FedBalancer: Data and Pace Control for Efficient Federated Learning on Heterogeneous Clients

Jaemin Shin
School of Computing
KAIST
Daejeon, Republic of Korea
jaemin.shin@kaist.ac.kr

Yunxin Liu
Institute for AI Industry Research (AIR)
Tsinghua University
Beijing, China
liuyunxin@air.tsinghua.edu.cn

Yuanchun Li∗
Institute for AI Industry Research (AIR)
Tsinghua University
Beijing, China
liuyanchun@air.tsinghua.edu.cn

Sung-Ju Lee
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
profsj@kaist.ac.kr

ABSTRACT

Federated Learning (FL) trains a machine learning model on distributed clients without exposing individual data. Unlike centralized training that is usually based on carefully-organized data, FL deals with on-device data that are often unfiltered and imbalanced. As a result, conventional FL training protocol that treats all data equally leads to a waste of local computational resources and slows down the global learning process. To this end, we propose FedBalancer, a systematic FL framework that actively selects clients’ training samples. Our sample selection strategy prioritizes more “informative” data while respecting privacy and computational capabilities of clients. To better utilize the sample selection to speed up global training, we further introduce an adaptive deadline control scheme that predicts the optimal deadline for each round with varying client training data. Compared with existing FL algorithms with deadline configuration methods, our evaluation on five datasets from three different domains shows that FedBalancer improves the time-to-accuracy performance by 1.20–4.48× while improving the model accuracy by 1.1–5.0%. We also show that FedBalancer is readily applicable to other FL approaches by demonstrating that FedBalancer improves the convergence speed and accuracy when operating jointly with three different FL algorithms.

CCS CONCEPTS

• Human-centered computing → Ubiquitous and mobile computing;
• Computing methodologies → Machine learning.

KEYWORDS

federated learning, heterogeneity, sample selection, deadline control

∗corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiSys ’22, June 25–July 1, 2022, Portland, OR, USA
© 2022 Copyright held by the owner/authors. Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9185-6/22/06.
https://doi.org/10.1145/3498361.3538917

1 INTRODUCTION

Federated learning (FL) is a machine learning paradigm that performs decentralized training of models on mobile devices (e.g., smartphones) with locally stored data [33]. FL trains on a large corpus of private user data without collecting them, with only the model weight updates communicated externally from the user’s device [11]. With FL, researchers have proposed to improve AI in diverse domains: human mobility prediction [24], RF localization [16], traffic sign classification [6], tumour detection [32, 47], and Clinical Decision Support (CDS) model for COVID-19 [18]. FL also has product deployments as large companies such as Google or Taobao deploy language processing and item recommendation tasks across millions of real-world devices [56, 77].

One of the key objectives in FL is to optimize time-to-accuracy performance, which is a wall clock time for a model to achieve the target accuracy [38]. Achieving high time-to-accuracy performance is important as FL consumes significant computation and network resources on edge-user devices [21]. For model developers who prototype a mobile AI with FL without a proxy dataset, achieving faster convergence on thousands to millions of devices is desired to efficiently test multiple model architectures and hyperparameters [33]. Service providers who frequently update a model with continual learning with FL require to minimize the user overhead with better time-to-accuracy performance [39].

In realistic FL scenarios, however, heterogeneous data being distributed over isolated users becomes the main challenge in achieving high time-to-accuracy performance [44, 79]. While data engineering techniques such as importance sampling [5, 34, 52, 61] are widely adopted in centralized learning to optimize the training process, applying them in FL is infeasible as it requires private user data sharing. For this reason, previous FL algorithms [38, 45, 53] mostly treat every client data equally, which could result in a waste...
of computational resources and slow convergence. We conducted a preliminary study to examine this phenomenon and found that the ratio of informative samples is reduced from 93.2% to 20% as the FL progresses. This could further exacerbate with heterogeneous hardware of FL clients, as low-end devices might fail to send model updates while training a large portion of unimportant samples.

To address this problem, we propose \textit{FedBalancer}, a systematic FL framework with sample selection. The sample selection of \textit{FedBalancer} prioritizes more “informative” samples of clients to efficiently utilize their computational effort. This allows low-end devices to contribute to the global training within the round deadline by focusing on smaller but more important training samples. To achieve high time-to-accuracy performance, the sample selection is designed to operate without additional forward or backward pass for sample utility measurement at FL rounds. Lastly, \textit{FedBalancer} can coexist and collaborate with orthogonal FL approaches to further improve performance.

To realize \textit{FedBalancer}, we addressed the following challenges: (1) Simply reducing the training data of a client with random sampling could lead to lower model accuracy as the statistical utility of the training data would decrease. As such, \textit{FedBalancer} selects samples based on their statistical utility measurement. (2) Collecting sample-level statistical utility for sample selection might break the privacy guarantee of FL. To address this problem, we propose client-server coordination to maintain \textit{loss threshold}, which allows clients to effectively select important samples while only exposing differentially-private statistics of their data. (3) The sample selection itself might not directly lead to time-to-accuracy performance improvement when the FL round deadline remains fixed. To formulate the benefit of selecting different deadlines, we propose a metric \textit{deadline efficiency} (DDE) that calculates the number of round-completed clients per time. This allows \textit{FedBalancer} to predict the optimal deadline with varying client training data.

We implemented \textit{FedBalancer} and conducted experiments on five datasets from three domains that contain real-world user data. Compared with existing FL aggregation algorithms with deadline configuration methods, \textit{FedBalancer} improves the time-to-accuracy performance by 1.20–4.48×. \textit{FedBalancer} achieves this improvement without sacrificing model accuracy; in fact, it improves the accuracy by 1.1%–5.0%. In addition, we implement \textit{FedBalancer} on top of three orthogonal FL algorithms to demonstrate that \textit{FedBalancer} is easily applicable to other FL approaches and improves their time-to-accuracy performance.

We summarize our contributions as follows:

- We propose a systematic framework for FL with sample selection, which actively selects high utility samples at each round without collecting privacy-invasive sample-level information from clients.
- We propose a deadline control strategy for each round of FL based on the newly defined metric \textit{deadline efficiency}, which optimizes the time-to-accuracy performance along with our sample selection.
- We implement \textit{FedBalancer} jointly with existing FL algorithms, showing improvement in both time-to-accuracy and model accuracy.

2 BACKGROUND AND MOTIVATION

2.1 Federated Learning

Federated Learning (FL) operates across multiple mobile devices to globally train a model from the distributed client data. FedAvg [53], the most commonly used FL approach [30], operates as follows: (i) Suppose there are \(N\) clients in an FL system. For each round of FL, the server randomly selects \(K\) clients (\(K \ll N\)) who participate in model training. (ii) At the \(R\)-th round, the server transmits the current model weights \(w_R\) to the selected clients. (iii) Each client then performs model training for \(E\) epochs with their local data and generates \(w^{k}_{R+1}\), where \(k\) denotes the client index. (iv) Clients upload the updated model parameters to the server, and (v) the server aggregates different clients’ updates and generate the updated model as \(w^{R+1} = \frac{\sum_{k=1}^{K} n_k w^{k}_{R+1}}{\sum_{k=1}^{K} n_k}\), where \(n_k\) indicates the number of data points of client \(k\) and \(n\) the number of all data points.

