Towards Efficient Horizontal Federated Learning

by

Wentai Wu

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

September 2021

Abstract

The advance of Machine Learning (ML) techniques has become the driving force in the development of Artificial Intelligence (AI) applications. However, it also brings about new problems as ML models and algorithms are more data-hungry than ever. First, the sources of data are often geographically distributed and thus gathering all the data for centralised training incurs heavy traffic and prohibitive communication costs. Second and more importantly, moving data out of local devices are now prohibited in many situations due to privacy concerns. As a promising solution, Federated Learning (FL) is a framework proposed by Google for learning from decentralised data without data sharing. Nonetheless, many challenges remain especially regarding how to facilitate efficient FL in different circumstances. The work presented in this thesis is mainly focused on optimising the efficiency of Horizontal Federated Learning, the most commonly used paradigm of FL, from three perspectives: i) redesigning the device–server synchronisation mechanism of FL for failure-tolerant collaborative training over unreliable, heterogeneous end devices (clients), ii) integrating FL with the Mobile Edge Computing (MEC) architecture to utilise the edge layer for robustness, resource efficiency and fast convergence, and iii) optimising the client selection policy in FL through the evaluation of clients' training value based on the learnt representations of their associated data. Results of extensive experiments are presented to demonstrate the effectiveness of the proposed approaches. In addition to algorithm design and experimental observations, this thesis also discusses the research spectrum of FL and overviews its development in the industry including popular platforms, systems and emerging applications. Based on the work presented, key conclusions and insights are provided with a research outlook for the future study of FL.

Acknowledgments

First and foremost, I would like to thank my parents for their endless love and support that have always been the most valuable assets of my life. This thesis is dedicated to my parents.

I would also like to show the sincerest gratitude to my supervisor, Dr. Ligang He, who has been leading my way through years of study that was filled with challenges and hard work. I believe working hard is important, but having the right guidance is pivotal. My supervisor is a great role model who has his unique influences on every student in the lab and that also helps to foster a not only positive but also progressive circumstance throughout the years of my study.

I also want to thank my labmates and friends as I feel very fortunate to share the time with all of you. Life is a marathon. From time to time we all need someone whom we can learn from, make progress with, and show the smiles to. The special circumstances recently have changed everyone's life but at the same time, mirrored the value of friendship. I would also like to appreciate the excellent work by the staff in my department, especially Sharon and Roger, who have been considerate and helped me out of many problems and difficulties. Thank you all.

Lastly, I feel grateful as I believe these years was a turning point of my life. Life is not a linear function of time. It is easy to get obsessed and forget to look into the mirror to see if we have turned into the kind of person we despised most. A certain mentality is crucial to keep one on the right track. Through these years of study, I have learnt knowledge and much more beyond knowledge. The motto remains my favourite: *Mind over matter* (*Mens agitat molem* in Latin).

Declarations

Publications

All of the content in this thesis is composed by the author in accordance with the regulations for the degree of Doctor of Philosophy. The work described in this thesis has been undertaken by the author unless otherwise stated. The research presented in this thesis is based on the following work:

- Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A Jarvis. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70(5):655–668, 2021. [1]
- Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1539–1551, 2021. [2]
- Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Fedprof: Selective federated learning with representation profiling. *arXiv preprint arXiv:2102.01733. Under review, ICLR'22*, 2021. [3]
- Wentai Wu, Ligang He, and Weiwei Lin. Variation-incentive loss re-weighting for regression analysis on biased data. *arXiv preprint arXiv:2109.06565.*, 2021. [4]

Research by the author during the period of registration also led to the following publications:

- Wentai Wu, Ligang He, Weiwei Lin, Yi Su, Yuhua Cui, Carsten Maple, and Stephen A Jarvis. Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality. *IEEE Transactions on Knowledge and Data Engineering. DOI: 10.1109/TKDE.2020.3035685*, 2020. [5]
- Wentai Wu, Weiwei Lin, Ligang He, Guangxin Wu, and Ching-Hsien Hsu. A power consumption model for cloud servers based on elman neural network. *IEEE Transactions on Cloud Computing*, 9(4):1268–1277, 2021. [6]

