TimelyFL: Heterogeneity-aware Asynchronous Federated Learning with Adaptive Partial Training

Anonymous authors
Paper under double-blind review

Abstract

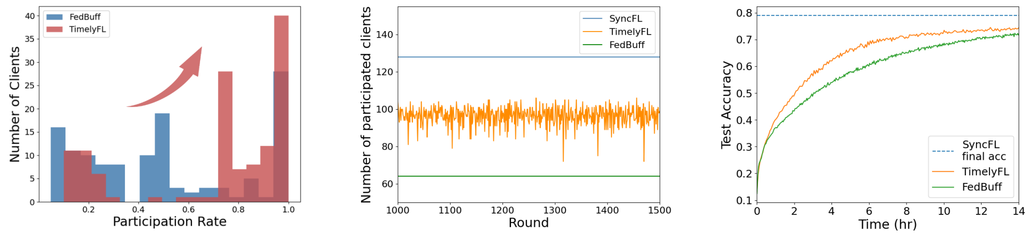
In cross-device Federated Learning (FL) environments, scaling synchronous FL methods is challenging as stragglers hinder the training process. Moreover, the availability of each client to join the training is highly variable over time due to system heterogeneities and intermittent connectivity. Recent asynchronous FL methods (e.g., `FedBuff` Nguyen et al. (2021)) have been proposed to overcome these issues by allowing slower users to continue their work on local training based on stale models and to contribute to aggregation when ready. However, we show empirically that this method can lead to a substantial drop in training accuracy as well as a slower convergence rate. The primary reason is that fast-speed devices contribute to many more rounds of aggregation while others join more intermittently or not at all, and with stale model updates. To overcome this barrier, we propose `TimelyFL`, a heterogeneity-aware asynchronous FL framework with adaptive partial training. During the training, `TimelyFL` adjusts the local training workload based on the real-time resource capabilities of each client, aiming to allow more available clients to join in the global update without staleness. We demonstrate the performance benefits of `TimelyFL` by conducting extensive experiments on various datasets (e.g., CIFAR-10, Google Speech, and Reddit) and models (e.g., ResNet20, VGG11, and ALBERT). We also validate the feasibility of `TimelyFL` by deploying it on an Android-based mobile device testbed. In comparison with the state-of-the-art (i.e., `FedBuff`), our evaluations reveal that `TimelyFL` improves participation rate by 21.13%, harvests 1.28× - 2.89× more efficiency on convergence rate, and provides a 6.25% increment on test accuracy.

1 Introduction

Federated learning (FL) has emerged as a promising distributed ML paradigm that preserves privacy. The gist of FL is to keep the clients' private data on the devices and perform local model training for each client. A central server will collect these locally trained models to update a global model and then push it back for the next round of training.

Most existing FL protocols are based on synchronous FL training (SyncFL), meaning that at each round all clients (or a selected cohort of clients) are updating their local models based on the latest update broadcast by the server at the beginning of that round. Due to the unbalanced communication or hardware capabilities and non-identical training data distribution, however, the time consumption for a local update can vary substantially from device to device, and some clients may even be temporarily disconnected during the training process. Thus, leaving the server with two suboptimal choices: to wait for *all* clients participating in each round to finish their local training and contribute to model aggregation (which will cause significant delays due to stragglers), or to only wait for a *subset* of the faster clients (which will ignore all the work and contributions from slower clients). These critical challenges largely impede the scalability of SyncFL and make it difficult to land in large-scale cross-device scenarios.

To address those challenges, recent works have proposed asynchronous federated learning (AsyncFL) Huba et al. (2021); Nguyen et al. (2021); Wu et al. (2021); Avdiukhin & Kasiviswanathan (2021), which allows slower clients to continue local training and contribute to future aggregation rounds.



(a) Participation rate distribution across all devices

(b) Number of participated clients during FL training rounds

(c) Time-to-accuracy performance for each strategy

Figure 1: Empirical performance of SyncFL, `FedBuff`, and `TimelyFL` in CIFAR-10 classification task with FedOpt as server aggregator (for experiment details and other evaluations see Section 4). `TimelyFL` includes more devices join in global update during the training (shown in (a) and (b)). As more devices participate timely, `TimelyFL` harvests both convergence rate and accuracy boost compared to `FedBuff` (shown in (c)).

AsyncFL *decouples* client local training from global model aggregation/updates, as only certain clients would simultaneously get an update from the cloud server, which decreases the impact of stragglers. The most recent AsyncFL work – `FedBuff` Nguyen et al. (2021); Huba et al. (2021) – proposes that the server should perform a gradient aggregation to create a global model once the number of received local updates reaches a requisite threshold, which is a tunable parameter referred as *aggregation goal*. The slower clients can still upload their updates later as long as they finish local training, but their updates may not be included based on staleness information.