A key objective in FL is to optimize time-to-accuracy performance. FL tasks typically require hundreds to thousands of rounds to converge [13, 38], and clients participating at a round undergo substantial computational and network overhead [21]. Deploying FL across thousands to millions of devices should be done efficiently, quickly reaching the model convergence while not sacrificing the model accuracy. This becomes more important when FL has to be done multiple times, as often the case when model developers prototype a new model with FL without a proxy dataset or periodically update a deployed model to new domain via continual learning or online learning with FL.

However, the heterogeneities of the real-world mobile clients make it challenging to achieve good time-to-accuracy performance. Data heterogeneity: The client-generated training data are imbalanced and not independent and identically distributed (non-IID) due to each user’s different mobile device usage or physical and mental characteristics [53, 73]. While the training data in centralized learning could be filtered and organized with data engineering techniques [5, 32, 42] to address data heterogeneity, applying the same solution is infeasible in FL as it is dealing with distributed private data on clients. Such data heterogeneity of FL clients results in slow convergence and suboptimal performance [44, 45, 79].

Hardware heterogeneity: The client devices have distinct computational capabilities and network connectivity, resulting in up to 12× more round completion time between different clients [43]. Thus, waiting for every client to complete its task at a round might significantly delay the training process. A common practice for such an issue is to set a \textit{deadline} for a round duration and drop clients that fail to send the model updates before the deadline [4, 55]. However, this results in less contribution from clients with low-end devices, which could result in delayed convergence or biased model training [45, 75].

2.2 Motivational Study

To address the heterogeneity problems, various approaches have been proposed. Researchers proposed FL algorithms that allow clients to train different numbers of local epochs [45] or different subsets of model weights [20, 28] based on clients’ hardware capabilities. Model personalization has also been proposed to tackle data heterogeneity [23, 40, 41, 49, 50, 57, 58, 68]. In addition, client selection strategies for FL training have been proposed to optimize
the convergence speed on heterogeneous clients [14, 15, 38]. Although these techniques have improved the convergence speed or accuracy, they treat all data of clients equally, which could lead to a waste of computational resources for training non-important samples and result in suboptimal time-to-accuracy performance.

We investigate such limitation of previous FL approaches and discuss how FedBalancer should be designed based on our experiments that simulate FL on heterogeneous clients. We used widely used datasets to benchmark FL methods: FEMNIST [17] and Shakespeare [63]. We simulated heterogeneous training latency and network connectivity on the real-world clients as described in Section 4.1.

**Inefficiency of Full Data Training.** As in FedAvg [53], most FL approaches let clients to fully train their entire data at a training round. Other approaches that samples a subset of client training data for each round use equal weights on all data [23, 31, 38]. However, previous studies in centralized machine learning [5, 34, 52] found that the importance of all samples are not equal; A large portion of samples are learned quickly after few training rounds and could be ignored afterwards. Thus, we conducted an experiment to verify if the same phenomenon also applies in FL and how significant it is. We suspect that the inefficiency problem would be more serious in FL as limited computational resources of mobile clients could be wasted on non-important samples.

In this experiment, we measured the Gradient Norm (GN) of each sample on federated clients to evaluate sample-level contribution on a model update. For each training round, we collected and sorted the GN of each sample from the selected clients. We removed top 5% of samples to avoid evaluating noisy samples and applied min-max scaling to \([0.0, 1.0]\) interval. The experiment ran until the model converged. The results from each dataset are shown in Figure 1a and 1b. Each column of graphs indicates the sorted GN of samples from a training round, where the largest value is located at the bottom. The \(x\)-shaped points illustrate where the gradient norm with scaled value of 0.2 exists at each training round.

From both datasets, we observe that the GN of samples are mostly high at the early stage of FL, but only small portion of samples show high GN afterwards, meaning that the number of samples that contribute knowledge to the model are reducing as the training progresses. In FEMNIST dataset, only 6.8% of the samples from early training rounds (round index 0~150) have less scaled GN value than 0.2, but 80.0% of the samples are less than 0.2 for the last 150 training rounds. Similarly in the Shakespeare dataset, 6.7% of the samples from early training rounds (round index 0~25) have less scaled GN value than 0.2, but 54.8% of the samples are less than 0.2 for the last 25 training rounds. This result also implies that the samples are not equally important during these FL tasks and current FL approaches could spend large portion of training time for samples that have negligible contribution to the model update.

This experiment motivates FedBalancer to start training with all the samples at the beginning, and then gradually remove samples that the model has already learned. In Section 3.2, we further illustrate how FedBalancer selects a subset of samples of each client at training rounds based on our findings in this experiment.

**Importance of Deadline Selection.** While the existence of optimal deadline for achieving shortest training time in FL has been studied [75], controlling the deadline for high time-to-accuracy performance has been largely overlooked. To understand the performance of existing deadline configuration methods, we conducted an experiment on the two datasets with SmartPC [43] and four different fixed deadlines — \(0.5T\), \(1.0T\), \(1.5T\), and \(2.0T\) — where \(T\) indicates the mean of round completion time on every participating client. For SmartPC, we implemented a training round to last until 80% of the clients complete their task, where 80% is suggested by Li et al. [43].

Figure 2a and 2b illustrate the results on two datasets. Our takeaway from this experiment is twofold: (1) The deadline is a significant factor in achieving fast convergence speed and high accuracy, and (2) no single method achieved the best performance for both FL tasks. In FEMNIST dataset, deadline \(1.0T\) achieved the highest final accuracy (.815), while being 43.6%, 47.0% and 77.9% faster than SmartPC, deadline \(1.5T\), and deadline \(2.0T\), respectively, in achieving the test accuracy of .750. On the other hand, in Shakespeare dataset, SmartPC achieved the highest final accuracy (.488) while being 61.7%, 42.8%, and 45.4% faster than deadline \(1.0T\), \(1.5T\), and \(2.0T\), respectively, in achieving the test accuracy of .450. Deadline \(0.5T\) could not achieve high accuracy in either task, as most clients failed to upload their model update within the deadline.

In a training round of FL, the computation time has been shown to be the bottleneck [48, 70, 75]. As the amount of client computation changes with the sample selection of FedBalancer, finding an optimal deadline could be an important problem in achieving high time-to-accuracy performance. To this end, in Section 3.3, we propose how FedBalancer finds the optimal deadline for each training round for efficient FL on heterogeneous clients.
3 **FEDBALANCER**

### 3.1 Overview

For each round of FL, FedBalancer adaptively selects the client training data and controls the deadline to achieve high time-to-accuracy performance. We first provide an overview of how FedBalancer operates in an FL round and then describe how each component of FedBalancer is designed.

Figure 3 depicts the FedBalancer architecture during an FL round. The main functionality of FedBalancer is to actively control two variables for each round: *loss threshold* ($l_t$) and *deadline* ($d_{dl}$). The loss threshold works as a parameter that determines each client’s training data (Section 3.2) and the deadline determines the round termination time. The numbers inside a circle show the seven steps of an FL round with FedBalancer.

