GPUs for processing without considering their computation capacity. As shown in Fig.17, Hourglass manages to speed up the convergence of all four models over Random GPU. For example, Fig.17(a) demonstrates that Hourglass converges VGG-16 to an accuracy of 85.1% within 7,918.9 seconds, reducing the training time by 44.3% compared to Random GPU. Its advantage over Random GPU in training other models is also remarkable, evidenced by a training time reduction of 22.1%, 34.8%, and 56.8% when converging ResNet-50 (on CINIC-10), CharCNN (on AG News) and VGG-16 (on Speech Command) to accuracies of 74.10%, 89.03%, and 78.66%.

Impact of Number of Trained Layers (L). In Hourglass, clients train a proportion of the model layers on their data and the fed server trains the rest of the model layers. When there are more layers in the server-side model partition(s), more knowledge is shared across the clients. This should lead to faster model convergence and higher model accuracy. In the meantime, a client-side model partition with fewer layers would reduce clients' training workloads and should accelerate model training. We conducted an experiment with 10 GPUs and 300 clients to validate these. In the experiment, we varied the number of layers (L) in the client-side model partition from 1 to 4 to evaluate the time-to-accuracy performance of Hourglass. Fig. 18 presents the results. It shows that a small L improves both model convergence and model accuracy substantially. Take VGG-16 on CIFAR-10 for example. As shown in Fig. 18(a), when $L = 1$, Hourglass manages to converge the model to an accuracy of 79.27% within 972.2 seconds, taking 69.0%, 89.9%, and 92.3% less time, respectively, than $L = 2$, $L = 3$, and $L = 4$. This experiment validates the benefits of offloading clients' training workloads to an external worker. The figure also clearly shows that by training the most model layers on the fed server with $L = 1$, Hourglass manages to converge a model to the highest accuracy. Let us take a look at ResNet-50 this time, whose convergence over time is illustrated in Fig. 18(b). Given about 4,500 seconds,



(a) VGG-16 on CIFAR-10 (b) ResNet-50 on CINIC-10

(c) CharCNN on AG News (d) VGG-16 on Speech

Figure 18. Model convergence and accuracy with different numbers of layers for 300 clients in 10-GPU configuration.

Hourglass can converge the model to an accuracy of 69.08% with $L = 1$, while the accuracies are only 58.44%, 48.47%, and 43.58% with $L = 2, 3, 4$. This experiment tells us that when training models with Hourglass, the setting of $L = 1$ delivers the best performance in both model convergence and model accuracy. From the clients' perspective, the setting of $L = 1$ can minimize the energy consumption caused by model training on their edge devices. This is a critical benefit when clients' edge devices are energy-constrained.

7 Related Work

Federated Learning. FL is an ML scheme designed to address the issue of privacy preservation with cloud-centric ML schemes [41]. It allows multiple clients to collaboratively train a shared global ML model under the coordination of a central parameter server without uploading their private data to the cloud for model training. A series of studies have encompassed diverse aspects of FL, such as methods for optimizing model convergence [32, 42], techniques to reduce communication overhead [47, 49], defense mechanisms to combat poisoning attacks [16, 31], and methods for preserving clients' data privacy [64, 77], among others.

Split Federated Learning. In conventional FL schemes, clients train the entire ML models on their own data. However, the sizes of ML models have increased rapidly in recent years, clients started to struggle with training the entire models independently due to resource constraints such as limited memory and computational power. To tackle these challenges, SplitFed [67] has been proposed. SplitFed partitions the ML model for training into two parts: a client-side model partition and a server-side model partition. Clients only need to train their client-side model partitions on their own data and transmit the intermediate features produced to the fed server. The fed server then trains the server-side model partitions with corresponding clients' intermediate features. This split FL scheme offloads the majority of the training load to the fed server, significantly reducing the computational burden on the clients and allowing them to train large-sized ML models in an FL system. Acknowledging the benefits of split FL, many researchers attempt to tackle its limitations and improve its performance. For example, Han et al. [19] introduced alternative local loss functions specifically designed for split learning, enabling client-side model partitions to be updated without receiving backpropagated signals from the fed server, thus improving latency and communication efficiency.

