FedSEA: A Semi-Asynchronous Federated Learning Framework for Extremely Heterogeneous Devices

Jingwei Sun[1], Ang Li[1], Lin Duan[1], Samiul Alam[4,3], Xuliang Deng[1], Xin Guo[2], Haiming Wang[2], Maria Gorlatova[1], Mi Zhang[3,4], Hai Li[1], Yiran Chen[1]
[1]Department of Electrical and Computer Engineering, Duke University [2]Lenovo Research
[3]The Ohio State University [4]Michigan State University
[1]{jingwei.sun, ang.li630, lin.duan, xuliang.deng, maria.gorlatova, hai.li, yiran.chen}@duke.edu,
[2]{guoxin9, wanghm14}@lenovo.com,[3]{mizhang.1}@osu.edu,[4]{alamsami}@msu.edu

ABSTRACT

Federated learning (FL) has attracted increasing attention as a promising technique to drive a vast number of edge devices with artificial intelligence. However, it is very challenging to guarantee the efficiency of a FL system in practice due to the heterogeneous computation resources on different devices. To improve the efficiency of FL systems in the real world, asynchronous FL (AFL) and semi-asynchronous FL (SAFL) methods are proposed such that the server does not need to wait for stragglers. However, existing AFL and SAFL systems suffer from poor accuracy and low efficiency in realistic settings where the data is non-IID distributed across devices and the on-device resources are extremely heterogeneous. In this work, we propose FedSEA – *a semi-asynchronous FL framework for extremely heterogeneous devices*. We theoretically disclose that the unbalanced aggregation frequency is a root cause of accuracy drop in SAFL. Based on this analysis, we design a training configuration scheduler to balance the aggregation frequency of devices such that the accuracy can be improved. To improve the efficiency of the system in realistic settings where the devices have dynamic on-device resource availability, we design a scheduler that can efficiently predict the arriving time of local updates from devices and adjust the synchronization time point according to the devices' predicted arriving time. We also consider the extremely heterogeneous settings where there exist extremely lagging devices that take hundreds of times as long as the training time of the other devices. In the real world, there might be even some extreme stragglers which are not capable of training the global model. To enable these devices to join in training without impairing the systematic efficiency, FedSEA enables these extreme stragglers to conduct local training on much smaller models. Our experiments show that compared with status quo approaches, FedSEA improves the inference accuracy by 44.34% and reduces the systematic time cost and local training time cost by 87.02× and 792.9×. FedSEA also reduces the energy consumption of the devices with extremely limited resources by 752.9×.

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 ACM 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.
SenSys '22, November 6–9, 2022, Boston, MA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9886-2/22/11…$15.00
https://doi.org/10.1145/3560905.3568538

CCS CONCEPTS

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

KEYWORDS

federated learning, device heterogeneity, edge intelligence

ACM Reference Format:
Jingwei Sun[1], Ang Li[1], Lin Duan[1], Samiul Alam[4,3], Xuliang Deng[1], Xin Guo[2],, Haiming Wang[2], Maria Gorlatova[1], Mi Zhang[3,4], Hai Li[1], Yiran Chen[1]. 2022. FedSEA: A Semi-Asynchronous Federated Learning Framework for Extremely Heterogeneous Devices. In *The 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, Boston, MA, USA, 14 pages. https://doi.org/10.1145/3560905.3568538

1 INTRODUCTION

Federated learning (FL) [4, 10, 25, 27, 35, 36] has attracted increasing attention as a promising technique to empower a vast number of edge devices with artificial intelligence [11, 22, 33, 39]. FL enables massive devices to train a shared model in a federated fashion without transferring their local data. A central server coordinates the FL process, where each participating device communicates only the model parameters with the central server while keeping local data private. Currently, most FL systems follow synchronous protocols [19, 27, 28] which are called synchronous FL. In synchronous FL, all the selected devices are required to complete local training and the server will not perform the aggregation until receiving all the local updates for each communication round. One representative synchronous FL framework is FedAvg [27]. However, due to the computational resource heterogeneity of edge devices in practice, the time needed to complete local training may vary significantly across devices. By applying the synchronous protocol, the server is required to wait for slow devices, i.e., stragglers, in each communication round, which decreases the efficiency of FL. In addition, in realistic settings, the devices may face the risk of network crash and power-off, which makes them fail to complete training and upload the updates. In such cases, the synchronous protocol will lead to extremely long waiting time for the server and poor scalability of FL systems.

Status Quo and their Limitations. To improve the efficiency of FL systems, FL algorithms with asynchronous settings are proposed and can be categorized into two types: asynchronous FL (AFL) [5, 24, 31, 38] and semi-asynchronous FL (SAFL) [6, 26, 34, 37, 40]. The overview of synchronous, asynchronous, and semi-asynchronous

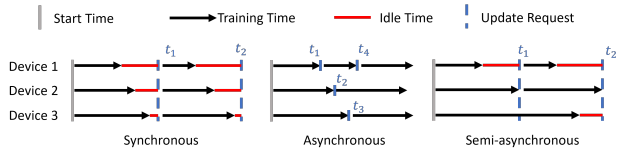

Figure 1: The workflow of different FL protocols.

FL protocols is shown in Figure 1. In asynchronous FL, the server updates the global model as soon as it collects a local update. By applying asynchronous protocols, the efficiency can be improved significantly since the server does not need to wait for the stragglers. However, asynchronous FL always faces two drawbacks: (1) the communication cost of asynchronous FL is much higher compared to synchronous FL due to the frequent communication between the server and devices; (2) the asynchronous protocol leads to the catastrophic staleness of the slow devices, especially under large-scale FL, which introduces errors into the global model and causes accuracy drop. To improve efficiency and mitigate the unacceptable staleness, semi-asynchronous FL was proposed. In semi-asynchronous FL, there are synchronous time points but devices are not required to synchronize with the server in every communication round. By requiring a part of the devices to synchronize with the server, semi-asynchronous FL improves the training efficiency because the server does not need to wait for the slow devices. At the same time, the communication cost of semi-asynchronous FL is controllable by adjusting the synchronization frequency. Semi-asynchronous FL looks more promising in efficiency and performance compared with synchronous and asynchronous FL frameworks. However, existing semi-asynchronous FL frameworks still have some fatal design flaws that set obstacles to achieving high efficiency and hinder the convergence of the global model in realistic settings.

The key to improving the efficiency of semi-asynchronous FL is an effective scheduler of synchronization. Existing semi-asynchronous FL frameworks follow two methodologies to conduct synchronization. The first methodology is that the system sets a fixed time interval T_{fix} [34, 37]. The server conducts aggregation to update the global model and lets a part of devices synchronize with the server every T_{fix}. The second type is that the server holds a cache storing the local updates uploaded from the devices [26, 40]. When the cache is full, the server conducts aggregation based on the local updates stored in the cache and requires a part of the devices to synchronize with the server by distributing the new global model to these devices. In realistic settings where computation power and communication latency on the edge are dynamic along with the time and heterogeneous across devices, both synchronization methods cannot guarantee efficiency. For the first methodology, it might be possible to derive a suitable T_{fix} by profiling all the devices before training. However, the availability of computation and communication resources on devices tends to change along with training, and it is unrealistic to find a fixed T_{fix} suitable for each round of training. A large T_{fix} will make the server wait for long time periods without receiving any local update. A small T_{fix} will aggravate the staleness of slow devices and increases communication cost. The second methodology of using a cache mechanism is adaptive to the resource changes during training, but it cannot solve the problem caused by resource heterogeneity across devices. Suppose the server has a cache length of 100 updates, it might take 1 minute to receive 95 local updates but more than half an hour to receive the rest 5 due to the slow training and bad communication conditions on these devices, which makes the system inefficient.

Besides low efficiency caused by the design of the synchronization scheduler, existing SAFL algorithms also suffer from accuracy drops in the global model under non-IID settings. To mitigate the straggling problems caused by the stale updates, existing SAFL either applies a weight decaying mechanism [6, 24, 31, 34, 38, 40] which assigns a smaller weight to the stale model updates during aggregation, or abandons the model updates that are too stale altogether [14, 26, 30, 37]. These methods are promising in IID settings but would cause accuracy degradation in non-IID settings. In the real world, there will be some slow devices holding unique and important data. Always decaying or abandoning the updates of these slow devices during aggregation will degrade the global model performance on the important data they hold. With the increase in device heterogeneity, there will be extremely lagging devices that are hundreds of times slower than the other devices to train and upload the local models to the server. Furthermore, these devices might be incapable to train the global model (i,e, limited energy, not enough computation power, etc.). Current SAFL designs will exclude these extremely lagging devices from contributing to the global model.

Overview of the Proposed Approach. Motivated by the limitations of existing works, we propose FedSEA, a unified SAFL framework that simultaneously (1) improves the time efficiency in realistic settings; (2) mitigates the accuracy drop caused by applying the semi-asynchronous protocol; and (3) enables extremely lagging devices to contribute to the training of the global model.

To achieve the first goal, we design a practical synchronization scheduler that can reduce the wasted waiting time of the server. Different from existing designs where the server sets a fixed time interval or applies a cache of model updates, our synchronization scheduler can dynamically adapt the time point of synchronization for each round. The server can predict the arriving time of local model updates and conduct synchronization at a time point when the benefit of continuing waiting (i,e, the number of arriving devices within unit time) degrades significantly.

To achieve the second goal, we first investigate the cause of the accuracy drop in SAFL. We theoretically show that besides the straggling problem of stale model updates, unbalanced aggregation frequency across devices is also an essential cause of performance degradation on the global model. Thus, we design a module that can schedule local training configurations of devices and is able to simultaneously mitigate the problem of staleness and unbalanced aggregation frequency. The intuition is that the slow devices train fewer steps such that they can upload updates more frequently. We also empirically and theoretically show that the accuracy drop is mitigated after applying the scheduler of local training configurations.