The server first transmits the current model weights $w_R$, the loss threshold $l_{t_R}$, and the deadline $d_{dl_R}$ ($R$ indicates the $R$-th round) to the selected clients of the round (Step 1). The *sample selection module* at each client selects the partial training data with the received loss threshold (Step 2) and trains the received model (Step 3). The client transmits the model update and the *metadata* collected from the sample selection and model training (Step 4), and the server aggregates these responses from all clients (Step 5). Based on the metadata from clients, the *loss threshold selection module* and the *deadline selection module* each selects the loss threshold $l_{t_{R+1}}$ and the deadline $d_{dl_{R+1}}$ for the next round (Steps 6 and 7).

**Challenges:** We aim to address the following challenges to realize FedBalancer:

1. **Sample selection without accuracy drop:** Simply reducing the training data with random sampling could result in degradation of model accuracy due to the decreased statistical utility. FedBalancer should thus prioritize samples based on their statistical utility.
2. **Privacy-preserving sample selection in FL:** While we aim to select an optimal set of client training data for each round, requiring up-to-date sample-level information from clients could harm the privacy guarantee of FL. FedBalancer should select client training samples without collecting privacy-invasive information from clients.
3. **Predicting optimal deadline with varying data:** As computation could be the bottleneck of an FL round [48, 71, 75], applying sample selection strategy of FedBalancer might greatly change the round completion time of clients. FedBalancer should adaptively predict the optimal deadline based on the sample selection status of heterogeneous clients for each round of FL.

### 3.2 Client Sample Selection

In Section 2.2, we observed that existing FL methods consume large portion of time to train samples that contribute only small gradient to the model. As these samples are quickly learned after few rounds, we design FedBalancer to start training with all samples and gradually remove already-learned samples. This enables FedBalancer to efficiently focus on more important samples at each round while optimizing the training process of FL. However, implementing such design in FL is non-trivial as the following question needs to be addressed: *How should FedBalancer distinguish between important and non-important samples at each stage of FL?*

A straightforward solution is to collect every sample-level importance information from all clients to a server at each round, and derive a criteria that determines more important samples to a current model. However, such approach is hardly applicable in FL as sharing information of every sample could break the privacy guarantee of FL and reveal the clients’ data. This approach also incurs significant network overhead in communicating all sample’s information at each round. An alternative approach is to have clients classify more important samples within their local data without exposing any information. However, as client data distributions are heterogeneous in FL [44, 53, 73, 79], clients could struggle to determine important samples without knowing the global data distribution.

To address this issue, we propose a client-server coordination to maintain a *loss threshold* variable, which enables clients to effectively select important samples without exposing private sample-level information. FedBalancer actively controls the loss threshold based on the collected metadata from clients with *loss threshold selection module*, where the metadata consists of differentially-private statistics of sample-level information.

Note that FedBalancer uses the *loss* of a sample to measure the statistical utility (and thus the importance) of a sample to the current model, similar to Importance Sampling [52, 61]. While other studies have also leveraged gradient norm or gradient norm upper bound [5, 34, 42] to achieve the same goal, we use loss as it is more widely applicable to FL tasks with non-gradient-based optimizations [59].

#### 3.2.1 Sample selection module

Algorithm 1 describes how the *sample selection module* of a client selects the samples at each round after it receives the loss threshold from the server. Let us assume that client $i$’s sample selection module is working at round $R$, with a given loss threshold $l_{t_R}$.

First, the module measures if a sample selection on a client is required for this round — i.e., it measures if a client is fast enough to...
Algorithm 1 Client $i$ at $R$-th round: Sample selection

1: func SelectSample($D^i, B^i, \text{loss}^i, ltr, dlr, w_r, p, E$)
2: $S \leftarrow \max \text{trainableSize}(\text{mean}(B^i), dlr, E)$
3: if $S \geq \text{len}(D^i)$ then
4: $D^i_R \leftarrow D^i$
5: else
6: $D^i_R, OT^i, UT^i \leftarrow \emptyset$
7: for $x \leftarrow 1$ to $\text{len}(D^i)$ do
8: if $\text{loss}^i[x] \geq ltr$ then $OT^i.insert(D^i[x])$
9: else $UT^i.insert(D^i[x])$
10: $L \leftarrow \max(S, \text{len}(OT^i))$
11: $D' \leftarrow \text{randSample}(OT^i, L \cdot p)$
12: $D'' \leftarrow \text{randSample}(UT^i, L \cdot (1 - p))$
13: $D^i_R \leftarrow \text{concatenate}(D', D'')$
14: return $D^i_R$

Algorithm 2 ltr selection for next $(R + 1)$-th round

1: func SelectLossThreshold($L_{\text{Low}_R}, L_{\text{High}_R}, ltr$)
2: $l \leftarrow \min(l_{\text{Low}_R})$
3: $l \leftarrow \text{mean}(L_{\text{High}_R})$
4: $ltr_{R+1} \leftarrow l + (ltr - l) \cdot ltr$
5: return $ltr_{R+1}$

The number of selected samples, $L$, is determined as the number of samples in $OT^i(\text{len}(OT^i))$. The loss threshold gradually increases (explained in Section 3.2.2), which allows clients to efficiently focus on samples with high statistical utility. However, if $S$ is larger than $\text{len}(OT^i)$, a client instead uses $S$ to maximize the statistical utility within the deadline (Line 10). As FedBalancer is built on top of Prox [45] that allows clients to train less number of epochs, clients with $S$ less than $\text{len}(OT^i)$ could still contribute to the model update.

3.2.2 Loss threshold selection module. The loss threshold selection module determines a loss threshold that effectively distinguishes the important and non-important samples. As the loss distribution of samples changes as FL proceeds, it is essential for the module to be knowledgeable of the current distribution. To respect privacy, the server collects few statistical information from the loss list of clients at the end of each round. Specifically, client $i$ at the $R$-th round provides $L_{\text{Low}_R}^i$ and $L_{\text{High}_R}^i$ values to the server, which indicate the low and high loss value of its current samples, respectively. We use the min loss value as $L_{\text{Low}_R}^i$ and use 80% percentile loss from the list as $L_{\text{High}_R}^i$ instead of the max value, as noisy samples could have abnormally high loss [66] and make FedBalancer to misjudge the loss distribution. As such values directly indicate a loss value of a specific sample, we apply Gaussian noise to the values to protect user privacy, as in differential privacy [38, 54, 72]. These values from clients get further aggregated on the server into a list as $L_{\text{Low}_R}$ and $L_{\text{High}_R}$. We report the performance of FedBalancer when different levels of noise is applied in Section 4.3.

With these metadata, the loss threshold selection module selects a loss threshold as described in Lines 1 - 5 in Algorithm 2. The module measures the loss low value ($l$) and loss high value ($l_h$) to estimate the current range of sample loss values of the clients (Lines 2 and 3). The module then outputs the loss threshold of the next round ($R + 1$) with variable loss threshold ratio ($ltr$), which calculates the linear interpolation between $l$ and $l_h$ (Line 4) as $ltr_{R+1} \leftarrow l + (l_h - l) \cdot ltr$. The loss threshold ratio ($ltr$) enables FedBalancer to start training with all samples and gradually remove already-learned samples. FedBalancer initialize $ltr$ as 0.0 and gradually increases the value by loss threshold step size ($iss$) as shown in Algorithm 3. Note that the deadline ratio ($dlr$), which controls the deadline of each round (described in Section 3.3), is also controlled with $ltr$.