However, split FL raises two main issues with the fed server. 1) Computational Overhead. In a conventional split FL system, the fed server needs a GPU to maintain and train each client's server-side model partition. The fed server may not have the luxury of having sufficient GPUs at its disposal, especially when it is running on a server such as an edge server [2, 62]. When the fed server has to switch clients' server-side model partitions in and out of GPUs, the computation overhead can slow down model training by as much as 4x. 2) Storage Overhead. Conventional FL systems have to store all clients' server-side model partitions. The storage overhead incurred can easily become a massive burden on the fed server when the size of the ML model for training and/or the number of clients in the system increases. These issues were thoroughly analyzed in Section 2 and Hourglass is the first attempt to address them systematically.

8 Conclusion

This paper presented Hourglass, a new split federated learning (FL) system to mitigate the computational overhead and storage overhead that hinder the application of split FL in practice. Instead of maintaining a server-side model partition for each client, Hourglass maintains server-side model partitions based on the number of GPUs available to eliminate model-switching overhead and reduce the storage capacity needed to facilitate split FL for clients. With a new Dissimilar Feature First (DFF) strategy, Hourglass manages to achieve top performance in model convergence and model accuracy in both single-GPU and multi-GPU configurations. Hourglass employs Locality-Sensitive Hashing (LSH) to cluster clients' intermediate features, enabling asynchronous forward passes for different clients and allowing it to accommodate practical scenarios where stragglers can easily slow down the entire training process. Compared to the state-of-the-art split FL system, Hourglass accelerates model convergence by up to 35.2x, and improves model accuracy by up to 9.28%. In the future, we will study model heterogeneity in split FL and its impact on Hourglass.

Acknowledgments

This research was supported in part by the National Key R&D Program of China under Grant No. 2023YFB4502400 and the Australian Research Council Discovery Project under Grant No. DP200102491.

References

[1] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).

[2] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 119–135.

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *Proceedings of machine learning and systems* 1 (2019), 374–388.

[4] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiaozhu Lin, and Mengwei Xu. 2023. Efficient federated learning for modern nlp. In *29th Annual International Conference on Mobile Computing and Networking*. 1–16.

[5] Debora Caldarola, Massimiliano Mancini, Fabio Galasso, Marco Ciccone, Emanuele Rodolà, and Barbara Caputo. 2021. Cluster-driven graph federated learning over multiple domains. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2749–2758.

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. 2021. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9650–9660.

[7] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *International conference for high performance computing, networking, storage and analysis*. 1–16.

[8] Hyunsung Cho, Akhil Mathur, and Fahim Kawsar. 2022. Flame: Federated learning across multi-device environments. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 3 (2022), 1–29.

[9] Giannis Daras, Nikita Kitaev, Augustus Odena, and Alexandros G Dimakis. 2020. Smyrf-efficient attention using asymmetric clustering. *Advances in Neural Information Processing Systems* 33 (2020), 6476–6489.

[10] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. 2018. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505* (2018).

[11] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1073–1081.

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[13] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[14] Hugging Face. 2024. *Quantization*. https://huggingface.co/docs/optimum/en/concept_guides/quantization

[15] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16091–16101.

[16] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local model poisoning attacks to {byzantine-robust} federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*. 1605–1622.

[17] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. 2020. Discrete model compression with resource constraint for deep neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1899–1908.

[18] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 19586–19597.

[19] Dong-Jun Han, Hasnain Irshad Bhatti, Jungmoon Lee, and Jaekyun Moon. 2021. Accelerating federated learning with split learning on locally generated losses. In *ICML 2021 workshop on federated learning for user privacy and data confidentiality. ICML Board*.