To achieve the third goal, FedSEA enables the extremely lagging devices to train smaller models rather than the global model. There are some works enabling the slow devices to train heterogeneous models [19, 20], but the small models in these works are subnets of the global model which limits the improvement of computation

Table 1: Comparison between FedSEA and existing AFL/SAFL frameworks.

Method	Time Efficiency in Realistic Settings	High Global Model Performance	Participation of Extreme Stragglers
AsyncFL [38]	✓	X	X
FedCS [30]	✓	X	X
SAFA [37]	X	X	X
FedSEA	✔	✔	✔

reduction. FedSEA does not have this limitation of the model architecture. The extremely lagging devices can train an MLP while the global model is CNN or LSTM, which reduces the computation cost of the lagging devices significantly.

System Implementation and Experimental Results. We implemented FedSEA and conducted extensive experiments to evaluate its performance. We applied FedSEA to develop three representative deep learning applications on edge devices. These applications are developed based on five datasets that are widely used in the computer vision and mobile sensing community. In addition, we also implemented four status quo approaches for comparisons, including FedAvg, FedCS, AsyncFL, and SAFA. Our results show that:

- FedSEA outperforms the compared baselines, improving inference accuracy by 11.8%-44.34% and reducing time cost by 1.27×-87.2×.
- FedSEA significantly reduces on-board resource cost caused by local training. Specifically, slow devices in FedSEA achieve as much as 792.9× reduction in local training time, 4.9× reduction on memory footprint, and 752.9× savings on energy consumption.

Summary of Contributions. To the best of our knowledge, FedSEA represents the first SAFL framework that practically improves time efficiency and global model performance under realistic settings where the devices are extremely heterogeneous and have dynamic on-board resource availability. We provide a theoretical explanation of the accuracy drop of the global model when applying for existing SAFL works, and how FedSEA mitigates this accuracy degradation. Table 1 provides a comparison between FedSEA and status quo AFL and SAFL methods. FedSEA proposes a series of novel techniques that effectively address the limitations of status quo methods. We believe our work represents a significant step towards enhancing the efficiency of FL systems.

2 BACKGROUND AND MOTIVATION

2.1 Heterogeneous and Dynamic Resource Availability on Devices

In the real world, the devices participating in FL are heterogeneous in terms of on-board resources. In addition, the edge devices usually run several primary tasks besides FL tasks, which leads to dynamic on-board resource availability. Such heterogeneous and dynamic on-board resource availability makes the synchronous protocol unrealistic when deploying FL frameworks in practice. To show the heterogeneous and dynamic resource across devices and the inefficiency of synchronous FL, we conduct experiments on two common edge devices: Jetson TX2 and Raspberry Pi 4. We train a LeNet-5 using CIFAR10 on these two devices. We set the batch size as 10 and measure the time cost of one step of training. To evaluate the impact of dynamic resource availability on the training time, we run two primary tasks on Jetson TX2: playing a 4-K video and conducting real-time object detection. The training time of two devices in different statuses is shown in Figure 2. It is shown that Raspberry Pi 4 needs nearly 100× longer time to complete one step of training compared with the idle Jetson TX2. When conducting the object detection application, the Jetson TX2 is nearly 10× slower than the idle status to complete training. Such huge heterogeneous and dynamic resource availability on board makes it necessary to apply asynchronous and semi-asynchronous protocols when deploying FL systems in real life.

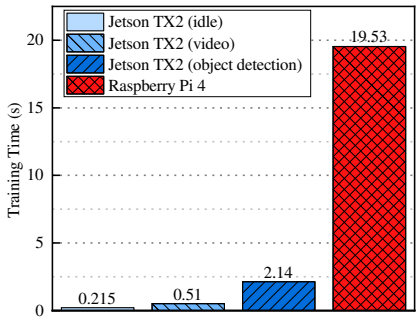

Figure 2: Time cost of one step of training on different devices.

2.2 Performance Degradation of Federated Learning with Asynchronous Settings

To reduce the problem of staleness in asynchronous FL, existing works apply weight decaying mechanisms that assign a smaller weight to the stale model updates during aggregation or abandon the model updates that are too stale from training. These methods are promising in IID settings but would cause accuracy degradation in non-IID settings. In the real world, there may be some slow devices holding unique and important data. Always decaying or abandoning the updates of these slow devices during aggregation will degrade the global model performance on the long-tail data they hold. We conduct experiments of FedAsync and SAFA on MNIST and CIFAR under non-IID settings. The evaluation is based on two types of devices: fast devices which are idle Jetson TX2 and slow devices which are Jetson TX2 conducting real time object detection. We partition the data following the configurations in [19], but only slow devices have access to the data of class 9-10 such that the slow devices hold unique classes of data. We set local epoch E as 1 and batch size B as 32. We apply SGD optimizer and set the learning rate η to 0.01. In each synchronization, the server randomly selects 10 idle devices to participate in training. We apply simple CNNs with 2 convolutional layers to both datasets. We conduct 1000 communication rounds of training for MNIST and 2000 communication rounds for CIFAR10. The results are shown in Table 2. It is shown that there is a significant accuracy drop when applying the weight decaying mechanism under non-IID settings.

Table 2: Results of FedAsync and SAFA under non-IID settings.

Methods	MNIST	CIFAR10
FedAvg	96.63	55.84
SAFA	93.85	51.52
FedAsync	83.26	43.14

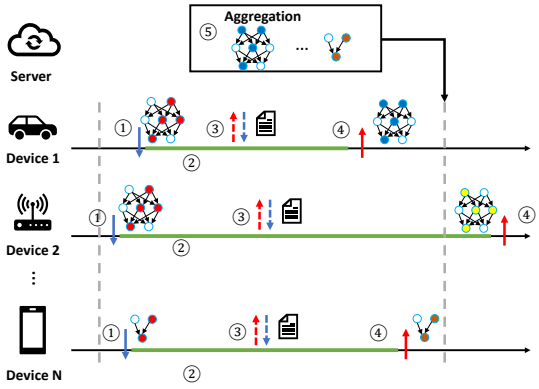

Figure 3: Overview of FedSEA.

2.3 The Exclusion of Extremely Lagging Devices

Due to the huge device heterogeneity in the real world, there are some devices with extremely limited computation and communication resources. These devices might take hundreds of times as long as the training time of the other devices to finish local training and upload the updates to the server. Furthermore, some extremely lagging devices might have no capacity to train the global model which is trained by the other devices. Even though these devices will not degrade the efficiency of FL with asynchronous settings, these extremely lagging devices are more likely to be excluded from training due to the extreme staleness or incapacity of training the global model. A natural option is to support heterogeneous model architectures across devices. However, existing frameworks[9, 19, 20] supporting model heterogeneous settings have three problems: 1) Existing works usually derive sub-networks from the dense network and these extremely lagging devices might not be able to even train such a sub-network. For example, the global model is CNN, whose sub-networks are also CNN, but the extremely lagging devices might only be able to train an MLP. 2) Even if the devices can train a sub-network of the global model, aggregating sub-models will hinder the convergence of the global model. Existing FL frameworks support heterogeneous architectures by allowing devices to train sub-models to achieve good performance on personalized models, but the global model performance degrades significantly. 3) The process of seeking the sub-network usually starts from training the dense network, which is inapplicable in realistic settings. Thus, we need to design a practical method that helps the devices with extremely limited resources to join in training without introducing additional computation and communication overhead to the devices.

3 DESIGN

3.1 Overview

Figure 3 depicts an overview of the proposed framework. Our system follows the general protocol of semi-asynchronous FL and has a synchronous time point for each communication round. At the start of each communication round, the server randomly selects a set of available devices which are not conducting local training and distributes the up-to-date global model to these devices (①). After receiving the starting signal and the global model, the selected devices set up training configurations and run local training (②). During local training, the devices continue communicating with the server. The server adapts the local training configuration of each device based on the computation power and network latency of all participating devices to improve convergence and efficiency (③). The communication between the devices and the server is unblocked, such that the devices can upload the model updates to the server once they complete local training (④).

Our framework also considers the devices whose resources are extremely constrained that cause unacceptable staleness or are incapable to train the same model architecture as the other devices. These extremely lagging devices conduct local training based on the model with a smaller architecture. We do not introduce additional computation overhead to the devices in FedSEA to handle device heterogeneity, and the energy and communication cost can be significantly reduced for the extremely lagging devices. At the end of each communication round, the server conducts aggregation on the received heterogeneous model updates (⑤). To aggregate the heterogeneous local models, the server extracts the knowledge contained in the smaller model on extremely lagging devices and fuses it into the larger global model.

The above process (①-⑤) repeats until reaching a predefined number of communication rounds.

3.2 Design Challenges

The design of our system has three key challenges:

Challenge#1: How to mitigate the performance degradation of the global model? In AFL, the lagging devices are inevitable, and directly aggregating stale model updates will do harm to the convergence of the global model. Previous works consider that the accuracy drop comes from the straggling problem caused by the stale model updates. They apply weight decaying mechanisms or abandon stale models during aggregation to solve the straggling problem but cause a serious accuracy drop of the global model. The weight decaying mechanism and abandoning stale models can indeed reduce the straggling problem caused by the stale model updates, but there should be some other reasons that lead to global model performance degradation under asynchronous settings. Therefore, we need to further explore the essential cause of performance degradation of the global model and design a method that can effectively alleviate the problem of accuracy drop.