To control $ltr$ and $dlr$, FedBalancer evaluates the benefit of the current configuration (loss threshold and deadline) at each round based on the statistical utility. For the $R$-th round, it is defined as $U_{R} \leftarrow \frac{L_{\text{Sum}_R}}{L_{\text{Sum}_R} + L_{\text{dlr}_R}}$, where $L_{\text{Sum}_R}$ is the loss sum of the selected samples, $L_{\text{dlr}_R}$ is the sum of the number of selected samples, and $dlr_{R}$ is the chosen deadline for the round. Note that $L_{\text{Sum}_R}$ and $L_{\text{dlr}_R}$ are calculated only from the clients who completed their task and succeeded

train its full dataset within the given deadline $dlr$. While FedBalancer makes clients focus on more important samples for efficient training, it allows clients to fully utilize its statistical utility if feasible. To this end, we calculate the maximum number of samples $S$ which client $i$ can train for $E$ epochs before the deadline, and verify if it is larger than the size of the client dataset $D^i$ (Line 2 - 4). As the computational ability of a client can change according to the device runtime conditions [76], FedBalancer collects the batch training latency of a client as $B^i$ during FL and uses the mean latency to estimate the max samples it can process. To calculate the mean latency from the first round, FedBalancer asks clients to sample the latency for $k$ times before FL begins. We used 10 for $k$ in our evaluation.

If the client $i$ is incapable of training its full dataset, the sample selection module determines which samples to train at the FL round by using the list of sample loss ($\text{loss}^i$). The loss list shows the statistical utility of all samples on the current model. While such loss list could be obtained by inferring all samples on up-to-date model at each round, it requires additional forward pass latency that could degrade the time-to-accuracy performance. Therefore, clients of FedBalancer perform whole-dataset forward pass only once when they are first selected at a round to generate a loss list. Then, whenever they train the subset of data, they update the loss values of selected samples that are obtained from the training procedure. We discuss the trade-off between latency reduction and obtaining up-to-date information in Section 5.

FedBalancer selects client samples based on the list of sample loss as follows: First, it divides client $i$’s samples into two groups: Under-Threshold ($UT^i$) and Over-Threshold ($OT^i$). Samples that have smaller loss than the loss threshold $ltr$ are put in $UT^i$ and otherwise in $OT^i$ (Line 7 - 9). We regard samples in $OT^i$ to be more important samples in training the current model and prioritize them in sample selection. We sample $L \cdot p$ samples from $OT^i$ and $L \cdot (1 - p)$ samples from $UT^i$ where $L$ indicates the number of selected samples and $p$ is a parameter in an interval of $[0.5, 1.0]$ (Line 10 - 12). The intuition of sampling a portion of data from $UT^i$ is to avoid catastrophic forgetting [35, 78] of the model on already-learned samples.
We explain how value by \( O_T \) Algorithm 3

\[ t \]

This is based on the formulation of statistical utility of state-of-the-art client selection method [38], which we only calculate the statistical utility from the data, applying them along with FL [14, 15, 38]. While these approaches prioritize clients with higher accuracy and low efficiency. On the other hand, configuring an extremely short deadline with a small \( t \) would result in almost no completed clients and low efficiency.

To understand how DDL-E is distributed at different deadlines at FL task, we profiled the DDL-E based on a large-scale smartphone dataset [75]. It contains the downlink and uplink network connectivity data and the model training latency data from real-world clients using heterogeneous hardware. We measured the DDL-E for deadlines in range of \([1sec, 1200sec]\) on two FL tasks: FEMNIST and Shakespeare.

Figure 4 presents the DDL-E measurements on two FL tasks. From both tasks, we observe that there exists a specific deadline that shows the peak DDL-E value. From the FEMNIST dataset, \( t = 172 \) seconds shows the max efficiency with DDL-E value of 2.57, while in the Shakespeare dataset, \( t = 157 \) seconds shows the max efficiency with DDL-E value of 0.13. The distribution shape of DDL-E values showing sharp peak around the max value implies that finding an optimal deadline for FL with FedBalancer could enable more completed clients and improved convergence speed.

We designed FedBalancer to select a deadline based on finding the best DDL-E values. Lines 1-17 of Algorithm 4 shows how FedBalancer finds the deadline with max DDL-E value. This is based on the clients’ hardware capability information: for client \( i \), they are DownLink speed (\( DL_i \)), UpLink speed (\( UL_i \)), and Batch training latency (\( B_i \)) (Line 1). When there are a total of \( N \) clients involved, we assume clients have already profiled \( \{DL_i, UL_i, B_i\}_{i=1}^N \) before FL and FedBalancer collects \( \{DL_i, UL_i, B_i\}_{i=1}^N \) during FL. FedBalancer first iterates over \( N \) clients to measure their completion time with the mean value of \( \{DL_i, UL_i, B_i\}_{i=1}^N \) (Lines 6-9). FedBalancer then measures the DDL-E from the smallest deadline \( t \) (we use \( t = 1sec \)) and increments \( t \) until all clients complete before \( t \) (Lines 10-16) and outputs the deadline with the max DDL-E value (Line 17).

As different subset of clients are selected for each round, FedBalancer finds the max DDL-E value among the selected clients of each round. Moreover, as clients use different size of training data with sample selection, we estimate the training time of client \( i \) (\( getTrainTime \) from Line 8) as follows:

\[
\text{len}(OT^i) - 1 \cdot \text{mean}(B^i) \cdot \text{numEpoch}. \\
\]

We use the length of \( OT^i \) in measuring the training time of a client to reflect the number of samples being selected. This allows FedBalancer to reliably find the deadline with the best DDL-E along with the sample selection strategy.
Algorithm 4 $ddl$ selection for next $(R + 1)$-th round

1: func FindPeakDDL-E($(DL^i, UL^i, B_i^N, \text{numEpoch})$
2:   completeTime ← 0
3:   $DDL$-E ← 0
4:   $c$ ← 0
5:   $t$ ← 1
6:   for $i$ ← 1 to $N$ do
7:     $T_{\text{network}}$ ← mean($DL^i$) + mean($UL^i$)
8:     $T_{\text{train}}$ ← getTrainTime(mean($B_i$), numEpoch)
9:     completeTime.insert($T_{\text{network}} + T_{\text{train}}$)
10: while $c$ ≠ $N$ do
11:     $c$ ← 0
12:     for $i$ ← 1 to $N$ do
13:       if $t$ ≥ completeTime($i$) then
14:         $c$ ← $c + 1$
15:         $DDL$-E.insert($\frac{c}{t}$)
16:     $t$ ← $t + 1$
17: return maxIndex($DDL$-E)
18: end func

3.3.2 Deadline selection module. The deadline selection module of FedBalancer determines the deadline that optimizes the training process and convergence speed. The module selects the deadline as shown in Lines 18 - 22 of Algorithm 2. The module measures deadline low value $dl$ and deadline high value $dh$, which are the deadlines with the max $DDL$-E value when clients are running 1 epoch and $E$ epochs, respectively (Lines 19 and 20).