[20] Dong-Jun Han, Do-Yeon Kim, Minseok Choi, Christopher G Brinton, and Jaekyun Moon. 2023. SplitGP: Achieving both generalization and personalization in federated learning. In *2023 IEEE Conference on Computer Communications (INFOCOM 23)*. IEEE, 1–10.

[21] Pengchao Han, Chao Huang, Geng Tian, Ming Tang, and Xin Liu. 2024. Convergence Analysis of Split Federated Learning on Heterogeneous Data. *arXiv preprint arXiv:2402.15166* (2024).

[22] Matan Haroush, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2020. The knowledge within: Methods for data-free model compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8494–8502.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[24] Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. 2022. Chex: Channel exploration for cnn model compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12287–12298.

[25] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. SwapAdvisor: Pushing deep learning beyond the GPU memory limit via smart swapping. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 20)*. 1341–1355.

[26] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2752–2761.

[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[28] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems* 4 (2022), 814–832.

[29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[30] Pranav Jeevan and Amit Sethi. 2021. Vision Xformers: Efficient attention for image classification. *arXiv preprint arXiv:2107.02239* (2021).

[31] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*. PMLR, 5201–5212.

[32] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*. PMLR, 5132–5143.

[33] Animesh Koratana, Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Lit: Learned intermediate representation training for model compression. In *2019 International Conference on Machine Learning*. PMLR, 3509–3518.

[34] Alex Krizhevsky and Geoffrey Hinton. 2009. CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[35] Shrinu Kushagra, Hemant Saxena, Ihab F Ilyas, and Shai Ben-David. 2019. A semi-supervised framework of clustering selection for deduplication. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 208–219.

[36] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.

[37] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. 2022. PyramidFL: A fine-grained client selection framework for efficient federated learning. In *28th Annual International Conference on Mobile Computing And Networking*. 158–171.

[38] Jingtao Li, Adnan Siraj Rakin, Xing Chen, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. 2022. ResSFL: A resistance transfer framework for defending model inversion attack in split federated learning. In *39th IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 22)*. 10194–10202.

[39] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2022. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 965–978.

[40] Songze Li, Duanyi Yao, and Jin Liu. 2023. Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models. In *International Conference on Machine Learning*. PMLR, 20296–20311.

[41] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[42] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 429–450.

[43] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).

[44] Yipeng Li and Xinchen Lyu. 2024. Convergence analysis of sequential federated learning on heterogeneous data. *Advances in Neural Information Processing Systems* 36 (2024).

[45] Yunming Liao, Yang Xu, Hongli Xu, Lun Wang, Zhiwei Yao, and Chunming Qiao. 2024. MergeSFL: Split federated learning with feature merging and batch size regulation. In *40th IEEE International Conference on Data Engineering (ICDE 24)*. IEEE, 2054–2067.

[46] Jiachen Liu, Fan Lai, Yinwei Dai, Aditya Akella, Harsha V Madhyastha, and Mosharaf Chowdhury. 2023. Auxo: Efficient federated learning via scalable client clustering. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 125–141.

[47] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2020. Client-edge-cloud hierarchical federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[48] Zongqing Lu, Swati Rallapalli, Kevin Chan, Shiliang Pu, and Thomas La Porta. 2019. Augur: Modeling the resource requirements of ConvNets on mobile devices. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 352–365.

[49] Bing Luo, Xiang Li, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. 2021. Cost-effective federated learning design. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10.

[50] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. 2018. ThiNet: Pruning CNN filters for a thinner net. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 10 (2018), 2525–2538.

[51] Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. 2021. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems* 34 (2021), 5972–5984.

[52] Jie Ma, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2023. Structured federated learning through clustered additive modeling. *Advances in Neural Information Processing Systems* 36 (2023), 43097–43107.

[53] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[54] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning*. PMLR, 7197–7206.

[55] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2023. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193* (2023).

[56] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 54–66.

[57] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2023. Gemel: Model merging for memory-efficient, real-time video analytics at the edge. In *20th USENIX Symposium on Networked Systems Design and Implementation*. 973–994.