As highlighted in Figure 1c, we empirically demonstrate that while `FedBuff` achieves much faster convergence to a certain intermediate accuracy, it can, unfortunately, lead to a substantial drop in final accuracy compared to SyncFL. The intuitive explanation is that, as `FedBuff` only accepts a *fixed* number of local updates to contribute to the global model in every communication round, it decreases the parallel computing efficiency by blocking other completed local updates into global aggregation, which turns them into stale updates as they would be postponed to the next round of global update. Moreover, the server aggregator prefers the fast-speed devices contributing more rounds of training, whereas low-speed devices may not enjoy the same frequency of contribution. Even when the slow devices participate in global training, they occasionally send the staled updates that potentially harm the convergence of the global loss. As shown in Figure 1a and Figure 1b, compared to SyncFL, `FedBuff` only includes a fixed number of local updates per round, and achieves a low participation rate (i.e., the number of aggregation participated divided by the total number of aggregation rounds) on average with a biased distribution, indicating that the inclusiveness of a group has been diminished, serving as the main reason for the test accuracy gap.

To close the gap between SyncFL and AsyncFL, we propose `TimelyFL`, a heterogeneity-aware asynchronous federated learning with adaptive partial training. One key difference from the previous AsyncFL works is that `TimelyFL` does not set a fixed number limit to the number of local updates for the global aggregation per round. Instead, to accommodate a *flexible* number of clients joining in the global update, we set a wall-clock time limit for every aggregation interval. The aggregation interval equals the kth fastest local update time among all clients, where k is a tunable parameter. As long as the device can deliver its model update to the server within this interval, it will be part of the global aggregation. To include more available devices to join in global aggregation without staleness, we introduce *partial model training* for clients with low capacity. Instead of fully training a model, only a part of the model that composes of a subset of consecutive output-side layers will be assigned to them for backward pass training. With partial model training, both local computation time and communication time will decrease for stale clients.

As illustrated in Figure 2, our method unifies the local training time by adaptively adjusting the workload (i.e., the local epoch number and partial training ratio) for each client, which makes it feasible for clients to finish the local training and upload the updates to the server within the calculated aggregation interval in every communication round. As such, `TimelyFL` tackles the system heterogeneity issue and eliminates the staleness of local update reports for slower devices.

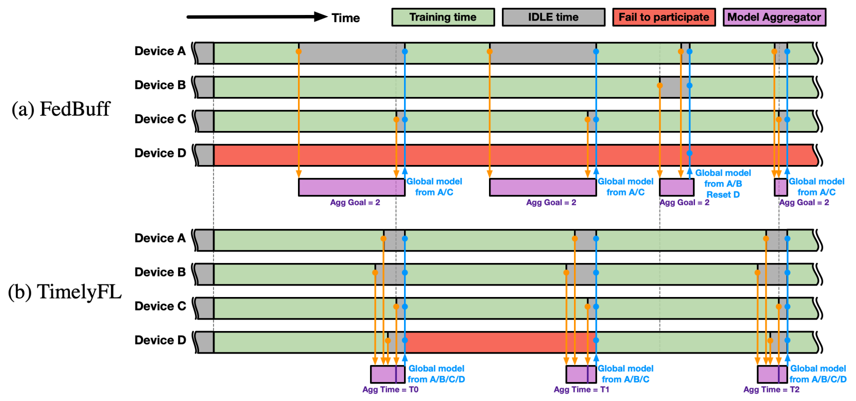


Figure 2: `FedBuff` (top): Server updates the global model as it receives the requisite number of local updates, and slower devices still could send their updates at a later time to the server. Fast devices participate more times in the global update, while slow devices contribute less or no participation. `TimelyFL` (bottom): Server updates the individual workload every round based on the real-time availability of each client to include more devices in global update timely, largely increases the participation rate for slow devices.

We evaluate the performance of `TimelyFL` by both the FL simulation and real deployment on Android devices. The experiments are designed across various application tasks, including image classification, speech recognition, and natural language processing on CIFAR-10 Krizhevsky (2009), Google Speech Command Warden (2018), and Reddit Comment red benchmark datasets, respectively, with two commonly used aggregation functions, FedAvg McMahan et al. (2017) and FedOpt Reddi et al. (2021). Our results show that 66.4% of devices increase the participation rate and the average participation rate increases by 21.1% in `TimelyFL` compared to `FedBuff`. Under the same scale of the FL system, `TimelyFL` outperforms `FedBuff` Nguyen et al. (2021) on both time-to-accuracy and final test accuracy, as shown in Figure 1c.

2 Related Work

Asynchronous Federated Learning. Due to the intermittent connectivity and availability among clients, asynchronous federated learning is a promising solution to take the system heterogeneity of devices into account Zhang et al. (2021). Most asynchronous FL works concentrate on solving the straggler problem, such as Li et al. (2021), Wu et al. (2021), and Xie et al. (2019). `PAPAYA` Huba et al. (2021) and `FedBuff` Nguyen et al. (2021) have been proposed to mitigate stragglers and enable secure aggregation jointly. Specifically, the individual updates are not incorporated by the server as soon they arrive. Instead, the server will keep receiving local updates in a secure buffer, where the buffer size is a tunable parameter, and update the global model whenever the buffer is full. The slow devices can also send the local update to the server after the global aggregation. Their update will be considered for the next available global update. However, practically speaking, fast devices participate in global updates many more times than slow devices, and some slow devices cannot join in the global aggregation even once due to the staleness control.