The module then outputs the deadline of the next round $(R + 1)$ with parameter deadline ratio ($ddlr$) that calculates the linear interpolation between $dl$ and $dh$ (Line 21) as $ddlr_{t+1} = dl + (dh - dl) \cdot ddlr$. The reason of selecting a value between $dl$ and $dh$ is because FedBalancer is built on top of Prox [45] that allows clients to train various number of epochs within the deadline. With an aim to optimize the training efficiency, FedBalancer initially configures $ddlr$ as 1.0 and gradually decreases the value by the parameter $dss$, as explained in Section 3.2.2.

3.4 Collaboration with FL Methods

One advantage of FedBalancer is its applicability to orthogonal FL approaches that do not perform sample selection on clients or control the deadline. Applying FedBalancer could be achieved by simply adding the sample selection and deadline control strategies on top of other methods. We demonstrate the collaboration capability of FedBalancer with its implementation on top of three existing FL methods and improved performances in Section 4.5.

Some recent FL approaches, such as Oort [38], use one batch instead of the full dataset for training in each local epoch, making them non-trivial to be directly integrated with FedBalancer. To address this issue, we propose OortBalancer, which is built on top of Oort, where the sample selection strategy of FedBalancer is adopted with one adjustment: $L$ from Algorithm 1 is fixed to the batch size. Intuitively, while Oort trains one randomly selected batch for each local epoch, OortBalancer selects samples for one batch that focuses on more important samples and thereby optimizes the training process. We demonstrate the performance of OortBalancer in Section 4.5 as one of the three examples of FedBalancer collaboration.

4 EVALUATION

We evaluate FedBalancer to answer the following key questions: 1) How much performance improvement (in terms of time-to-accuracy and model accuracy) does FedBalancer achieve over existing FL methods? 2) How sensitive is FedBalancer with different choice of parameters? 3) How much performance improvement does each component of FedBalancer achieve? 4) How does FedBalancer perform when it jointly operates with orthogonal FL approaches?

4.1 Experimental Setup

Implementation. We developed FedBalancer on FLASH [75], a heterogeneity-aware benchmarking framework for FL based on LEAF [13]. FLASH provides a simulation of heterogeneous computational capabilities and network connectivity from a large-scale real-world trace dataset collected over 136k smartphones that span one thousand types of devices. We implement FedBalancer with the state-of-the-art FL aggregation method Prox [45]. Our implementation is based on Python 3.6 and TensorFlow 1.14 with 2,062 lines of code on top of FLASH. The source code of our FedBalancer implementation are available at https://github.com/jaeminshin/FedBalancer.

Datasets. To simulate FL tasks in our evaluation, we use five datasets that contain data generated by real-world users, which are categorized in three different domains as follows:

- Computer Vision (CV): For CV, we evaluated FedBalancer on two image recognition datasets: FEMNIST [17] and Celeba [51]. FEMNIST dataset contains images of handwritten digits and characters from 712 users with total 157,132 samples. Celeba dataset contains face attributes of 915 users with 19,923 samples. We use CNN models for both datasets as in previous work [75].
- Natural Language Processing (NLP): We evaluate FedBalancer on two NLP tasks each on different dataset: next-word prediction on Reddit [13] dataset and next-character prediction on Shakespeare [63] dataset. The Reddit dataset contains reddit posts from 813 users with 32,680 samples, and the Shakespeare dataset contains 845,231 samples separated into 171 users. We use LSTM models for both datasets as in previous work [75].
- Human Activity Recognition (HAR): We use UCI-HAR dataset [7] that contains six types of activity data on accelerometer and gyroscope from 30 users with 10,299 samples. As in previous work, we use CNN models [22].

Metrics. As in the previous work [38] that evaluated on heterogeneous FL clients, we mainly evaluate time-to-accuracy performance and final model accuracy on the experiments. Here, the time-to-accuracy performance indicates the wall clock time that is required for a model training task to reach an accuracy target. We repeat each experiment for three times with different random seeds and report the average and standard deviation of these evaluation metrics.
We do not test SPC with Prox as it is nontrivial to accept stragglers’ performing deadline configuration methods with Prox, which is identical with FedAvg when \( \mu = 0 \), which is used by most datasets. We test the sample selection method with the best performing deadline configuration methods with Prox, which is Prox+2T for Reddit dataset and Prof+17 otherwise.

| Method | We first ran FedAvg+1T on five datasets until convergence, with the number of rounds that are suggested by previous works [13, 22, 67, 75]: 1000, 100, 600, 40, and 50 rounds for FEMNIST, Celeba, Reddit, Shakespeare, and UCI-HAR. Based on the user trace data of FLASH, we measured the wall clock time which FedAvg+1T ran for each dataset, and ran experiments with other baselines and FedBalancer until the same wall clock time. Among the four FedAvg baselines, we pick the one with best accuracy for each dataset and configure it as the target accuracy of that task. Then, we measured the speedup of other methods in achieving the target accuracy and their final model accuracy achieved within the same wall clock time.

We tested the following combination of parameters for FedBalancer: \( w \) in \([5, 20]\), \( lss \) in \([0.01, 0.05, 0.10]\), \( dss \) in \([0.05, 0.10, 0.25]\), and \( p \) in \([0.75, 1.00]\). Among the results, we report the performance of FedBalancer with only one set of parameter: \( \{w, lss, dss, p\} = \{20, 0.05, 0.05, 1.00\} \). This is a set of parameters that we recommend FL developers to try with their task when they are not knowledgeable of which parameter performs the best. We used this parameter for all the experiments in Section 4. In Section 4.2, we also report the parameter set with the best final accuracy for each dataset as FedBalancer-A and the best speedup as FedBalancer-S to demonstrate the maximum performance FedBalancer could achieve.

Other configurations. As in previous study [75], we use batch size of 100 for Shakespeare and 10 for rest of the datasets. We select 100 clients at each round for datasets with more than 500 users, and otherwise select 10 users for Shakespeare and 5 users for UCI-HAR. We configured the clients to train five local epochs per round. We use learning rate of 0.001 for FEMNIST and Celeba, 2 for Reddit, 0.8 for Shakespeare, and 0.005 for UCI-HAR.

### 4.2 Speedup and Accuracy on Five FL Tasks

Table 1 shows the performance of FedBalancer on five datasets compared with the baseline methods, and Table 2 shows the parameters used for FedBalancer, FedBalancer-A, and FedBalancer-S for each dataset. We observed that FedBalancer shows improved time-to-accuracy performance over the baselines on every dataset: FedBalancer reaches the target accuracy 1.43\(\times\)1.57\(\times\) faster than the FedAvg-based baselines on CV datasets, 1.36\(\times\) to 1.58\(\times\) faster than

### Baselines

We use the following list of approaches as a baseline to compare with FedBalancer in our evaluation:

- **Aggregation algorithms**: We use FedAvg [53] and Prox [45], the most widely used aggregation algorithms for FL. Prox offers an optimizer with convergence guarantee on heterogeneous clients, while allowing clients to train various number of local epochs within the deadline. For the \( \mu \) parameter of Prox, we tested \( \mu \) in \([0, 0.01, 0.01, 0.1, 1.0]\) as suggested by the paper and pick one with the best final accuracy. All the datasets showed the best accuracy with 0.0 except Celeba with 0.1.