Challenge#2: How to aggregate the smaller models trained by the devices with extremely limited resources into the global model? We design a module that allows extremely lagging devices to train a smaller model. This module is based on an unlabeled auxiliary dataset on the server which is more realistic since it is

easy for the server to collect public data, but it is very expensive to label these data samples. At the beginning of a round, this module distills the knowledge of the global model into a smaller model, and the extremely lagging devices conduct local training based on this smaller model. After the lagging device completes local training and uploads the updated small model to the server, this module will infuse the knowledge of the updated smaller models into the global model. The design is inspired by knowledge distillation (KD) [13, 15, 32] in deep learning, but KD cannot be directly applied to our system for two challenges: (1) KD is based on labeled data while in our system there is no labeled public data. Conducting distillation based on unlabeled data will introduce unacceptable error; (2) conventional KD is to distill knowledge from a large model to a small model. However, in our design, we need to infuse information from a small local model into the large global model, and directly applying KD will cause a catastrophic overfitting problem in our system. Hence, we need to design novel distillation and infusion methods to support effective knowledge transfer between the small local model and the global model, such that the extremely lagging devices can participate in training by training based on a small model.

Challenge#3: How to determine the synchronization time point in realistic settings? The key point of improving the efficiency of semi-asynchronous FL is the design of an effective scheduler of synchronization. In the real world, due to the dynamic and heterogeneous resource availability, the distributions of local updates' arriving time are different across communication rounds. Setting a fixed time interval or applying the cache mechanism cannot guarantee efficiency in practice. Natural thinking is conducting profiling for the device participating in training before the start of each communication round and deriving the real-time resource availability. However, conducting extra profiling for each communication round is extremely inefficient. In addition, even if the server has access to the real-time resource availability of devices, it is still not clear when to finish this communication round and conduct synchronization. Thus, it is a challenge to design an effective and efficient synchronization scheduler and improve the efficiency of the pipeline in realistic settings.

3.3 Training Configuration Scheduler

As described in section 2.2, although decaying the weight of stale model updates as previous works can reduce the error caused by the stale model, it still causes an accuracy drop on the data held by the slow devices. We analyze that this accuracy drop mainly comes from the low aggregating frequency of slow devices. Such unbalanced aggregating frequency causes unbalanced performance across the global data distribution. To show the impact of unbalanced aggregating frequency theoretically, we formulate the problem in a simplified setting. The learning objective of FL is defined as:

$$\boldsymbol{W} = \min_{\boldsymbol{W}} \{F(\boldsymbol{W}) \triangleq \sum_{k=1}^{N} p^k F^k(\boldsymbol{W})\}, \tag{1}$$

where $\boldsymbol{W}$ is the weights of the global model, N represents the number of devices, F^k is the local objective of the k-th device, p^k is the weight of the k-th device, $p^k \geq 0$ and $\sum_{k=1}^{N} p^k = 1$.

Equation 1 is computed in an iterative device-server communication fashion. For a given communication round (e.g. the t-th), the central server first randomly selects K devices according to a probability vector $[q_1, q_2, ..., q_N]$ to compose a set of participating devices $\mathbb{S}_t$ and then broadcasts the latest global model $\boldsymbol{W}_{t-1}$ to these devices. Afterwards, each device (e.g. the k-th) in $\mathbb{S}_t$ performs I iterations of local training using their local data following:

$$\boldsymbol{W}_{t,i+1}^k \leftarrow \boldsymbol{W}_{t,i}^k - \eta_{t,i} \nabla F^k(\boldsymbol{W}_{t,i}^k, \xi_{t,i}^k), \tag{2}$$

where $\eta_{t,i}$ is the learning rate, $\xi_{t,i}^k$ is a batch of data samples uniformly chosen from the k-th device, and $\boldsymbol{W}_{t,0}^k$ is initialized as $\boldsymbol{W}_{t-1}$. Finally, the server averages the local updates of the selected K devices and updates the global model as follows:

$$\boldsymbol{W}_t \leftarrow \frac{N}{K} \sum_{k \in \mathbb{S}_t} p^k \boldsymbol{W}_{t,I}^k. \tag{3}$$

With the above formulation, we analyze the impact of the probability vector $[q_1, q_2, ..., q_N]$ on the convergence of $F(W)$. Before presenting our theoretical results, we first make the following Assumptions 1-4 same as [21].

ASSUMPTION 1. *$F^1, F^2, ..., F^N$ are L-smooth: $\forall \boldsymbol{V}, \boldsymbol{W}$, $F^k(\boldsymbol{V}) \leq F^k(\boldsymbol{W}) + (\boldsymbol{V} - \boldsymbol{W})^T \nabla F^k(\boldsymbol{W}) + \frac{L}{2} ||\boldsymbol{V} - \boldsymbol{W}||_2^2$.*

ASSUMPTION 2. *$F_1, F_2, ..., F_N$ are μ-strongly convex: $\forall \boldsymbol{V}, \boldsymbol{W}$, $F^k(\boldsymbol{V}) \geq F^k(\boldsymbol{W}) + (\boldsymbol{V} - \boldsymbol{W})^T \nabla F^k(\boldsymbol{W}) + \frac{\mu}{2} ||\boldsymbol{V} - \boldsymbol{W}||_2^2$.*

ASSUMPTION 3. *Let ξ_t^k be sampled from the k-th device's local data uniformly at random. The variance of stochastic gradients in each device is bounded: $\mathbb{E}||\nabla F^k(\boldsymbol{W}_{t,i}^k, \xi_{t,i}^k) - \nabla F^k(\boldsymbol{W}_{t,i}^k)||^2 \leq \sigma_k^2$ for $k = 1, ..., N$.*

ASSUMPTION 4. *The expected squared norm of stochastic gradients is uniformly bounded, i.e., $\mathbb{E}||\nabla F^k(\boldsymbol{W}_{t,i}^k, \xi_{t,i}^k)||^2 \leq G^2$ for all $k = 1, ..., N$, $i = 0, ..., I-1$ and $t = 0, ..., T-1$.*

We define F^* and F^{k*} as the minimum value of F and F^k and the total number of rounds is T. Then, we have the following theorem about the convergence of $F(\boldsymbol{W}_T)$.

THEOREM 1. *Let Assumptions 1-4 hold. If q_i does not follow uniform distribution, which means that the uploading frequency of devices is unbalanced, then there will be error involved to hinder the convergence of $\mathbb{E}[F(\boldsymbol{W}_T)]$. Except for the error involved, the convergence rate is also hindered by the variance of q_i as $O\left(\frac{1+N^2\sigma_q^2}{T}\right)$, where σ_q^2 is the variance of q_i.*

PROOF. Our proof is mainly inspired by [21]. Following [21], we describe the training process of the setting in Theorem 1 as: for all $k \in [N]$

$$\begin{aligned} \boldsymbol{V}_{t,i+1}^k &= \boldsymbol{W}_{t,i}^k - \eta_{t,i} \nabla F^k(\boldsymbol{W}_{t,i}^k, \xi_{t,i}^k), \\ \boldsymbol{W}_{t,i+1}^k &= \begin{cases} \boldsymbol{V}_{t,i+1}^k & \text{if } i+1 \neq I, \\ \text{samples } \mathbb{S}_t \text{ with } \{q_j\}_{j \in [N]} & \\ \text{and average } \{\boldsymbol{V}_{t,i+1}^k\}_{k \in \mathbb{S}_t} & \text{if } i+1 = I. \end{cases} \end{aligned} \tag{4}$$

Similar with [21], we define two virtual sequences $\bar{\boldsymbol{V}}_{t,i} = \sum_{k=1}^{N} p_k \boldsymbol{V}_{t,i}^k$ and $\bar{\boldsymbol{W}}_{t,i} = \sum_{k=1}^{N} p_k \boldsymbol{W}_{t,i}^k$ to support the analysis. For convenience, we also define $g_{t,i} = \sum_{k=1}^{N} p_k \nabla F^k(\boldsymbol{W}_{t,i}^k)$ and $\bar{g}_{t,i} = \sum_{k=1}^{N} p_k \nabla F^k(\boldsymbol{W}_{t,i}^k, \xi_{t,i}^k)$. Therefore, $\bar{\boldsymbol{V}}_{t,i+1} = \bar{\boldsymbol{W}}_{t,i+1} - \eta_{t,i} g_{t,i}$.

Note that

$$\begin{aligned}\|\bar{\boldsymbol{W}}_{t,i+1}-\boldsymbol{W}^*\|_2 =&\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}+\bar{\boldsymbol{V}}_{t,i+1}-\boldsymbol{W}^*\|_2\\ =&\underbrace{\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}\|_2}_{A_1}+\underbrace{\|\bar{\boldsymbol{V}}_{t,i+1}-\boldsymbol{W}^*\|_2}_{A_2}\\ &+\underbrace{2\left\langle\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1},\bar{\boldsymbol{V}}_{t,i+1}-\boldsymbol{W}^*\right\rangle}_{A_3}.\end{aligned}\tag{5}$$