All of the above approaches assume that the client should process the local training within the full-model size. As the slower users participate in the global aggregation, they can only contribute with stale updates. Some previous works have pointed out that the effects of the stale update on distributed machine learning can directly harm the overall convergence behavior for the final model, aligned with the asynchronous distributed optimization theory suggested by Zhou et al. (2018); Dai et al. (2019); Giladi et al. (2020). Moreover, the participation rate is mainly unbalanced due to the high-speed devices contributing more rounds to global updates compared to the slow-speed devices. In contrast to previous approaches, we focus on enabling all clients to join in the global aggregation effectively based on their local resources to improve the inclusiveness of the final global model training.

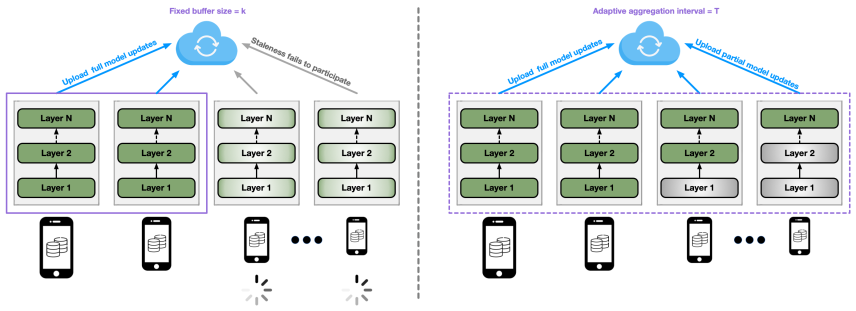

Fixed buffer size = k
Upload full model updates
Staleness fails to participate
Layer N
Layer 2
Layer 1
Adaptive aggregation interval = T
Upload full model updates
Upload partial model updates

Figure 3: **Left**: The standard AsyncFL framework. The server will initiate the global update once it collects the requisite number of local updates. The other clients will be postponed to a latter communication round with stale update information. **Right**: The proposed `TimelyFL`. The server will include all the received local updates within aggregation interval to global update. Clients with a weaker capacity are assigned to train a subset of the model to catch the aggregation interval time.

Partial Model Training. Partial model training can be viewed as an efficient approach to reduce both communication and computation workload on the client-side of the FL system. `FedPrune` Munir et al. (2021) proposes a method that prunes the global model for each client based on their device capabilities, where slow clients are served smaller models and faster clients train on larger models. `FedPT` Sidahmed et al. (2021) leverages the partially trainable neural networks on clients to reduce communication costs and enable faster training with a smaller memory footprint and with few implications on model performance. Other works such as Yang et al. (2021) and Ro et al. (2022) also address that partial model training can save both communication cost and memory usage productively in cross-device federated learning. All of the above works maintain the partial ratio for the sub-model of a certain client as constant during the entire FL training process, which neglects that the availability of each device is not stable throughout the time. In this work, we adaptively adjust the partial ratio for the local model training based on the real-time device availability, which aims to improve both efficiency and utility for each client.

3 Our Method

3.1 Standard Asynchronous Federated Learning

Figure 3 (left) illustrates the standard asynchronous FL framework. Instead of waiting for all clients to finish the local model training, the server stores the individual updates from clients in a buffer and then adjusts the global model once the buffer size reaches the requisite number of the aggregation goal. Other non-participating devices will postpone their contribution to global updates in the latter communication round once they finish the training. Given that the standard AsyncFL framework suffers from inclusiveness constraints described in the introduction section, we propose an efficient AsyncFL framework as shown in Figure 3 (right) to address this issue.

3.2 TimelyFL Design

3.2.1 Preliminaries

To increase the participation rate for the slow clients, we aim to design a cross-device asynchronous federated learning framework where each device can finish the local update within a limited time interval by adaptively adjusting its workload per round. Here, the workload is defined as the product of local training epoch number E and partial model training ratio α. To formalize this, our time utility function, which considers the local training optimization at the client side, is designed as follows:

$$\underset{E,\alpha}{\arg\max} \quad (\tilde{t}_{cmp,c} \times E \times \alpha + \tilde{t}_{com,c} \times \alpha) \leq T_k \tag{1}$$

where $\tilde{t}_{cmp,c}$ is the estimated local computation time, and $\tilde{t}_{com,c}$ is the estimated local communication time of the client c for one epoch of full model training in a certain communication round calculated by the server. Note that both $\tilde{t}_{cmp,c}$ and $\tilde{t}_{com,c}$ are not constant throughout the training due to the nature of mobile devices. By adjusting E and α, each device is supposed to participate in the global aggregation every communication round timely and maximally utilize its resource capability within aggregation interval T_k. Therefore, the overall distributed optimization involves more iterations on diverse datasets, leading to faster convergence.