- Weiwei Lin, Wentai Wu, and Ligang He. An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers. *IEEE Transactions on Services Computing*, 2019. [7]

- Tiansheng Huang, Weiwei Lin, Wentai Wu, Ligang He, Keqin Li, and Albert Zomaya. An efficiency-boosting client selection scheme for federated learning with fairness guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 2020. [8]

- Weiwei Lin, Fang Shi, Wentai Wu, Keqin Li, Guangxin Wu, and Al-Alas Mohammed. A taxonomy and survey of power models and power modeling for cloud servers. *ACM Computing Surveys (CSUR)*, 53(5):1–41, 2020. [9]

- Weiwei Lin, Wentai Wu, and Keqin Li. Energy-saving technologies of servers in data centers. In Hwaiyu Geng, editor, *Data Center Handbook: Plan, Design, Build, and Operations of a Smart Data Center*, chapter 19, pages 337–347. John Wiley & Sons, New Jersey, 2021. [10]

Sponsorships and Grants

The research presented in this thesis was made possible by the support of the following funding sources:

- Warwick China Scholarship (2018–2021)

- China Scholarship Council (CSC) (2018–2022)

Contents

List of Figures

List of Tables

List of Notations

X	the feature (vector) of a sample
Y	the label (vector) of a sample
(X, Y)	a random data sample
(X_i, Y_i)	the i-th sample in some dataset
η	learning rate
∇	gradient
∂	partial derivative
χ	feature space
$\mathcal{Y}$	target/label space
$D/\mathcal{D}$	the joint (full) dataset/the population distribution
$D_k/\mathcal{D}_k$	local dataset/local data distribution for client k
D^*	benchmark/validation dataset
w	(the parameter vector of) the global model
w_k	the local model on client k
F	objective function
ℓ	loss function
$\mathcal{N}$	Gaussian distribution
$\mathbb{E}[\cdot]$	expectation
$\xrightarrow{d}$	converges in distribution
KL	Kullback–Leibler divergence
$[[\cdot]]$	ciphertext under Homomorphic Encryption
$\mathbb{R}^n$	n-dimensional real value space
$\lvert \cdot \rvert$	absolute value of/size of
$\lvert\lvert \cdot \rvert\rvert_2$	Euclidean norm

Acronyms

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
AsyncSGD	Asynchronous Stochastic Gradient Descent.
BCE	Binary Cross Entropy.
BPS	Bits Per Sample.
BS	Base Station.
CFCFM	Compensatory First Come First Merge.
CLT	Central Limit Theorem.
CPB	Cycles Per Bit.
C-S	Client-Server.
CE	Cross Entropy.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CV	Computer Vision.
DC	Data Center.
DNN	Deep Neural Network.
DP	Differential Privacy.
EUR	Effective Update Ratio.
FC	Fully Connected.

FedAvg	Federated Averaging.
FFN	Feed-Forward Network.
FL	Federated Learning.
FTL	Federated Transfer Learning.
GD	Gradient Descent.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit.
HE	Homomorphic Encryption.
HFL	Horizontal Federated Learning.
IID	Independent and Identically Distributed.
IoT	Internet of Things.
KL	Kullback–Leibler.
LR	Learning Rate.
LSE	Least Square Estimation.
LSTM	Long Short-Term Memory.
MAE	Mean Absolute Error.
MEC	Mobile Edge Computing.
ML	Machine Learning.
MLP	Multi-Layer Perceptron.
MPE	Mean Percentage Error.
MSE	Mean Square Error.
NLP	Natural Language Processing.
P2P	Peer-to-Peer.
PS	Parameter Server.
RL	Reinforcement Learning.

RNN Recurrent Neural Network.

RSF Regional Slack Factor.