[58] Paperswithcode. 2012. Image classification on CIFAR-10. https://paperswithcode.com/sota/image-classification-on-cifar-10

[59] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. 2021. Unleashing the tiger: Inference attacks on split learning. In *2021 ACM SIGSAC Conference on Computer and Communications Security*. 2113–2129.

[60] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. In *International Conference on Learning Representations*.

[61] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2020. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems* 32, 8 (2020), 3710–3722.

[62] Sudipta Saha Shubha and Haiying Shen. 2023. AdaInf: Data drift adaptive scheduling for accurate and SLO-guaranteed multiple-model inference serving at edge servers. In *ACM SIGCOMM 2023 Conference*. 473–485.

[63] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[64] Mengkai Song, Zhibo Wang, Zhifei Zhang, Yang Song, Qian Wang, Ju Ren, and Hairong Qi. 2020. Analyzing user-level privacy attack against federated learning. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2430–2444.

[65] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*. PMLR, 10096–10106.

[66] Xiu Tang, Sai Wu, Gang Chen, Jinyang Gao, Wei Cao, and Zhifei Pang. 2021. A learning to tune framework for LSH. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2201–2206.

[67] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. 2022. SplitFed: When federated learning meets split learning. In *AAAI Conference on Artificial Intelligence*, Vol. 36. 8485–8493.

[68] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing*

systems 34 (2021), 24261–24272.

[69] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*. PMLR, 10347–10357.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[71] Irene Wang, Prashant Nair, and Divya Mahajan. 2024. Fluid: Mitigating stragglers in federated learning using invariant dropout. *Advances in Neural Information Processing Systems* (2024), 13–36.

[72] Kaibin Wang, Qiang He, Feifei Chen, Chunyang Chen, Faliang Huang, Hai Jin, and Yun Yang. 2023. FlexiFed: Personalized federated learning for edge clients with heterogeneous model architectures. In *ACM Web Conference 2023*. 2979–2990.

[73] Kaibin Wang, Qiang He, Feifei Chen, Hai Jin, and Yun Yang. 2023. FedEdge: Accelerating Edge-Assisted Federated Learning. In *Proceedings of the ACM Web Conference 2023*. 2895–2904.

[74] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. *Advances in Neural Information Processing Systems* 31 (2018).

[75] Zhiyuan Wang, Hongli Xu, Jianchun Liu, He Huang, Chunming Qiao, and Yangming Zhao. 2021. Resource-efficient federated learning with hierarchical aggregation in edge computing. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.

[76] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[77] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3454–3469.

[78] Wen Wu, Mushu Li, Kaige Qu, Conghao Zhou, Xuemin Shen, Weihua Zhuang, Xu Li, and Weisen Shi. 2023. Split learning over wireless networks: Parallel design and resource management. *IEEE Journal on Selected Areas in Communications* 41, 4 (2023), 1051–1066.

[79] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. 2024. FwdLLM: Efficient federated finetuning of large language models with perturbed inferences. In *2024 USENIX Annual Technical Conference (ATC 24)*. 579–596.

[80] Xuyun Zhang, Wanchun Dou, Qiang He, Rui Zhou, Christopher Leckie, Ramamohanarao Kotagiri, and Zoran Salcic. 2017. LSHiForest: A generic framework for fast tree isolation based ensemble anomaly analysis. In *33rd International Conference on Data Engineering*. IEEE, 983–994.

[81] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* 28 (2015).

[82] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. FLDetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2545–2555.

[83] Quan Zhou, Haiquan Wang, Xiaoyan Yu, Cheng Li, Youhui Bai, Feng Yan, and Yinlong Xu. 2023. MPress: Democratizing billion-scale model training on multi-GPU servers via memory-saving inter-operator parallelism. In *IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 556–569.

[84] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. 2018. Discrimination-aware channel pruning for deep neural networks. *Advances in Neural Information Processing Systems* (2018), 31–43.