3.2.2 Adaptive Model Training

Due to resource limitations, some weak devices may not finish the full model training effectively within the time interval T_k, making them become stale clients in the system and impeding them from contributing to the global model. To address this issue, we introduce partial model training to slow devices. Figure 3 (right) illustrates our approach when partial model training lands on the device heterogeneity FL system. Instead of a full training model, weak devices will be assigned to train partial models composed of a subset of consecutive output-side layers. During the training, only the subset of assigned layers will operate both forward pass and backward propagation, while the other layers will only process the forward pass for the input data but be frozen during weight updates. After local training finishes, the partially-trained clients only return the update for the assigned layers to the server for the global aggregation, as the frozen layers are unchanged during training.

We target to solve two bottlenecks in the cross-device FL with partial model training, communication and computation efficiency for the stale clients. In edge computing, the backward propagation consumes much more time than the forward pass. Partial model training would essentially reduce the training time, as it does not need to calculate gradients for the frozen parameters. The reduced time is roughly proportional to the reduced model size, as we empirically shown in the Appendix A.2.2. Moreover, we only send the trainable part of updates to the server, substantially improving communication efficiency, especially when stragglers with limited network connections exist. By implementing partial model training, we aim to let low-capacity devices report their local updates to the server timely without staleness, thereby improving their participation rate during FL training.

3.2.3 TimelyFL Algorithm

Based on the adaptive model training, we propose the `TimelyFL`. `TimelyFL` tries to unify each client's round time to the limited aggregation interval T_k by adaptively adjusting the workload concerning its real-time availability per communication round. The workload is defined as the product of the partial training ratio α and the local epoch number E. `TimelyFL` framework is composed of three main parts, `TimelyFL` server, local time update, and workload scheduling.

Algorithm 1: TimelyFL.

```
Input: $k$: the aggregation participation target, $n$: the number of training concurrency
1  for $r \in \{0, \cdots, R-1\}$ communication rounds do
2      Global server do:
3          Sample $n$ clients for $\mathcal{S}$ uniformly at random and send $W_s^r$ to clients in $\mathcal{S}$ ;
4      Clients $c \in \mathcal{S}$ in parallel do:
5          $\tilde{t}_{total}, \tilde{t}_{cmp}, \tilde{t}_{com}$ = LocalTimeUpdate($M$) ;
6      Global server do:
7          $T_k^r \leftarrow$ the $k$th smallest number in $\langle \tilde{t}_{total} \rangle$ ;
8          $\langle E^r \rangle, \langle \alpha^r \rangle, \langle t_{rpt}^r \rangle$ = WorkloadScheduling($T_k{}^r, \langle \tilde{t}_{cmp} \rangle, \langle \tilde{t}_{com} \rangle$) ;
9      Clients $c \in \mathcal{S}$ in parallel do:
10         $W_c^r \leftarrow$ adaptive model training ;
11     Global server do:
12         $W_s^{r+1} \leftarrow$ aggregate $\langle W_c^r \rangle$ ;
13 end
Output: $W_s^R$
```

`TimelyFL` Server. `TimelyFL` server is in charge of adjusting the aggregation interval T_k, local training epoch E, and partial training ratio α for each device during the FL training, as summarized in Algorithm 1. The aggregation interval T_k in each round equals the kth smallest value among $\langle \tilde{t}_{total} \rangle$, as the estimated unit total time for all clients. At each communication round, `TimelyFL` server randomly samples n clients to construct the collection $\mathcal{S}$ and distributes the global model to the clients inside $\mathcal{S}$, which means n clients would start the local training in this round, same as the definition of training concurrency in the `FedBuff`. Each selected client would perform one data batch full model training to estimate its time consumption and report it to the server. Then, aggregation interval time T_k and training hyperparameters for client c (i.e., local training epoch number E and partial training ratio α) would be adjusted based on all selected clients' status during the FL training process. The server would also return a local computation budget time $t_{rpt,c}$, as the wall-clock time when the client must report its training status.

Local Time Update. To efficiently accommodate the capabilities, each participant needs to update its time consumption to the server as summarized in Algorithm 2. Specifically, each client would collect the real computation time t_{cmp} from one data batch full model training. The unit computation time $\tilde{t}_{cmp}$ is estimated by t_{cmp} and progress β, where β is defined as the ratio of trained batch number to the total data batch number. The local communication time equals the model's file size M over the device's real-time network bandwidth Bw, as the same setting in the previous FL system work Lai et al. (2021a).