- **Deadline configuration methods**: We use four different deadline configuration methods in the evaluation. We configure two different fixed deadlines as a baseline, which are 17 and 2T. Before the training begins, we sample the round completion time of all participating clients and calculate the mean value as \( T \). Thus, 2T uses double of that mean value as a fixed deadline. We also adopt SmartPC (SPC) [43], and implemented it to involve the certain portion \( U_{required} \) of users to complete at a training round. As suggested in the paper, we use 80% for \( U_{required} \). Lastly, we use a method that waits every client to finish a round, which we named as WaitForAll (WFA).

- **Sample selection method**: We implement the baseline sample selection method that is a combination of the following: (1) We determined how many samples to select based on FedSS [12], which controls the training dataset size on clients with larger datasets for each training round. (2) There were several approaches that propose which samples to select based on loss [52, 61], gradient [5], or gradient norm upper bound [34, 42] of samples. As in FedBalancer, we use loss to select samples for the baseline experiment.

For baseline experiments, we use the combination of the aggregation algorithms and the deadline configuration methods of above. We do not test SPC with Prox as it is nontrivial to accept stragglers’ model update with less number of epochs when \( U_{required} \) users complete a training round. Moreover, we do not test WFA with Prox as it is identical with FedAvg when \( \mu = 0 \), which is used by most datasets. We test the sample selection method with the best performing deadline configuration methods with Prox, which is Prox+2T for Reddit dataset and Prof+17 otherwise.

### Method

| Table 1: Speedup and accuracy on five datasets with the real-world user data. |
|--------------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| **Task** | **Dataset** | **CV** | **Celeba** | **NLP** | **HAR** |
| **Methods** | **Speedup** | **Acc.** | **Speedup** | **Acc.** | **Speedup** | **Acc.** | **Speedup** | **Acc.** |
| FedAvg+1T | 1.00±0.00 | 0.796±0.007 | 0.97±0.05 | 0.851±0.005 | 0.08±0.12 | 0.909±0.001 | 0.24±0.22 | 0.399±0.020 | 0.49±0.36 | 0.814±0.029 |
| FedAvg+2T | 0.59±0.01 | 0.763±0.009 | 0.59±0.004 | 0.824±0.010 | 0.58±0.07 | 0.104±0.000 | 0.39±0.10 | 0.373±0.046 | 0.68±0.07 | 0.819±0.014 |
| FedAvg+SPC | 0.71±0.02 | 0.777±0.004 | 0.80±0.14 | 0.829±0.013 | 0.87±0.08 | 0.112±0.001 | 0.56±0.10 | 0.416±0.017 | 0.91±0.13 | 0.840±0.016 |
| FedAvg+WFA | 0.33±0.20 | 0.994±0.005 | 0.34±0.04 | 0.813±0.020 | 1.00±0.00 | 0.111±0.001 | 0.00±0.00 | 0.439±0.017 | 0.68±0.07 | 0.819±0.014 |
| Prox+1T | 0.99±0.02 | 0.795±0.008 | 1.05±0.03 | 0.855±0.002 | 2.87±0.43 | 1.21±0.005 | 1.14±0.16 | 0.47±0.003 | 0.96±0.09 | 0.849±0.008 |
| Prox+2T | 0.65±0.02 | 0.767±0.006 | 0.75±0.04 | 0.833±0.010 | 3.63±0.43 | 1.27±0.005 | 1.00±0.14 | 0.457±0.003 | 0.68±0.07 | 0.819±0.014 |
| SampleSelection | 1.01±0.01 | 0.799±0.006 | 1.03±0.06 | 0.852±0.002 | 1.52±0.38 | 1.18±0.001 | 0.85±0.24 | 0.439±0.021 | 0.90±0.05 | 0.845±0.008 |
| FedBalancer | 1.57±0.03 | 0.815±0.006 | 1.43±0.07 | 0.862±0.006 | 4.48±0.23 | 1.46±0.001 | 1.20±0.10 | 0.48±0.004 | 1.56±0.28 | 0.855±0.034 |
| FedBalancer-A | 1.60±0.06 | 0.820±0.003 | 1.52±0.07 | 0.873±0.004 | 4.08±0.76 | 1.54±0.000 | 1.31±0.28 | 0.505±0.004 | 1.39±0.43 | 0.893±0.008 |
| FedBalancer-S | 1.71±0.01 | 0.819±0.002 | 1.67±0.04 | 0.859±0.006 | 4.99±0.42 | 1.48±0.003 | 1.83±0.14 | 0.48±0.001 | 1.98±0.48 | 0.863±0.010 |

MobiSys ’22, June 25–July 1, 2022, Portland, OR, USA
Jaemin Shin, et al.
Table 2: \{w, lss, dss, p\} parameters from the FedBalancer experiments in Table 1. Parameters with forward slashes indicate the different parameters from the multiple runs of experiments with three different random seeds.

| Parameter | FedBalancer | FedBalancer-A | FedBalancer-S |
|-----------|-------------|---------------|--------------|
| w lss dss p | w lss dss p | w lss dss p | w lss dss p |
| FEMNIST | 20 0.01/0.01/0.01 | 0.10 | 1.00 | 20 0.05/0.05/0.05 | 0.25/0.25/0.25 | 1.00 | 1.00 |
| Celeba | 20/5/5  | 0.05/0.05/0.05 | 0.1/0.1/0.1 | 0.1/0.1/0.1 | 0.1/0.1/0.1 | 1.00/1.00/1.00 |
| Reddit | 20/20/20 | 0.01 | 0.1/0.1/0.1 | 0.1/0.1/0.1 | 0.1/0.1/0.1 | 0.1/0.1/0.1 |
| Shakespeare | 5 | 0.01/0.05/0.05 | 0.1/0.25/0.25 | 0.1/0.25/0.25 | 0.1/0.25/0.25 |
| UCI-HAR | 5 | 0.1/0.01/0.01 | 0.1/0.1/0.1 | 0.1/0.1/0.1 | 1.00/1.00/1.00 |

Table 3: Speedup and accuracy on different choice of FedBalancer parameters.

| Parameter | FedBalancer | FedBalancer-A | FedBalancer-S |
|-----------|-------------|---------------|--------------|
| w lss dss p | w lss dss p | w lss dss p | w lss dss p |
| FEMNIST | 1.52±0.15 | 1.50±0.14 | 1.54±0.07 | 1.41±0.15 | 1.50±0.17 |
| Reddit | 3.70±0.23 | 3.66±0.44 | 3.66±0.44 | 3.71±0.30 | 3.39±0.49 |

The Prox-based baselines, and 1.39−1.55× faster than the SampleSelection baseline. FedBalancer achieves 1.20−4.48× speedup and 1.05−1.23× speedup compared with the FedAvg and Prox-based methods respectively, while achieving 1.41−2.95× speedup over the SampleSelection baseline on NLP datasets. FedBalancer achieves speedup of 1.56X, 1.63X, and 1.73X compared with FedAvg, Prox, and SampleSelection baselines on a HAR dataset, respectively.