If $q_i=\frac{1}{N}$ for $i\in[N]$ which means that the probability of sampling is balanced across devices, A_3 would vanish and $\|\bar{\boldsymbol{W}}_{t,i+1}-\boldsymbol{W}^*\|_2$ is able to converge to zero with specific learning rates. However, if the sampling is unbalanced across devices as claimed in Theorem 1, A_3 cannot vanish and the bound $\|\bar{\boldsymbol{W}}_{t,i+1}-\boldsymbol{W}^*\|_2$ can achieve when $i+1=I$ without additional assumptions is:

$$\begin{aligned}\|\bar{\boldsymbol{W}}_{t,i+1}-\boldsymbol{W}^*\|_2 \le&2\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}\|_2+2\|\bar{\boldsymbol{V}}_{t,i+1}-\boldsymbol{W}^*\|_2\\ \le&(2-2\eta_{t,i})\mathbb{E}\|\bar{\boldsymbol{W}}_{t,i}-\boldsymbol{W}^*\|_2+2\eta_{t,i}^2(Q+C)\\ =&\underbrace{(1-2\eta_{t,i})\mathbb{E}\|\bar{\boldsymbol{W}}_{t,i}-\boldsymbol{W}^*\|_2+2\eta_{t,i}^2(Q+C)}_{D_1}\\ &+\underbrace{\mathbb{E}\|\bar{\boldsymbol{W}}_{t,i}-\boldsymbol{W}^*\|_2}_{D_2},\end{aligned}\tag{6}$$

where $Q=\sum_{k=1}^{N}p_k^2\sigma_k^2+6L\Gamma+8(I-1)^2G^2$, C is the upper bound of $\frac{1}{\eta_{t,i}^2}\mathbb{E}_{\mathbb{S}_t}\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}\|_2$. The first inequality comes from Cauchy-Schwarz inequality and AM-GM inequality. The second inequality comes from Lemma 1-3 and Lemma 5 in [21]. From Equation 6 we can see that the existence of D_2 hinders the convergence of $\|\bar{\boldsymbol{W}}_{t,i+1}-\boldsymbol{W}^*\|_2$, and D_2 comes from the failure of vanishment of A_3 in Equation 5. We have shown that A_3 in Equation 5 does not vanish because of the inequality of q_i, which represents the unbalanced uploading frequency across devices. Thus, we can derive that if the uploading frequency of devices is unbalanced, then there will be error involved to hinder the convergence of the learning objective.

If we do not consider the error caused by the unbalanced uploading frequency in Equation 5, which means that we assume A_3 is able to vanish, then we can analyze the convergence rate of $\|\bar{\boldsymbol{W}}_{t,i}-\boldsymbol{W}^*\|_2$ by deriving C, which is the upper bound of $\frac{1}{\eta_{t,i}^2}\mathbb{E}_{\mathbb{S}_t}\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}\|_2$:

$$\begin{aligned}&\frac{1}{\eta_{t,i}^2}\mathbb{E}_{\mathbb{S}_t}\left\|\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,i+1}\right\|_2\\ =&\frac{1}{\eta_{t,i}^2}\mathbb{E}_{\mathbb{S}_t}\left\|(\bar{\boldsymbol{W}}_{t,i+1}-\bar{\boldsymbol{W}}_{t,0})+(\bar{\boldsymbol{V}}_{t,i+1}-\bar{\boldsymbol{V}}_{t,0})\right\|_2\\ =&\frac{1}{\eta_{t,i}^2}\left\|\sum_{k=1}^{N}q_k\left(\boldsymbol{V}_{t,i+1}^k-\boldsymbol{V}_{t,0}^k\right)-\sum_{k=1}^{N}\frac{1}{N}\left(\boldsymbol{V}_{t,i+1}^k-\boldsymbol{V}_{t,0}^k\right)\right\|_2\\ \le&\frac{N}{\eta_{t,i}^2}\sum_{k=1}^{N}\left\|\left(q_k-\frac{1}{N}\right)\left(\boldsymbol{V}_{t,i+1}^k-\boldsymbol{V}_{t,0}^k\right)\right\|_2\\ \le&\frac{N}{\eta_{t,i}^2}\sum_{k=1}^{N}\left(q_k-\frac{1}{N}\right)^2 I\sum_{j=0}^{i}\mathbb{E}\left\|\eta_{t,j}\nabla F^k(\boldsymbol{W}_{t,j}^k,\xi_{t,j}^k)\right\|_2\\ \le&4N(N-1)I^2G^2\sigma_q^2,\end{aligned}\tag{7}$$

where σ_q^2 is the variance of q_k. The first inequality comes from the convexity of $\|\cdot\|_2$. The second inequality comes from Cauchy-Schwarz inequality. In the last inequality, we use the fact that $\eta_{t,i}$ is non-increasing and $\eta_{t,0}\le 2\eta_{t,I}$. Then from Theorem 3 in [21], we derive that except for the error involved by the unbalanced uploading frequency, $\mathbb{E}[F(\boldsymbol{W}_T)]$ converges as $\mathcal{O}\left(\frac{1+N^2\sigma_q^2}{T}\right)$.

□

Remark 1. *Besides the error involved in the convergence, unbalanced uploading frequency also hinders the scalability of an FL system. From theorem 1 we can see that, when the variance of q_i is not zero, the convergence will be slower when the number of devices N is larger.*

Based on this analysis, we balance the uploading frequencies of different devices by adjusting the training configuration of devices. The basic idea is that the slow devices should train fewer steps such that the error caused by the staleness and the performance unbalance can be reduced simultaneously. Following this idea, we reduce the local training steps of slow devices and conduct experiments following the settings in Section2.2. We reduce the steps of slow devices of SAFA by 5 and 10 times which are noted as SAFA(#step ↓,5×) and SAFA(#step ↓,10×), respectively. The results are shown in Table3.

Table 3: Results of reducing the training steps of slow devices.

Methods	MNIST	CIFAR10
FedAvg	96.63	55.84
SAFA	93.85	51.52
FedAsync	83.26	43.14
SAFA(#step ↓,5×)	94.15	50.9
SAFA(#step ↓,10×)	93.54	50.62
SAFA(#step ↓, lr↑,5×)	96.32	54.34
SAFA(#step ↓, lr↑,10×)	96.36	54.13

Algorithm 1 Scheduler of Training Configuration.

Input: The anticipated end training time t_{end}^j; Anticipated time for this round T_a; Pre-set #batch B; Pre-set Learning rate η;
Output: #batch $\hat{B}$; learning rate $\hat{\eta}$

```
function TRAINING_CONFIG_SCHEDULER(t^i_end, T_a, B)
    if t^i_end > αT_a then
        B̂ = ⌊B αT_a / t^i_end⌋;
        η̂ = η t^i_end / αT_a;
    else
        B̂ = B;
        η̂ = η;
    end if
    return {B̂, η̂};
end function
```

We can see that simply reducing training steps cannot improve accuracy. The reason is that even if the slow devices upload the model updates more frequently, the fewer training steps will obliterate the contribution of slow devices during aggregation. To solve this problem, we increase the learning rate of the slow devices of SAFA by the same times of reducing training steps. The results are shown

in Table 3 and are listed as SAFA(#step ↓, lr↑,5×) and SAFA(#step ↓, lr↑,10×), respectively. It is shown that by increasing the learning rate of slow devices, the accuracy gap between synchronous FL and semi-asynchronous FL is nearly eliminated. Following this preliminary result, we design a ***training_configuration_scheduler*** on the server to adjust the training configurations of training devices, and the detailed algorithm is shown in algorithm 1. It is notable that this scheduler depends on the anticipated training time of the device and the anticipated lasting time of this round. These two variables are generated by the ***end_time_predictor*** on the server, which will be introduced later. Involving the *end_time_predictor* is also the key of our design since in the real world we cannot reschedule the slow devices by foreseeing the latency and training time of all the devices in the system as we did in Table 3.

3.4 Distillation Module and Infusion Module

We have shown that dynamically scheduling training configurations of devices can improve the balance of aggregation across devices, thereby improving the performance of the global model. However, in the real world, there are some devices with extremely limited resources. To guarantee a balanced aggregation frequency, these devices might be forced to train very few iterations before uploading the model updates to the server. In addition, some extreme stragglers might derive $\hat{B}$ less than 1 by directly applying algorithm 1 or even have not enough power to train the global model. To support the devices with extremely limited resources to participate in training and improve the convergence of the global model, we design a ***distillation_module*** and an ***infusion_module*** which enable the extremely lagging devices to train models with smaller architectures. By doing this, the extremely slow devices are not required to have unacceptably large manipulations on the training configurations since training the smaller model will be much faster. Moreover, the devices which have not enough resource to train the global model can also participate in training by training a smaller model. The design of *distillation_module* and *infusion_module* is inspired by knowledge distillation (KD) [15] in deep learning, and the workflow is shown in Figure 4. At the start of each communication round t, the server utilizes the *distillation_module* to distill the knowledge from the up-to-date global model W_t to a smaller model $\hat{W}_t$ (①) and distributes $\hat{W}_t$ to an extremely lagging device (says the k-th) (②). Then the k-th device conducts local training based on $\hat{W}_t$ (③). After τ communication rounds, the k-th device completes local training and uploads $\hat{W}^k_{t+\tau}$ to the server (④) and the server stores it into a cache. Before the end of this communication round, the server aggregates the cached small local models uploaded by the set of slow devices $\mathbb{S}_{slow}(t+\tau)$ in this round and gets a small model $\hat{W}_{t+\tau}$ (⑤). Then the server calls the *infusion_module* to infuse the knowledge from $\hat{W}_{t+\tau}$ to the global model in current round and gets a large model $W'_{t+\tau}$ (⑥). At the end of this round, the infused model $W'_{t+\tau}$ is aggregated into the global model with the weight of $\sum_{k\in\mathbb{S}_{slow}(t+\tau)} p^k$.

In our design, there is an unlabeled public dataset D_{pub} on the server as an auxiliary dataset. This is a practical setting since it is easy for the server to collect public data but very expensive to label the data points. For example, if a company wants to develop an FL application of face recognition, this company will store a public dataset of face analysis such as CelebA [23] on the central server.

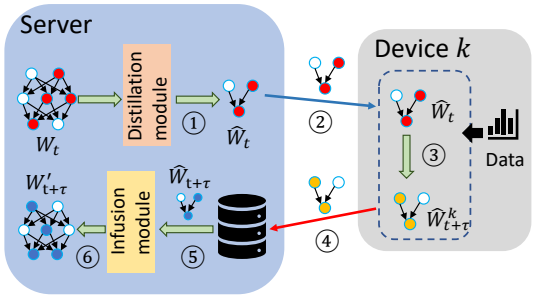

Figure 4: The workflow of *distillation_module* and *infusion_module*.