Algorithm 2: Local Time Update.

```
Input: $M$: the file size of the received global model, $Bw$: the real-time network bandwidth
1 $t_{cmp}, \beta \leftarrow$ one data batch training ;
2 $\tilde{t}_{com} = M/Bw$ ;
3 $\tilde{t}_{cmp} = t_{cmp}/\beta$ ;
4 $\tilde{t}_{total} = \tilde{t}_{cmp} + \tilde{t}_{com}$ ;
Output: $\tilde{t}_{total}, \tilde{t}_{com}, \tilde{t}_{cmp}$
```

Workload Scheduling. `TimelyFL` would adjust the local epoch number E and partial training ratio α for each client in every communication round based on the estimated $\tilde{t}_{com,c}, \tilde{t}_{cmp,c}$ and aggregation interval T_k, as the relationship shown in 1. If one's unit total time is smaller than T_k, then the server would try to maximize its local training utility and minimize the idle time, as to assign more than one local epoch training for the next round. Otherwise, the server would assign less amount of workload to them by decreasing the model training ratio α, which guarantees they can finish at least one local epoch training within the report time $t_{rpt,c}$ and catch up the global aggregation timely. We summarized the scheduler as Algorithm 3.

Algorithm 3: Workload Scheduling.

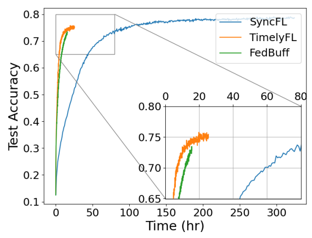
```
Input: $T_k$: aggregation interval time, $\langle\tilde{t}_{cmp}\rangle$: unit computation time, $\langle\tilde{t}_{com}\rangle$: unit
       communication time
1 for each client $c \in \mathcal{S}$ in parallel do
2     $E_c = \max(\lfloor(T_k - \tilde{t}_{com,c})/\tilde{t}_{cmp,c}\rfloor, 1)$ ;
3     $\alpha_c = \min(T_k/(\tilde{t}_{com,c} + \tilde{t}_{cmp,c}), 1)$ ;
4     $t_{rpt,c} = T_k - \tilde{t}_{com,c} \times \alpha_c$ ;
5 end
Output: $\langle E\rangle, \langle\alpha\rangle, \langle t_{rpt}\rangle$
```

4 Experiment

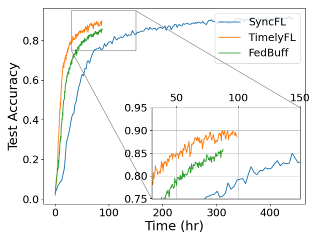
4.1 Experimental Settings

Datasets, Models, and Tasks. To demonstrate `TimelyFL`'s effectiveness across tasks, we evaluate `TimelyFL` on three benchmark datasets from various categories of FL applications:

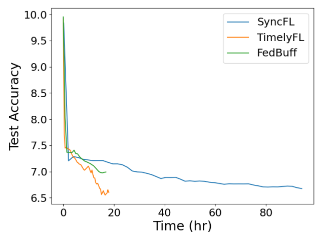
1. **Image Classification.** The CIFAR-10 dataset Krizhevsky (2009) consists of 60,000 colour images in 10 classes. There are 50,000 training images and 10,000 test images. To follow the realistic non-iid data in FL scenarios, we partition both datasets into 128 clusters using a



(a) CIFAR-10 with FedOpt



(b) Google Command with FedOpt



(c) Reddit with FedOpt

Figure 4: Time-to-accuracy performance for SyncFL, `FedBuff` and `TimelyFL`.

Dirichlet distribution with α equals 0.1. We evaluate the dataset with ResNet-20 He et al. (2016) model.

2. **Speech Recognition.** The Google Command speech dataset Warden (2018) covers 105,829 audio recordings collected from 2,618 clients. The training set includes recordings from 2,112 speakers, the validation set includes 256 speakers, and the test set includes 250 speakers. The data set is composed of 35 common words from the everyday vocabulary, such as "Yes", "No", "Up", and "Down". We evaluate the dataset with VGG11 Simonyan & Zisserman (2015) model for a 35-class keyword spotting task. We also evaluate the dataset with a lightweight model based on one related work Zhang et al. (2022), and the detailed results for the lightweight model and data-preprocessing methods are presented in the Appendix A.3.
3. **Natural Language Processing.** Reddit red consists of comments from 1,660,820 users in the Reddit forum. In this dataset, we filter the users with less than 20 words in total and restrict to the 30k most frequently used words, as the same settings in the previous work Lai et al. (2021a). Then, we train the lightweight Albert Lan et al. (2020) model for the next-word-prediction task. The performance is evaluated by the perplexity loss (ppl), which lower is better.

Experiment Setup. We implemented two computing paradigms of the `TimelyFL` system: (1) Android-based mobile edge training and (2) CPU/GPU-based distributed training. For the deployment on the edge device, we built up a testbed with 32 Andriod phones, which is composed of 10 makes from 7 different brands. The detail of the configuration for the mobile devices can be found in Appendix A.1.1. We use the FedML platform He et al. (2020), an open-source framework for FL, to execute our framework. On the CPU/GPU training side, to approach the real-world heterogeneous client system performance in emulation, we acquire the local computation times of deep learning models across hundreds of device types from the AI benchmark Ignatov et al. (2019) and communication times from Network Measurements on mobiles Huang et al. (2011). These data will be assigned to the simulated devices we create in the experiment, the same as the settings in previous FL works Lai et al. (2021a;b); Li et al.. The distribution of heterogeneous system utility across simulated clients will be shown in the Appendix A.1.3.