We noticed that FedBalancer consistently shows high time-to-accuracy performance on all datasets, while the performance of baselines was inconsistent across datasets. For example, among FedAvg-based baselines, FedAvg+1T shows the best time-to-accuracy performance on CV tasks but shows extremely low performance on NLP tasks. In contrast, FedAvg+WFA shows the best time-to-accuracy performance on NLP tasks among FedAvg-based baselines but shows low performance on CV tasks. In UCI-HAR, FedAvg+1T and FedAvg+SPC resulted in the best performance at different experiments with different random seeds. Prox+1T shows the best performance among baselines on Celeba, Shakespeare, and UCI-HAR, but shows worse performance than SampleSelection and Prox+2T on FEMNIST and Reddit respectively.

We observed that FedBalancer achieves this improvement of time-to-accuracy performance without sacrificing model accuracy; in terms of the final model accuracy, FedBalancer showed improvement over the baselines on all datasets. Compared to the FedAvg-based methods, FedBalancer achieved 1.1−5.0% accuracy improvement on different datasets. FedBalancer achieved 0.6−3.2% and 1.0−5.0% accuracy improvement over Prox-based methods and the SampleSelection baseline. FedBalancer-A, which marks the best accuracy of FedBalancer, also shows time-to-accuracy performance improvement over baselines at all datasets — showing further improvement in speedup on FEMNIST, Celeba, and Shakespeare. On the other hand, FedBalancer-S, which reports the best time-to-accuracy performance of FedBalancer, shows accuracy improvement over baselines at all datasets, with further improvement in accuracy on FEMNIST, Reddit, and UCI-HAR. This suggests that the performance of FedBalancer could be further improved with a carefully selected parameter for an FL task, while the fixed parameter set we recommend still shows improved performance.

4.3 Parameter Sensitivity Analysis

Table 3 shows the time-to-accuracy performance and final accuracy of FedBalancer on different choice of parameters \{w, lss, dss, p\}. For each type of parameter, we fixed it to a certain value and averaged the performance from experiments with different combination of other parameters. We used speedup compared to the best FedAvg-based baseline to measure the time-to-accuracy performance. We chose FEMNIST and Reddit to explore the effect of different parameters at different domains of FL tasks (CV and NLP).

For w, FedBalancer shows similar final accuracy (81.3% and 81.6%) performance on both of the candidate values on FEMNIST, but reports better time-to-accuracy performance (1.52x over 1.39x) with w = 20. On Reddit, w = 20 showed better speedup (3.71x over 3.42x) and accuracy (14.8% over 13.3%). In terms of lss, FedBalancer achieves faster training and higher accuracy when lss ≥ 0.05 on FEMNIST, but achieves better performance when lss ≤ 0.05 on Reddit. With dss, FedBalancer performs the better when dss ≤ 0.1 on both datasets, where both showed the best time-to-accuracy and accuracy with smaller dss = 0.05. Lastly, FedBalancer performs better with p = 1.0 on FEMNIST but reports better performance with p = 0.75 on Reddit. For the different trend of performance with lss and p parameters on two datasets, we suspect that Reddit requires more rounds with data sampled from full dataset in the early stage of training to achieve better performance, and big lss and p = 1.00 performs worse as it might quickly remove the low-loss samples from training. On the other hand, FEMNIST training performs better with high-loss samples from the early stage of training. This suggests that the best set of parameters could be selected based on how the training at an FL task proceeds.

Other than the algorithm parameter of FedBalancer, we study the effect of different level of noise on differential privacy that we applied to mask the metadata shared by the clients. Our implementation of differential privacy is based on the previous work [38],
adding the noise drawn from Gaussian distribution on the metadata, with the mean as zero and the standard deviation as Noise Factor (NF). Figure 5a shows the effect of different NFs on the performance of FedBalancer. We observe that the performance of FedBalancer degrades as the NF increases, as NF 0.0 achieves 81.9% accuracy but NF 0.5 and 5.0 each achieves 81.4% and 81.1% accuracy. However, NF 0.5 and 5.0 still achieves better time-to-accuracy performance and accuracy compared to Prox, while NF of 5.0 is considered to be very large noise [3]. This result implies that FedBalancer could achieve performance improvement over the baselines while applying the differential privacy.

4.4 Effect of FedBalancer Components

We conducted an experiment to understand the performance brought by each component of FedBalancer: Sample Selection (SS) and Deadline Control (DC). As FedBalancer is built on top of Prox [45], we add the components one by one to observe how the performance changes when each component is introduced.

Figure 6 reports the result of the experiment on FEMNIST and Reddit dataset. On FEMNIST dataset, we observe that the performance drops when SS is introduced, but gets further improved when DC is added. On Reddit dataset, however, the accuracy escalates as each component is added. The reason of performance degradation on FEMNIST with SS is due to the reduced statistical utility trained at each round with the same deadline. In contrast on Reddit dataset, the performance improved with SS as it allowed more clients to successfully send their model updates within the deadline with selected client data. Moreover, we suspect that SS has brought more performance improvement on Reddit than FEMNIST by effectively selecting more important samples and the clients that have such data. The result with DC on both dataset shows its effectiveness with SS in time-to-accuracy performance improvement.

### 4.5 Collaboration with FL Algorithms

To demonstrate the applicability of FedBalancer on orthogonal FL algorithms, we implement FedBalancer on top of three widely used FL approaches from different categories: **Oort** [38] as a client selection algorithm, q-FFL [46] as an aggregation algorithm, and **Structured Updates** [37] as a gradient compression algorithm. Figure 7 reports the experiment result, which we observe improvement in time-to-accuracy performance and model accuracy from all three cases. Collaboration with FedBalancer achieved 1.84×, 1.19×, and 1.13× speedup, while achieving 2.6%, 2.5%, 1.7% accuracy improvement on three algorithms respectively. These results suggest that FedBalancer could be implemented on top of various advanced FL algorithms to achieve further performance improvement.

### 4.6 Testbed Experiments with Android Clients

We further conducted experiments on Android clients to understand the effectiveness of FedBalancer on real hardware devices. We implemented the server using Flower [10], which is an open-source FL framework that communicates with clients via gRPC (Google Remote Procedure Call) and Protocol Buffers [1]. We used Ubuntu 18.04 server with Intel Xeon Gold 6254 Processor @ 3.10GHz and

---

Table 4: Android devices for the testbed experiments.

| Builder | Year | Device | Processor | Quantity |
|---------|------|--------|-----------|----------|
| Google  | 2016 | Pixel  | Snapdragon 821 | 2 |
|         | 2017 | Pixel  | Snapdragon 835 | 1 |
|         | 2017 | Pixel 2 | Snapdragon 835 | 2 |
|         | 2018 | Pixel 3 | Snapdragon 845 | 1 |
|         | 2020 | Pixel 5 | Snapdragon 765G | 1 |
| Samsung | 2016 | Galaxy S7 | Exynos 8890 | 1 |
|         | 2017 | Galaxy J7 | Exynos 7870 | 1 |
|         | 2019 | Galaxy Fold | Snapdragon 855 | 1 |
| Huawei  | 2015 | Nexus 6P | Snapdragon 810 | 3 |
|         | 2018 | P20 Lite | Kirin 659 | 1 |
| Motorola| 2014 | Nexus 6 | Snapdragon 805 | 2 |
| LG      | 2015 | Nexus 5X | Snapdragon 808 | 4 |
| Essential| 2017 | Essential Phone | Snapdragon 835 | 1 |