For the *distillation_module*, we utilize the soft label produced by the global model W_t to distill knowledge to a smaller model $\hat{W}_t$. Given a sample ξ_i from D_{pub}, the logits output by W_t and $\hat{W}_t$ are $f(\xi_i; W_t)$ and $\hat{f}(\xi_i; \hat{W}_t)$, respectively. Given a logit z, the probability vector is $p = g(z;T)$, where $g(z;T)$ is defined as:

$$p_i = g(z;T)_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}, \tag{8}$$

where T is the temperature of distillation. Then the learning objective of $\hat{W}_t$ in *distillation_module* is formulated as:

$$\hat{W}_t = \arg\min_{W} \sum_{i}^{|D_{pub}|} D_{KL}(g(f(\xi_i; W_t);T)||g(\hat{f}(\xi_i; W);T)). \tag{9}$$

Compared with the conventional KD, the *infusion_module* has two key unique challenges. First, the public dataset on which the distillation is based has no true labels. Only applying the soft label generated by the global model in *distillation_module* is acceptable since the global model contains the knowledge from a tremendous number of devices, which is less overfitted and of high quality. However, only applying the soft label generated by the aggregated small model $\hat{W}_{t+\tau}$ in *infusion_module* will introduce significant noise into the global model. The reason is that $\hat{W}_{t+\tau}$ is aggregated by the local models uploaded by very few extremely lagging devices in the system, which is usually much more overfitted than the global model. The second challenge is that conventional knowledge distillation is to distill the knowledge from a larger model to a smaller model, but the *infusion_module* in our design is to infuse the knowledge from a smaller model to a larger model. When teaching a model with higher representation capacity by learning knowledge from a simpler model, directly applying KD in *infusion_module* will cause an unacceptable overfitting problem. To reduce the error from the small model $\hat{W}_{t+\tau}$ infused to the global model, *infusion_module* teaches $W'_{t+\tau}$ to learn knowledge from both the aggregated small model $\hat{W}_{t+\tau}$ and current global model $W_{t+\tau}$. Then, the infusion objective of $W'_{t+\tau}$ in *infusion_module* can be formulated as:

$$L_{inf}(W'_{t+\tau}) = \alpha_{inf} \sum_{i}^{|D_{pub}|} D_{KL}(g(\hat{f}(\xi_i; \hat{W}_{t+\tau}); T) || g(f(\xi_i; W'_{t+\tau}); T)) + (1-\alpha_{inf}) \sum_{i}^{|D_{pub}|} D_{KL}(g(f(\xi_i; W_{t+\tau}); T) || g(f(\xi_i; W'_{t+\tau}); T)), \tag{10}$$

where $\alpha_{inf} \in (0, 1)$ controls the trade-off between infusing more knowledge from the local models and the global model. To mitigate the overfitting problem caused by infusing $\hat{W}_{t+\tau}$ to $W'_{t+\tau}$, we add an L2-norm regularization term to the learning objective. Then the learning objective of $W'_{t+\tau}$ is formulated as:

$$W'_{t+\tau} = \arg\min_{W} L_{inf}(W; \alpha_{inf}) + \beta \|W\|_2. \tag{11}$$

As shown in Figure 4, there is no additional computational overhead on local devices by applying the *distillation_module* and *infusion_module*. All the computational overhead introduced on the server is at most one time of distillation and infusion in one communication round. In real systems, the central servers are usually much more powerful than the edge devices, and only one time of distillation and infusion on the server would not impair the efficiency of the system.

3.5 End Time Predictor and Round Time Scheduler

To improve the efficiency of the pipeline, we need to design a practical *synchronization_scheduler* to manage when to stop the current communication round and conduct synchronization. The significant information needed to schedule the deadline of this communication round is the anticipated end time of local training of the devices. In addition, as stated in Section 3.3, the design of *training_configuration_scheduler* also requires the end time of local training of each device to adapt its local training configurations. Thus, we design an *end_time_predictor* to predict the end time of local training of each device. The natural thought is to profile the training time of each device before they join the FL training. However, in real life, the devices would often have dynamic computation power, which makes it necessary to profile a device before the start of training in each round of participation. By doing this, the devices will suffer from tremendous computation overhead and the efficiency of the whole system will be significantly harmed. In addition, to improve the precision of prediction, it is always required to profile multiple steps of training, which increases the overhead further. To avoid incurring additional computation overhead and improve efficiency, we propose to conduct profiling as long as the starting steps of local training. After receiving the global model, the participating devices (says the k-th) will start training and record the training time of each step (says the b-th) $t^k_{train,b}$. After several steps, the k-th device will send the collected training time of steps $\{t^k_{train,b}\}$ and the number of training steps B^k to the server. The specific time point when the devices upload the profiling information to the server will be introduced in the next section. After receiving the profiling information, the server calls the *end_time_predictor* to predict the end time of training t^k_{end} of the k-th device.

The basic assumption of *end_time_predictor* is that within a certain interval of local training, the training time of each step follows a normal distribution. The *end_time_predictor* inferences the statistics of the distribution of training time of one device. Then it derives the predicted end time point of local training by which there is an 80% probability that this device can complete local training and successfully upload the updated model to the server. The detailed algorithm of *end_time_predictor* is shown in Algorithm 2.

Algorithm 2 End time predictor of local training.

Input: Local training time of iterations $\{t_{train,k}\}$; Start training time t_{start}; Latency $t_{latency}$; Number of batch B;
Output: End time of training t_{end};

function END_TIME_PREDICTOR($t_{start}, t_{latency}, \{t_{train,k}\}$)
  $\hat{\mu} = \frac{1}{K} \sum_{k=1}^{K} t_{train,k}$;
  $\hat{\sigma^2} = \frac{1}{K-1} \sum_{k=1}^{K} (t_{train,k} - \hat{\mu})^2$;
  $t_{end} = t_{start} + t_{latency} + B\hat{\mu} + \sqrt{B}\hat{\sigma}\Phi^{-1}(0.8)$;
  return t_{end};
end function

With the set of predicted end time of local training $\{t^k_{end}\}$ cached on the server, the *synchronization_scheduler* determines when to finish this communication round. The specific time point when the *synchronization_scheduler* will conduct scheduling will be introduced in the next section. The intuition of *synchronization_scheduler* is to find the "bonus" time point after which the "benefit" of waiting will decrease, which means that fewer devices can arrive by waiting for the same time. The detailed design of *synchronization_scheduler* is shown in Algorithm 3. An important parameter of the *training_configuration_scheduler* and *synchronization_scheduler* is the anticipated time of the round T_a. At the start of the first round, we predict the training time of the selected devices with the *end_time_predictor* and derive the initial T_a by averaging the predicted training time of these devices. In the following rounds, we update T_a with the weighted sum of the latest T_a and the real lasting time of the last round.

Algorithm 3 Scheduler of Synchronization.

Input: The set of anticipated end training time $\{t^j_{end}\}$; Anticipated time for this round T_a;
Output: The end time of this round T;

function SYNCHRONIZAION_SCHEDULER($\{t^j_{end}\}, T_a$)
  Sort $\{t^j_{end}\}$ in increasing order to get a list Q;
  for $k = 1, 2, ...$ **do**
    if $Q_{k+1} - Q_k > 0.5T_a$ or $Q_{k+1} > 1.5T_a$ **then**
      $T = Q_k$;
    end if
  end for
  return T;
end function

3.6 Workflow Overview

In our system, there is no additional computation overhead on the devices, and the main scheduling operations happen on the server.

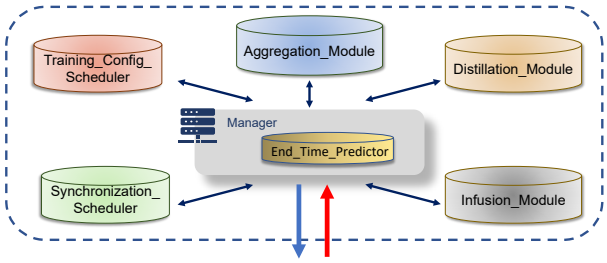

Figure 5: The structure of the server.

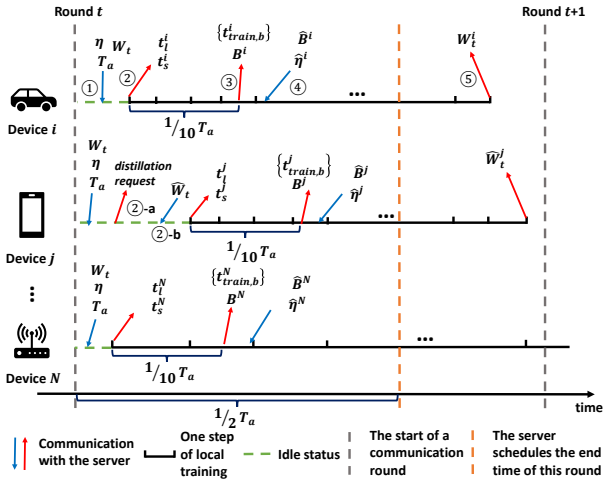

Figure 6: The workflow of FedSEA.

The structure of the central server is shown in Figure 5. The *aggregation_module* is responsible for aggregating the received model updates and we follow the aggregation method of FedAvg. The manager is responsible for communicating with edge devices and coordinating the other modules on the server. As shown in Figure5, the *end_time_predictor* is embedded in the manager.