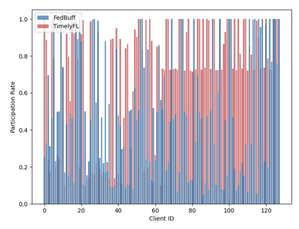
Evaluation Metrics and Baselines. We compare `TimelyFL` with `FedBuff` Nguyen et al. (2021); Huba et al. (2021) as the AsyncFL baseline. To demonstrate applicability of `TimelyFL`, we present the evaluation results using two aggregation function, FedAvg McMahan et al. (2017) and FedOpt Reddi et al. (2021). We evaluated the performance of `TimelyFL` and its baseline using the following three metrics: *test accuracy/loss*, *time-to-accuracy*, and *participation rate*. The participation rate is defined as the total number of rounds that the device contributes to the global update divided by the total communication round number. The rate is distributed in the interval between 0 and 1, which implies how often a client participates in the global model update.

4.2 End-to-End Performance

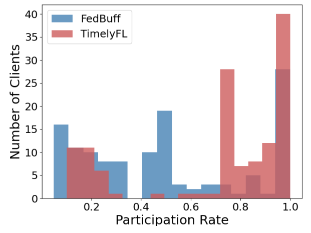
We begin by comparing the end-to-end performance of `TimelyFL` on benchmark datasets, conducting on the CPU/GPU-based training. The training concurrency is set to 128 for CIFAR-10 related experiments, 20 for Google speech related experiments, and 100 for Reddit related experiments.

Table 1: Wall clock training time to reach target validation accuracy on benchmark datasets (lower is better). "> 200 hr" indicates the target accuracy was not reached.

Dataset	Agg. function	Accuracy/Loss	TimelyFL	FedBuff	SyncFL
CIFAR-10	FedAvg	60%	5.50 ±2.5% hr	7.86 ±2.1% hr (1.43×)	76.81 ±2.4% hr (13.96×)
		70%	12.81 ±1.8% hr	> 200 hr	150.98 ±1.7% hr (11.78×)
	FedOpt	60%	3.58 ±2.5% hr	5.68 ±2.6% hr (1.59×)	34.87 ±2.3% hr (9.74×)
		70%	6.46 ±1.8% hr	18.73 ±2.3% hr (2.89×)	58.84 ±0.8% hr (9.11×)
Google Speech	FedAvg	70%	22.90 ±2.1% hr	42.71 ±2.3% hr (1.87×)	103.07 ±2.1% hr (4.50×)
		80%	40.54 ±1.2% hr	70.60 ±2.0% hr (1.74×)	187.93 ±1.2% hr (4.64×)
	FedOpt	70%	18.08 ±1.1% hr	30.60 ±1.7% hr (1.69×)	66.13 ±1.2% hr (3.66×)
		80%	31.39 ±0.9% hr	53.36 ±0.9% hr (1.70×)	107.38 ±0.7% hr (3.42×)
Reddit	FedAvg	7.0 (ppl)	9.56 ±3.1% hr	15.82 ±2.9% hr (1.65×)	23.36 ±1.5% hr (2.44×)
		6.8 (ppl)	17.99 ±0.7% hr	> 200 hr	67.32 ±0.5% hr (3.74×)
	FedOpt	7.0 (ppl)	10.99 ±2.7% hr	14.09 ±2.8% hr (1.28×)	27.25 ±2.1% hr (2.48×)
		6.8 (ppl)	12.86 ±0.6% hr	> 200 hr	57.65 ±0.4% hr (4.48×)



(a) Participation rate for each client



(b) Participation rate distribution

Figure 5: Participation rate evaluation.

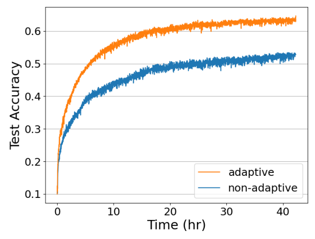


Figure 6: TimelyFL performance under adaptive and non-adaptive workload schedule.

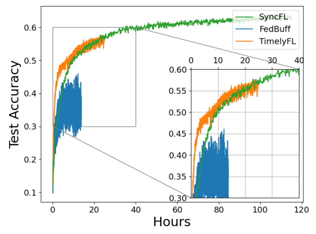
The communication round is set to be 2000, 1000, and 500 for CIFAR-10, Google speech, and Reddit, respectively. For both FedBuff and TimelyFL, we set the aggregation goal and aggregation participation target equal to 50% of training concurrency for a fair comparison. We run each experiment five times with different random seeds and report its mean and standard deviation for the time consumption in the Table 1.