Table 5: Results from the testbed experiments.

| Method                  | Speedup | Accuracy |
|-------------------------|---------|----------|
| FedAvg+1T               | 0.99±0.03 | .852±.020 |
| FedAvg+2T               | 0.75±0.30 | .800±.054 |
| FedAvg+SPC              | 0.61±0.35 | .849±.013 |
| FedAvg+WFA              | 0.92±0.07 | .846±.007 |
| FedProx+1T              | 1.03±0.23 | .860±.007 |
| FedProx+2T              | 0.90±0.20 | .860±.013 |
| SampleSelection         | 0.99±0.13 | .846±.014 |
| **FedBalancer**         | 1.33±.08  | .885±.017 |

---

![Figure 5: Performance evaluation of FedBalancer with different options on FEMNIST.](image)

![Figure 6: Performance breakdown of FedBalancer into Sample Selection (SS) and Deadline Control (DC).](image)
To configure fixed deadlines for the deadline condition analysis system is integrated to accurately predict the round completion time of selected clients at each round. We leave this as future work.

We suspect the variability of the communication latency would be higher in real deployments as users with mobility and unstable network conditions would be involved. Moreover, the impact of variable communication latency would be more significant when we train larger models with >100M parameters (e.g., BERT [19]) in FL. As FedBalancer is not designed to actively respond to the network connectivity changes of clients in real-time, we expect FedBalancer could be improved further if the client-side network condition analysis system is integrated to accurately predict the round completion time of selected clients at each round. We leave this as future work.

5 DISCUSSION

Local Epoch Training Policies. While FedBalancer was originally designed for FL based on FedAvg [53] that performs full data training per local epoch of client, recent studies such as Oort [38] propose to perform single batch training per local epoch. There are pros and cons in both approaches; single batch training offers more frequent global model update with shorter training time on clients, but this could lead to excessive communication overhead when training large models. Full data training exploits full statistical utility of client data per round, but offers less frequent global model update with longer round. While it is the model developer’s role to determine the option, FedBalancer is readily applicable and improves time-to-accuracy performance on both as we demonstrated in the evaluation.
Effect of Forward Pass at a Client. When FedBalancer selects a clients’ samples, it uses a list of sample loss that is maintained on a client from the beginning of FL, instead of performing a forward pass on the client data at each round. A sample loss at the maintained list is only updated whenever the sample is selected and trained by a client, which may result in containing outdated information. We conducted an experiment to understand the effect of forward pass on performance of FedBalancer, which is shown in Figure 5b. On FEMNIST dataset, integrating the forward pass with FedBalancer resulted in slower training, which supports our design without the forward pass. This is because that the outdated loss values are generally larger than the newly-updated ones, which encourages FedBalancer to select more diverse samples while prioritizing informative samples.

Robustness of Sample Selection. One of the possible limitations of FedBalancer is that it might perform worse on FL tasks with noisy data, as noisy samples are highly likely to be selected by the sample selection module that prioritizes high loss. As we observed the performance improvement with FedBalancer on five real-world user datasets which may already have certain noise level, we expect FedBalancer would be helpful on most FL tasks. To improve further, we could systematically involve robust training approaches at centralized learning [50, 64, 65] to actively deal with noisy data. This is part of our future work.

Potential Bias of Sample Selection. While FedBalancer prioritizes more “informative” samples for training, it might integrate more samples from certain classes or sensitive groups than others, potentially leading to performance degradation on less sampled entities. To address this issue, we could integrate our sample selection strategy with other sample selection or reweighting approaches [60, 74] that are designed to achieve unbiased model training. We leave this as future work.

6 RELATED WORK
We survey closely related work with FedBalancer other than the FL approaches on heterogeneous clients which we discussed earlier in Section 2.2.

Sample Selection in Machine Learning. There are several sample selection approaches in the field of machine learning research that could be arranged in threefold: (1) Curriculum Learning (CL) [9, 26, 29, 73]: CL is a sample ordering technique which trains a network with easier samples in early training stage and gradually increase the difficulty to improve convergence speed and model generalization. However, applying it in FL is challenging as CL require a reference model to determine the difficulty of samples, which hardly exists in FL scenarios. (2) Active Learning (AL) [8, 25, 36, 62]: AL is a sample selection technique on unlabeled data, which interactively queries the user to label new data points that is likely to be more informative to the given task. Applying AL in FL could be non-trivial, as the training data is isolated and the labels are known but not shared externally from the clients. (3) Importance Sampling [5, 34, 52, 61]: Being motivated by the fact that the importance of each training samples is different, researchers have proposed importance sampling techniques to accelerate the model training. While their idea could be brought to FL to prioritize samples during training, determining how many and which samples to use for each training round and when to calculate the sample importance is yet unknown.

Sample Selection in FL. Tuor et al. [69] proposed a scheme that selects relevant clients’ data to the given FL task, but it only selects the dataset before the FL starts. Moreover, it requires an example dataset, which is hardly applicable at FL scenarios where the client data distributions are usually unknown. Li et al. [42] proposes how we can prioritize client training samples with higher importance in FL using gradient norm upper bound [34]. However, their approach does not provide how many samples should be selected per each round. While other methods such as FedSS [12] determines the amount of client training samples during FL, it does not specify which samples to select, simply adopting random sampling of the data. Moreover, combining FedSS with Li et al. is nontrivial, as FedSS assumes random sampling of the data. Unlike previous approaches, FedBalancer is the first systematic framework that actively determines (1) how many and (2) which samples to select during FL to improve time-to-accuracy performance. We believe such design of FedBalancer enables the use of client sample selection to improve time-to-accuracy performance.

Deadline Control in FL. Determining an optimal deadline has been largely overlooked by previous approaches; only SmartPC determines a deadline to enable a specified proportion of the devices to complete a training round. It assumes that every client uses the same set of data for each round of FL. FedBalancer on the other hand utilizes a new deadline control strategy for FL that enables high convergence speed where client training samples dynamically change during FL due to sample selection.

7 CONCLUSION
We presented FedBalancer, a systematic FL framework with sample selection for optimized training process. FedBalancer actively selects the samples with high statistical utility through client-server coordination at each FL round without exposing private information of users. To further accelerate FL with our sample selection, we design adaptive deadline control strategy for FedBalancer to predict the optimal deadline for each round with client sample selection. Our evaluation of on five real-world datasets from three different domains reveal that FedBalancer achieves 1.20∼4.48× speedup over existing FL algorithms with different deadline configuration methods, while improving the model accuracy by 1.1∼5.0%. Our design of FedBalancer is easily applicable on top of orthogonal FL methods, that we demonstrate the joint implementation of FedBalancer with three existing FL algorithms and report the improved time-to-accuracy performance and model accuracy.

ACKNOWLEDGMENTS
We thank anonymous reviewers and our shepherd, Lin Zhong, for their constructive suggestions. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2020R1A2C1004062) and the Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00900).