The workflow of our system is shown in Figure 6. In each communication round (says the t-th), the manager randomly selects a set of available devices which are not conducting local training to participate in FL training. The manager sends the up-to-date global model W_t, the anticipated lasting time of this round T_a, the default learning rate η, and the start signal to the selected devices at the start of this communication round, and the devices receive the information later due to the network latency (①). If the device (says the i-th) has enough computation power and energy to conduct training based on W_t, it will start to set up the local training program. Due to the network latency and time needed to set up the training program, the start time of local training is not the time point when a device receives the start signal of this round. At the start of local training, this selected device sends the time stamp of the training start t_s^i and the network latency of transferring the neural network t_l^i back to the manager (②). During local training, the device records the training time of each batch (says the b-th batch) $t_{train,b}^i$. At $\frac{1}{10}T_a$ after the arrival of the start signal, the device uploads the training time set $\left\{t_{train,b}^i\right\}$ of batches it has trained and the total number of batches it needs to train B^i to the server (③). The numbers of local training batches per round in our platform of different applications are set to be more than 30. Therefore, we ask the devices to upload the training time set after $\frac{1}{10}T_a$, such that the server can have local training time of more than 3 batches to estimate the distribution of local training time. If the device cannot finish one batch of training within $\frac{1}{10}T_a$ after the start signal, it uploads the training time of the first batch once after it completes the first batch of training. After receiving $\left\{t_{train,b}^i\right\}$ and B^i, the manager calls the *end_time_predictor* to inference the end time of local training t_{end}^i of the i-th device. Then the manager sends t_{end}^i, T_a and B^i to the *training_configuration_scheduler* and gets the adjusted learning rate $\hat{\eta}^i$ and the adjusted number of batches to train $\hat{B}^i$ for the i-th device. Then the manager sends the adjusted training configurations $\hat{\eta}^i$ and $\hat{B}^i$ to the i-th device (④) and the device continues training with the new configurations. With the new number of training steps $\hat{B}^i$, the manager updates the end time of local training t_{end}^i. At $\frac{1}{2}T_a$ after the start of this communication round, the server starts to schedule the ending time of this communication round based on the expected ending time of local training of different devices $\left\{t_{end}^i\right\}$ that have been cached in the manager. The manager sends the cached $\left\{t_{end}^i\right\}$ and T_a to the *synchronization_scheduler*, and the *synchronization_scheduler* outputs the scheduled lasting time of this communication round T_t. The communication between the device and the manager is unblocked, such that the device can upload its updated local model W_t^i (⑤) to the manager once the local training is completed. It is notable that devices are not required to finish training and upload their updated model to the server within one communication round (e.g., the N-th device). At T_t after the start of this communication round, the manager sends the cached set of local updates $\left\{W_t^i\right\}$ and current global model W_t to the *aggregating_module* and gets the global model of the next round W_{t+1}. After getting W_{t+1}, the manager updates T_a and starts the next communication round.

The device (says the j-th) which does not have sufficient resources to train the global model W_t will send a *distillation request* to the manager (②-a) after receiving the start signal of this communication round. Then the manager coordinates the *distillation_module* to generate a smaller model $\hat{W}_t$ which contains the knowledge in W_t and sends $\hat{W}_t$ to device j (②-b). After that, the j-th device conducts local training and interacts with the server as same as the other devices and derives the updated local model $\hat{W}_t^j$. At $\frac{4}{5}T_t$ after the start of this communication round, the server aggregates the uploaded small models and coordinates the *infusion_module* to derive a large model W_t' which contains the knowledge of small models uploaded in this round. W_t' is aggregated into the new global model at the end of this communication round. The server aggregates the small models at $\frac{4}{5}T_t$ after the start of this round rather than T_t because we expect the server to complete infusion before the end of this round. In our platform, the server can train a network more than 5 times faster than the edge devices. Thus, the server can hopefully complete infusion within $\frac{1}{5}T_t$.

4 EVALUATION

4.1 System Implementation

We have implemented FedSEA on 3 NVIDIA Jetson TX2s and 6 Raspberry Pi 4s. The central server is equipped with an Intel Xeon E5-2630@2.6GHz, 128G RAM, and 4 RTX TITAN GPUs. We use TL-SG116 to connect the server and devices. It is hard to evaluate FL applying asynchronous protocols with hundreds of devices using only 9 physical devices. Thus, we collect local training time for multiple batches and the overhead of communication and scheduling on physical devices and conduct simulations using software. To evaluate the performance of FedSEA in realistic settings where the devices conduct some other primary tasks other than FL training and have dynamic resource availability, we divide Jetson TX2 devices into two groups. For one group, the devices switch between the idle condition and the condition of playing 4K videos, which are referred to as fast devices in our system for convenience. For the other group, the TX2 devices conduct real-time object detection with the restriction of at most 5% FPS drop caused by conducting FL training and are referred to as medium devices. The Raspberry Pi 4s which have no GPUs are referred to as slow devices in the system.

4.2 Applications, Datasets, and Models

To demonstrate that FedSEA performs well in different applications, we apply FedSEA to three representative edge AI applications that benefit significantly from FL. The statistics of the datasets that are used in these applications are summarized in Table 4. Even though previous works [37, 38] partition data following non-IID configurations, the overall data distribution of slow devices is the same as the fast devices, which means that abandoning slow devices will not cause a significant accuracy drop. In addition to following the non-IID configurations in [19] to build the non-IID datasets, the overall data distribution is heterogeneous across different levels of on-device resource availability, i.e., slow devices having unique data essential to the global model performance, which is more realistic.

Application#1: Image Classification (IC). Image classification is a popular computer vision application to classify images into categories. With the increasing computation capabilities on devices, image classification applications are widely deployed on edge devices. In this work, we use MNIST, EMNIST [8] and CIFAR10 [18] datasets to develop three image classification applications, i.e., IC-MNIST, IC-CIFAR10 and IC-EMNIST. The models of IC-MNIST and IC-CIFAR10 are the same as in section. 2.2. The model of IC-EMNIST is a CNN with 2 convolutional layers and 2 FC layers. EMNIST is a handwriting image classification dataset grouped by the writers, and hence we naturally distribute one writer's images to one user. In this application, we sample 2000 writers' data and distribute them to users. For MNIST and CIFAR10, each device holds 2-class data and these two classes can be varied across users. For all these three applications, 60% users will conduct local training on fast devices, 20% users train on medium devices and the rest 20% users train on slow devices. To evaluate under the setting of heterogeneous data distributions across different levels of on-device resources, there are some non-overlapped classes between fast, medium, and slow devices. The detailed class distribution is shown in Table 4. The auxiliary dataset of the IC-MNIST and IC-CIFAR10 are SVHN [29] and CIFAR100 [17], respectively. For the IC-EMNIST, we sample additional 400 writers' data as the auxiliary dataset, which are in different data distributions compared with the data on devices.

Table 4: Class distribution of data distributed to devices.

Dataset	Fast devices	Medium devices	Slow devices
MNIST	1-6	7-8	9-10
EMNIST [8]	21-62	11-62	1-62
CIFAR10 [18]	1-6	7-8	9-10
HAR [1]	1-3	1-4	1-6
Shakespeare [27]	40-80	20-80	1-80

Application#2: Human Activity Recognition (HAR). Human activity recognition has become a popular feature for wearable devices using data collected from different types of on-board sensors, such as accelerometers, gyroscopes, etc. This application is developed for recognizing various activities performed by the device owner based on the sensor data. In this work, we use HAR [1] dataset to build this application. HAR collects smartphone accelerometer and gyroscope data from 30 individuals, including six labeled activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying down. We employ a 3-layer fully connected neural network to recognize human activities. We distribute 15 individuals' data to fast devices, 5 individuals' data to medium devices, and 5 individuals' data to slow devices. The class distribution is shown in Tabel 4. The rest 5 individuals' data without labels is used as auxiliary data.

Application#3: Next-Character Prediction (NCP). Next-character prediction is a very practical application on smartphones, e.g., text auto-completion in the virtual keyboard. This application aims to predict what character comes next given the current input. We apply Shakespeare [27] dataset to develop this application. This dataset is built on *The Complete Works of William Shakespeare* by separately extracting different roles' dialogues. In this dataset, the dialogues are distributed to devices according to the speaking role. We build an RNN constructed by an 8-D encoder, including two LSTM layers and three fully connected layers, as the global model for this application. We sample 90 users to train on fast devices, 30 users to train on medium devices, and 30 users to train on slow devices. We also sample data from 30 users as the auxiliary data without labels.

4.3 Experimental Setup

Baselines. To comprehensively evaluate the performance of FedSEA, we compare FedSEA against five baselines:

- **Standalone** trains a model using local data only on each device without collaborations between devices. To make fair comparisons, devices in the standalone method conduct the same epochs of local training as in FedSEA. As there is no global model in Standalone, we average the accuracy of the local models to get the global accuracy.
- **FedAvg** [27] is the most classical synchronous FL method and has been applied to commercial products [3]. Devices communicate updated local parameters to the central server and download the aggregated global model for continuous local training.
- **FedCS** [14] is an efficient FL framework that is aware of resource heterogeneity across devices. FedCS estimates the speed at which devices work and filters out some slow clients proactively (at the stage of client selection) to improve the overall efficiency of FL.

- **FedAsync** [38] is an asynchronous FL method that provides a theoretical convergence guarantee. There is no synchronization time point in FedAsync and the server updates the global model as soon as it receives any uploaded model updates.
- **SAFA** [37] is the state-of-the-art semi-asynchronous FL framework. The system requires a part of the devices to be synchronized with the server every fixed time interval. In each round of synchronization, the system will aggregate the uploaded model updates to update the global model. For a fair comparison, we set the fixed time interval as 0.5×, 1× and 2× of the initial T_a of FedSEA, and get the baseline of SAFA(short), SAFA(medium) and SAFA(long), respectively.