Speedup of TimelyFL. Given the same heterogeneous data, TimelyFL achieves the shortest training time to reach all target accuracy/loss. Table 1 shows the training time needed to converge to the target accuracy/loss for each strategy considered. Compared to TimelyFL, synchronous FL requires 3.42 - 13.96× more times, and FedBuff needs 1.28 - 2.89× in terms of wall clock time. Besides of the time-to-accuracy speedup, TimelyFL also harvests test accuracy increment compared to FedBuff within the same communication rounds. As the learning curve in the Figure 4, TimelyFL achieves 3.27% and 4.01% higher final accuracy on CIFAR-10 and Google Command, and 0.43 lower ppl on Reddit in comparison to FedBuff with FedOpt. Under FedAvg, TimelyFL achieves 4.93% and 6.25% higher final accuracy on CIFAR-10 and Google Command, respectively, and 0.20 lower ppl on Reddit compared to FedBuff.

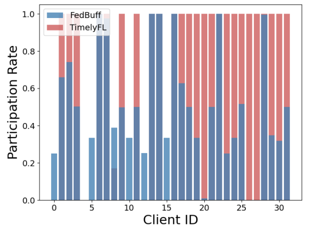
4.3 Understanding the Advantages of TimelyFL

TimelyFL improves inclusiveness. [1] In Table 1, we view the SyncFL as the standard baseline that does not include any asynchronous technique and FedBuff as the baseline that only introduces a fixed buffer size to accelerate the training. Instead of fixed buffer size, TimelyFL adopts a flexible buffer size controlled by aggregation interval time, which allows more available clients to participate in the global update per round. As illustrated in Figure 5, 66.4% of devices are able to achieve an increased participation rate, and the average participation rate per client increases by 21% in TimelyFL compared to FedBuff under the CIFAR-10 experiment setting we implemented in the

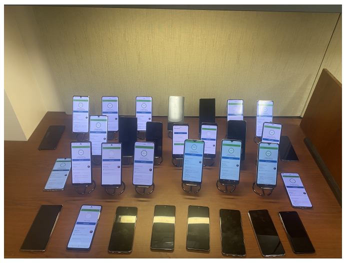
[1] In this paper, the inclusiveness increment represents the participation rate increment in the FL training.



(a) Time-to-accuracy performance



(b) Participation rate for each device



(c) Android-based FL testbed

Figure 8: Empirical performance of real-world deployment for SyncFL, `FedBuff`, and `TimelyFL`.

last section. The average participation rate increment is the main reason for the time-to-accuracy speed-up. As each client joins the global model update more rapidly, the learning efficiency increases during the FL training. In addition, combined with more devices contributing to the global model more frequently, `TimelyFL` improves inclusiveness during the model training compared to `FedBuff`.

The contribution of inclusiveness for model performance is especially significant when training on the non-iid dataset, where each client brings a unique local update to the global model. To demonstrate our point, we test both `TimelyFL` and `Fedbuff` with FedAvg as an aggregator on the CIFAR-10 dataset using a non-iid partition. As shown in Figure 7, as the parameter for Dirichlet distribution goes up, the convergence-time gap between `TimelyFL` and `Fedbuff` increases as well, which demonstrates our advantage for non-iid data training compared to `Fedbuff`.

`TimelyFL` brings heterogeneity-awareness. Under cross-device federated learning, most participating entities have limited computing capability and intermittent connectivity. As such, it could not be guaranteed that devices would complete their training workload in every communication round as assigned initially. To effectively resist the disturbance, the training hyperparameters, such as the partial training ratio and local epoch number, should be adaptively scheduled based on the real-time capability of each device. To demonstrate our strategy, we test the training under the `TimelyFL` framework both with and without adaptive workload scheduling on the CIFAR-10 dataset, with the training concurrency equals to 64. Figure 6 shows the learning curves for both scenarios. With adaptive workload scheduling, `TimelyFL` saves 4.09× convergence time to 50% accuracy and 10.89% test accuracy increment, which illustrates that real-time workload scheduling essentially improves both learning efficiency and accuracy.

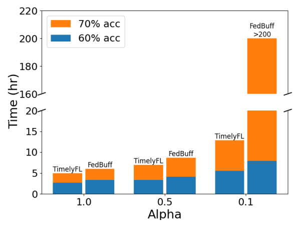


Figure 7: Time-to-accuracy performance under different non-iid distribution.

4.4 Real-world Deployment

To evaluate the feasibility of real-world deployment, we implement `TimelyFL` on the real Android-based FL testbed with 32 mobile devices, as shown in Figure 8c. The experiments are under FedAvg as an aggregator function on CIFAR-10 dataset for 1000 communication rounds. The dataset is partitioned into 32 clusters using Dirichlet distribution with parameters 0.1. The aggregation goal and aggregation participation target are set to be 16, and training concurrency is 32 for both FL frameworks. As shown in Figure 8a, `TimelyFL` achieves a higher convergence speed compared with the other two strategies before reaching 50% test accuracy, which confirms the simulation results elaborated in Section 4.2. `FedBuff` takes longer to reach the same test accuracy than SyncFL. One reasonable argument is the system heterogeneity difference between the testbed and the device profiles drawn from the benchmark. In the FL testbed, the computation speed of the high-end device is around five times faster than the low-end one, while the benchmark used for simulation has a much larger variety of computation speeds among all devices. As shown in Figure 8b, compare to `FedBuff`, 56.3% of devices increases their participation rate to the upper bound. As fewer clients join in global aggregation per round for `TimelyFL`, the convergence speed gradually approaches that of SyncFL towards the end of the training.

5 Conclusion

In this work, we propose `TimelyFL`, a heterogeneity-aware asynchronous FL scheme with adaptive partial training. To include more available devices joining in global aggregation timely, `TimelyFL` introduces partial model training to the slow-speed devices. Moreover, `TimelyFL` are resilient to system heterogeneity by adjusting the local training workload based on the real-time resource capabilities of each client during FL training. We empirically demonstrate that `TimelyFL` could outperform major AsyncFL proposals in terms of both time-to-accuracy and test accuracy. Our implementation on Andriod-based mobile phones demonstrates the feasibility of a real-world deployment. We leave extending convergence analysis as future work.

References

Reddit Comment Data. `https://files.pushshift.io/reddit/comments/`.

Dmitrii Avdiukhin and Shiva Prasad Kasiviswanathan. Federated learning under arbitrary communication patterns. In *ICML*, 2021.

Wei Dai, Yi Zhou, Nanqing Dong, H. Zhang, and Eric P. Xing. Toward understanding the impact of staleness in distributed machine learning. *ArXiv*, abs/1810.03264, 2019.

Niv Giladi, Mor Shpigel Nacson, Elad Hoffer, and Daniel Soudry. At stability's edge: How to adjust hyperparameters to preserve minima selection in asynchronous training of neural networks? *ArXiv*, abs/1909.12340, 2020.

Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Han Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *ArXiv*, abs/2007.13518, 2020.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Junxian Huang, Cheng Chen, Yutong Pei, Zhaoguang Wang, Zhiyun Qian, Feng Qian, Birjodh Tiwana, Qiang Xu, Z Mao, Ming Zhang, et al. Mobiperf: Mobile network measurement system. *Technical Report. University of Michigan and Microsoft Research*, 2011.

Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Michael G. Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. Papaya: Practical, private, and scalable federated learning. *ArXiv*, abs/2111.04877, 2021.

Andrey D. Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3617–3635, 2019.

Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yuezhi Cai, Tianhang Yu, Chengfei Lv, and Zhihua Wu. Mnn: A universal and efficient inference engine. *ArXiv*, abs/2002.12418, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Fan Lai, Yinwei Dai, Xiangfeng Zhu, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning. *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, 2021a.

Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *OSDI*, 2021b.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942, 2020.

Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. Pyramidfl: A fine-grained client selection framework for efficient federated learning.

Xingyu Li, Zhe Qu, Bo Tang, and Zhuo Lu. Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients. *ArXiv*, abs/2102.06329, 2021.

H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Muhammad Tahir Munir, Muhammad Mustansar Saeed, Mahad Farah Ali, Zafar Ayyub Qazi, and Ihsan Ayyub Qazi. Fedprune: Towards inclusive federated learning. *ArXiv*, abs/2110.14205, 2021.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael G. Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *ArXiv*, abs/2106.06639, 2021.

Sashank J. Reddi, Zachary B. Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konecný, Sanjiv Kumar, and H. B. McMahan. Adaptive federated optimization. *ArXiv*, abs/2003.00295, 2021.

Jae Hun Ro, Theresa Breiner, Lara McConnaughey, Mingqing Chen, Ananda Theertha Suresh, Shankar Kumar, and Rajiv Mathews. Scaling language model size in cross-device federated learning. *ArXiv*, abs/2204.09715, 2022.

Hakim Sidahmed, Zheng Xu, Ankush Garg, Yuan Cao, and Mingqing Chen. Efficient and private federated learning with partially trainable networks. *ArXiv*, abs/2110.03450, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *ArXiv*, abs/1804.03209, 2018.

Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A. Jarvis. Safa: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70:655–668, 2021.

Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *ArXiv*, abs/1903.03934, 2019.

Tien-Ju Yang, Dhruv Guliani, Franccoise Beaufays, and Giovanni Motta. Partial variable training for efficient on-device federated learning. *ArXiv*, abs/2110.05607, 2021.

Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and Salman Avestimehr. Federated learning for internet of things: Applications, challenges, and opportunities. *arXiv preprint arXiv:2111.07494*, 2021.

Tuo Zhang, Tiantian Feng, Samiul Alam, Sunwoo Lee, Mi Zhang, Shrikanth S. Narayanan, and Salman Avestimehr. Fedaudio: A federated learning benchmark for audio tasks. 2022.

Zhengyuan Zhou, P. Mertikopoulos, Nicholas Bambos, Peter W. Glynn, Yinyu Ye, Li-Jia Li, and Li Fei-Fei. Distributed asynchronous optimization with unbounded delays: How slow can you go? In *ICML*, 2018.