The slow devices (i.e., Raspberry Pi 4) will train smaller models in FedSEA and train the global model the same as the other devices in the baselines. In IC-MNIST and IC-CIFAR10, the slow devices train a 2-layer MLP with a hidden layer of dimension 256. In IC-EMNIST, the slow devices train a small CNN composed of one convolutional layer with 10 kernels and one fully connected layer. In HAR, the slow devices train a multiclass logistic regression model. In NCP, the devices train a small RNN with only one LSTM layer.

Evaluation Metrics. We evaluate the training performance of FedSEA using two sets of metrics:

- **Metrics for Training Performance:** (1) *inference accuracy:* for IC-MNIST and IC-CIFAR10, we evaluate the inference accuracy of the global model on the global test data. For IC-EMNIST, HAR and NCP, we evaluate the inference accuracy of the global model on each device's test data and report the average accuracy for evaluations; (2) *time cost:*for a fair comparison, we fix the communication cost (i,e, data volume of communication) of the whole system and measure the time cost of the system to finish training, and normalize it as the ratio to the time cost of FedAvg as reported time cost;
- **Metrics for On-board Resource Cost:** for a fair comparison, FedSEA and baselines will stop training after consuming an identical communication cost. We evaluate the cost of various on-board resources for federated training: (1) *local training time:* we measure the time cost for local training performed on devices during the whole federated training and normalize it as the ratio to the local training time of FedAvg. We report the normalized training time as the local training time; (2) *energy consumption:* we measure the average energy consumption across devices brought by participating in FL and calculate the energy consumption saving percentage; (3) *memory footprint:* we measure the memory footprint of different applications on device and calculate the memory footprint reduction on the extremely lagging devices during training.

4.4 Training performance

We compare FedSEA with the baselines in terms of the accuracy-time cost tradeoff. For a fair comparison, we set the communication cost (i,e, transmitted data volume) to be the cost of 2000 rounds of FedAvg for FedSEA and baselines. Ideally, we expect the FL system finishes training in a shorter time with higher accuracy.

We first compare FedSEA with the SOTA semi-asynchronous FL algorithm SAFA. It is shown that by setting longer synchronization intervals, SAFA can achieve higher accuracy but needs to take

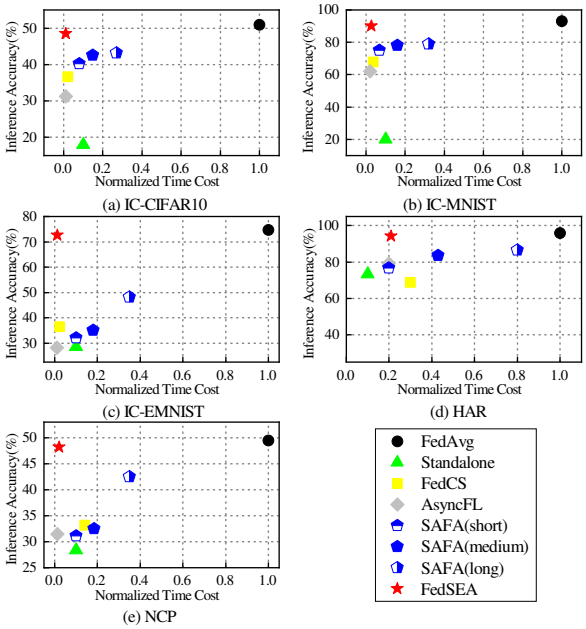

Figure 7: Comparison between FedSEA and baselines in inference accuracy-time cost space.

longer time to finish training, which is more like a zero-sum game between performance and efficiency. However, FedSEA is able to improve inference accuracy and system efficiency simultaneously compared with SAFA. In particular, compared with SAFA applying medium synchronization interval, FedSEA improves inference accuracy by 11.9%, 5.97%, 37.44%, 17.6%, and 15.7% on IC-MNIST, IC-CIFAR10, IC-EMNIST, HAR, and NCP, respectively. Besides, it also reduces 8.62×, 12.5×, 15×, 2.05× and 8.81× time cost in those applications, respectively.

Second, compared with FedAsync which focuses on time efficiency, FedSEA achieves higher inference accuracy while having comparable time cost. Specifically, FedSEA improves inference accuracy by 27.9%, 17.3%, 44.34%, 15.13%, and 16.7% on IC-MNIST, IC-CIFAR10, IC-EMNIST, HAR, and NCP, respectively.

Third, compared with FedCS, FedSEA improves inference accuracy and system efficiency simultaneously. The key reason is that FedCS does not consider the data distribution heterogeneity across devices when sampling devices to participate in training. In particular, FedSEA improves inference accuracy by 21.9%, 11.8%, 36.04%, 25.5%, and 16.3% on IC-MNIST, IC-CIFAR10, IC-EMNIST, HAR, and NCP, respectively. In addition, FedSEA also reduces 1.27×, 1.75×, 2.01×, 1.43× and 6.74× time cost, respectively.

Fourth, compared to FedAvg which is the vanilla synchronous FL, FedSEA does not outperform in inference accuracy since FedAvg does not consider the time efficiency. However, FedSEA can achieve a significant reduction in time cost compared with FedAvg. Specifically, FedSEA reduces 72.3×, 83.3×, 87.2×, 4.76× and 34.7× time cost on IC-MNIST, IC-CIFAR10, IC-EMNIST, HAR, and NCP, respectively. It is also notable that there are some realistic scenarios where slow devices do not have the capability to train the global model. In this case, FedSEA can still achieve high inference accuracy by letting the slow devices train smaller models, but FedAvg

will suffer a serious accuracy drop due to the exclusion of those extremely lagging devices.

For standalone, although the devices do not need to communicate with the server nor wait for the other devices for training, it is still time-consuming for slow devices to train the global model. Unsurprisingly, without collaborations across devices, the local models trained locally perform poorly on the global dataset.

Scheduling overhead. The only scheduling overhead that would affect the efficiency of the whole system is the scheduler of synchronization since the other scheduling actions are conducted in parallel with local training. To schedule the synchronization, the server only needs to sort an array of numbers and compute the difference between the elements, which is significantly efficient. For example, the local training time of one batch on an idle Jetson TX2 is nearly 0.2 seconds for IC-CIFAR10. However, the time cost of the server in our platform to schedule even 10,000 devices, which are much more than the devices in our evaluation, is lower than 0.007 seconds and not comparable with one step of local training. Thus, the impact of scheduling overhead on the efficiency of the system is negligible.

4.5 Hyper-parameter Evaluation

Number of Participating Devices: In each round of synchronization, FedSEA randomly selects several idle devices to participate in training. We evaluate the impact of the number of selected devices in each synchronization on the performance. We conduct experiments on IC-EMNIST, IC-CIFAR10 and NCP, and vary the number of selected devices in each round as {20, 40, 60}. Figure 8 shows that the inference accuracy increases slightly when the number of selected devices increases. Specifically, the inference accuracy increases by 2.47%, 1.1% and 2.6% for IC-EMNIST, IC-CIFAR10 and NCP when the number of selected devices increases from 20 to 60.

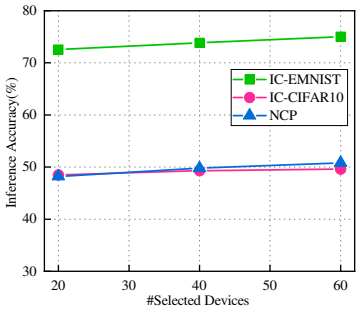

Figure 8: Impact of the number of selected devices in each round of synchronization on the inference accuracy.

Configuration Scheduler Hyper-parameter α: An important hyper-parameter of FedSEA is the tolerance of staleness, which is α in the algorithm of *training_config_scheduler*. We conduct experiments on IC-EMNIST, IC-CIFAR10, NCP and vary α as {2, 4, 6, 8, 10} to explore the impact of α on the performance of FedSEA. The results are shown in Figure 9. The results illustrate that reducing α improves the inference accuracy of FedSEA. The reason for accuracy improvement is that *training_config_scheduler* can mitigate the staleness and improve the balance of aggregating frequency across devices by applying a smaller α, which is consistent with our Theorem 1. It is also notable that when decreasing α extremely (i,e, from 4 to 2), the inference accuracy shows a slight drop. The accuracy drop after applying extremely small α comes from the significant manipulation of local training configurations. However, this accuracy drop is marginal because the *distillation&infusion_module* avoids extremely significant manipulation on local training configurations, and this marginal accuracy drop will not be an obstacle to applying FedSEA in the real world.

Table 5: The reduction of local training time.

Applications	Reduction of local training time	
	Jetson TX2	Raspberry Pi 4
IC-MNIST	2.1×	714.5×
IC-CIFAR10	2.7×	792.9×
IC-EMNIST	3.1×	13.4×
HAR	1.6×	4.9×
NCP	2.3×	12.7×

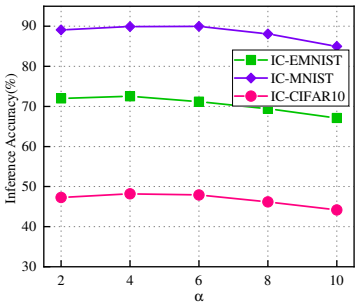

Figure 9: Impact of the tolerance of staleness α on the inference accuracy.

4.6 On-board Resource Cost

Local Training Time: One key benefit of applying FedSEA is that devices can reduce the local training time because the slow devices train fewer steps in each round or conduct local training based on a smaller model. This reduction of training time will promote the application of FL systems in practice since the local training of FL will hinder the performance of the primary task on edge devices such as real-time object detection. To quantify the benefit of local training time reduction, we compare the average training time of Jetson TX2s with the primary task of objective detection and Raspberry Pi 4s in FedSEA with the corresponding devices in FedAvg with the same communication cost. Table 5 shows that applying FedSEA devices can reduce local training time significantly, especially for the extremely lagging devices (i,e, Raspberry Pi 4s). For example, Raspberry Pi 4 can reduce local training time by 714.5× and 792.9× in IC-MNIST and IC-CIFAR10, respectively. The reason for such a huge reduction is that Raspberry Pi 4s in FedSEA just train 2-layer MLP rather than CNN in IC-MNIST, IC-CIFAR10.

Energy Consumption: FedSEA can reduce the energy consumption of devices by decreasing the local training steps and letting slow devices train smaller models. We measure the average energy consumption of a Raspberry Pi 4 to complete one round of local training

Table 6: Memory footprint reduction of FedSEA.

Applications	Memory footprint (MB)	
	FedSEA	Baselines
IC-MNIST	1.01	4.85
IC-CIFAR10	3.98	7.47
IC-EMNIST	6.59	14.19
HAR	0.32	0.72

and compare the results between FedSEA and the other baselines in Figure 10. The results demonstrate that devices in FedSEA can complete one round of local training with less than 3 mWh energy consumption, which saves more than 700× energy compared with other baselines in IC-MNIST and IC-CIFAR10.

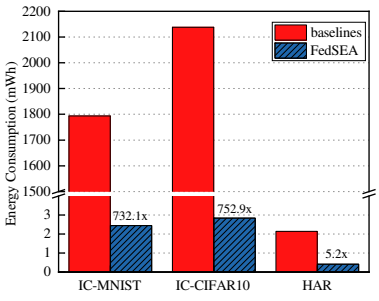

Figure 10: Comparison between FedSEA and the baselines on energy consumption (Raspberry Pi 4).

Memory Footprint: FedSEA is able to dramatically reduce the memory footprint of local training on slow devices by allowing them to train smaller models. To quantify the benefit of memory footprint reduction, we set the batch size as 64 and measure the memory needed by a Raspberry Pi 4 to conduct local training in different applications. For the baselines, devices conduct local training on the same model architecture, and hence the reported result is the same for all the baselines. As Table 6 shows, FedSEA is able to reduce memory footprint by 4.9×, 1.9×, 2.2×, and 2.3× in IC-MNIST, IC-CIFAR10, IC-EMNIST, and HAR, respectively. This reduction in memory footprint will be even more significant when applying larger batch sizes or larger global model sizes in the real world.

5 DISCUSSION

Extra Communication Cost. It is notable that FedSEA introduces extra communications between the server and the local devices. However, these extra communications are only for control information exchange. The communication energy and time for control information are not comparable with the exchange of model parameters. In addition, the exchange of control information happens in parallel with local training and will not affect the efficiency of the system.

Client Selection. There have been many works [2, 7, 12] discussing the improve or client selection in FL. Most of the works focus on solving the non-IID problem to improve or accelerate the convergence of the global model. A fairness-guaranteed algorithm termed RBCS-F [16] was proposed to model the fairness-guaranteed client selection as a Lyapunov optimization problem. It is an interesting topic that we apply active client selection to FedSEA and improve the efficiency of the system further.

6 RELATED WORK

Asynchronous and Semi-asynchronous FL. To improve the efficiency of FL systems, asynchronous and semi-asynchronous FL are proposed. However, existing AFL and SAFL systems cannot guarantee efficiency in realistic settings and suffer from the accuracy drop of the global model. One major challenge of AFL and SAFL systems is how to mitigate the straggling problem caused by the stale models. Current AFL and SAFL frameworks use two strategies to solve the straggling problem: (1) Stale model updates are set smaller weights during aggregation [6, 24, 31, 34, 38, 40]; (2) Extremely stale model updates are abandoned by the server [14, 26, 30, 37]. Both methods cause a serious accuracy drop under non-IID settings where slow devices hold unique and important data. The other challenge is to improve the efficiency of the FL system in reality. An effective scheduler of synchronization is the key to improving the efficiency of SAFL. Previous works either set fixed time intervals for synchronization [34, 37] or apply a cache for model updates [26, 40] which is fully filed before aggregation. Both schemes cannot guarantee efficiency when the devices have dynamic resource availability.

Knowledge Distillation. Knowledge distillation [15] was proposed to reduce the model size and boost the training of small models. It is based on the observation that when you conduct KD from a teacher model to a student model, the convergence of the student model outperforms directly training it. Our distillation and infusion modules differ from conventional KD in two aspects: (1) Conventional KD requires the original training dataset with labels, while in FedSEA the server only has an unlabeled public dataset that is non-IID with the data on devices; (2) Conventional KD distill knowledge from a large model to a small model, while the infusion module in our system infuses the knowledge from a small model to a large model.

7 CONCLUSION

In this paper, we present the design, implementation and evaluation of FedSEA, an efficient semi-asynchronous FL framework for extremely heterogeneous devices. By applying FedSEA, the central server can adjust the training configurations of participating devices such that the problems of staleness and unbalanced aggregation frequency are mitigated, and hence the performance of the global model is improved. The synchronization strategy of FedSEA is adaptive to the predicted training time of devices and the dynamic on-device resource availability, which significantly improves the system's efficiency. FedSEA also enables the extremely lagging device to participate in training, which improves the scalability of FL systems. We evaluate FedSEA using three representative FL applications. The results demonstrate that FedSEA significantly outperforms the state-of-the-art methods in accuracy, time efficiency, local training time, energy consumption, and memory footprint. FedSEA improves the efficiency of FL systems and represents a significant step towards the deployment of efficient FL systems in real life.

ACKNOWLEDGEMENT

This work is supported in part by NSF 1822085, IUCRC for ASIC and the memberships contributed by Lenovo, etc.

REFERENCES

[1] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3.

[2] Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff Bilmes. 2021. Diverse client selection for federated learning: Submodularity and convergence analysis. In *ICML 2021 International Workshop on Federated Learning for User Privacy and Data Confidentiality*.

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[4] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081* (2020).

[5] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 15–24.

[6] Yang Chen, Xiaoyan Sun, and Yaochu Jin. 2019. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems* 31, 10 (2019), 4229–4238.

[7] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. 2020. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243* (2020).

[8] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2921–2926.

[9] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[10] Zhixu Du, Jingwei Sun, Ang Li, Pin-Yu Chen, Jianyi Zhang, Hai Li, Yiran Chen, et al. 2022. Rethinking Normalization Methods in Federated Learning. *arXiv preprint arXiv:2210.03277* (2022).

[11] Biyi Fang, Jillian Co, and Mi Zhang. 2017. Deepasl: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[12] Jack Goetz, Kshitiz Malik, Duc Bui, Seungwhan Moon, Honglei Liu, and Anuj Kumar. 2019. Active federated learning. *arXiv preprint arXiv:1909.12641* (2019).

[13] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.

[14] Jiangshan Hao, Yanchao Zhao, and Jiale Zhang. 2020. Time efficient federated learning with semi-asynchronous communication. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 156–163.

[15] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[16] Tiansheng Huang, Weiwei Lin, Wentai Wu, Ligang He, Keqin Li, and Albert Y Zomaya. 2020. An efficiency-boosting client selection scheme for federated learning with fairness guarantee. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1552–1564.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* 55 (2014).

[19] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2020. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371* (2020).

[20] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 42–55.

[21] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the Convergence of FedAvg on Non-IID Data. In *International Conference on Learning Representations*.

[22] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[23] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2018. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August* 15, 2018 (2018), 11.

[24] Xiaofeng Lu, Yuying Liao, Pietro Lio, and Pan Hui. 2020. Privacy-preserving asynchronous federated learning mechanism for edge network computing. *IEEE Access* 8 (2020), 48970–48981.

[25] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. 2019. Differentially private asynchronous federated learning for mobile edge computing in urban informatics. *IEEE Transactions on Industrial Informatics* 16, 3 (2019), 2134–2143.

[26] Qianpiao Ma, Yang Xu, Hongli Xu, Zhida Jiang, Liusheng Huang, and He Huang. 2021. FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing. *IEEE Journal on Selected Areas in Communications* 39, 12 (2021), 3654–3672.

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

[28] Yuchen Mu, Navneet Garg, and Tharmalingam Ratnarajah. 2022. Communication-Efficient Federated Learning For Massive MIMO Systems. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 578–583.

[29] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[30] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 1–7.

[31] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. 2021. Sageflow: Robust federated learning against both stragglers and adversaries. *Advances in Neural Information Processing Systems* 34 (2021), 840–851.

[32] Mary Phuong and Christoph Lampert. 2019. Towards understanding knowledge distillation. In *International Conference on Machine Learning*. PMLR, 5142–5151.

[33] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1421–1429.

[34] Dimitris Stripelis and José Luis Ambite. 2021. Semi-synchronous federated learning. *arXiv preprint arXiv:2102.02849* (2021).

[35] Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. 2021. Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective. *Advances in Neural Information Processing Systems* 34 (2021), 12613–12624.

[36] Jingwei Sun, Ang Li, Binghui Wang, Huanrui Yang, Hai Li, and Yiran Chen. 2021. Soteria: Provable defense against privacy leakage in federated learning from representation perspective. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9311–9319.

[37] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis. 2020. SAFA: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Trans. Comput.* 70, 5 (2020), 655–668.

[38] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[39] Xiao Zeng, Kai Cao, and Mi Zhang. 2017. MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 56–67.

[40] Chendi Zhou, Hao Tian, Hong Zhang, Jin Zhang, Mianxiong Dong, and Juncheng Jia. 2021. TEA-fed: time-efficient asynchronous federated learning for edge computing. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 30–37.