SAFA Semi-Asynchronous Federated Averaging.

SGD Stochastic Gradient Descent.

SNR Signal-to-Noise Ratio.

SR Synchronization Ratio.

SyncSGD Synchronous Stochastic Gradient Descent.

TPU Tensor Processing Unit.

VFL Vertical Federated Learning.

Wh Watt hour.

Chapter 1

Introduction

The exploration by computer scientists in the area of Artificial Intelligence (AI) has never stopped ever since the last century. Even though there is still a lot of ground to cover before human-level AI can be truly realised [22], researchers have made and are still making rapid advances in the development of Machine Learning (ML) techniques that empower a variety of AI applications that we are using today. A prominent trend for the evolution of AI-driven technologies is the constant increase of complexity in terms of the models we built to excel in specific ML tasks. This is, in a large part, a result of the paradigm shift from the rule-based reasoning systems (e.g., expert systems [23]) to the soft computing-based, connectionist models like Deep Neural Networks (DNNs) [24] whilst the paradigm shift, to a great extent, stems from the increasing availability of data. From the Convolutional Neural Networks (CNNs) [25] for Computer Vision (CV) tasks and the Recurrent Neural Networks (RNNs) [26] for Natural Language Processing (NLP) tasks to the emerging cross-domain models like Transformers [27], deep models coupled with large/huge datasets (e.g., ImageNet [28] and C4Corpus [29]) have made the rule of thumb for the recent success in a wide range of applications such as object detection [30–32], image generation [33–35] and neural machine translation [36–38].

Machine learning is always data-driven. The more powerful the AI applications we desire, the more complex the models need to be. Consequently, more data are required to feed the models during training. This was not a problem when we built our AI applications and trained the underlying models using *centralised* computing resources (e.g., servers in a data center), in which case we first collected a sufficient amount of data, stored the data in a central storage and trained our model(s) on this complete set of data. To better utilise hardware capacity, distributed model training methods (e.g., [39–43]) have also been proposed to facilitate parallel training. However, both centralised training and traditional distributed training methods are mainly designed for performance-oriented scenarios where the training is performed on high-spec facilities (typically servers and GPU/TPU clusters) with high-speed network connections between data holders and the parameter server. The training data are often directly or indirectly accessible to all the workers (i.e., machines that run the training logic) for the purpose of faster model convergence.

With sources of data moving towards the network edge and the rising concerns of data privacy, learning from decentralised data over general-purpose devices (e.g., mobile phones, smart sensors and wearable devices) has been a topic of interest and shown great practical value. For example, building an accurate keyboard input prediction model needs to make use of many users' typing data which should be kept private on their personal mobile phones [12, 15]. Apart from privacy protection, communication efficiency is also an important factor in practical application scenarios including industrial control, autonomous vehicles and the Internet of Things (IoT). It is predicted that the data generated by IoT devices will account for 75% of the total volume in 2025 [44]. End devices, such as sensors and mobile phones, have limited performance and bandwidth and are typically networked through noisy channels. Under these circumstances, centralised and traditional distributed training methods may no longer be feasible due to the following outstanding issues:

1. **Prohibitive communication cost**: the transmission of data (for centralised training) or frequent exchange of gradients[1] and models (for distributed training) incurs heavy traffic over the network connections, resulting in high communication latency and expenses (end devices may use charged networks);

2. **Non-IID and imbalanced data**: the data are not evenly spread and identically distributed across end devices, where local datasets may differ a lot from each other in data distribution, resulting in potentially biased updates with traditional distributed training methods;

3. **Data privacy concerns**: moving user data out of their local devices has been restricted by law in many countries (e.g., the GDPR[2] enforced in EU) whilst

[1]In the domain of machine learning, the term gradient refers to the partial derivative of the objective function (which we aim to optimise) with regard to the model's parameters (e.g., neural networks' weight vectors) over one or a set of data samples.

[2]General Data Protection Regulation. `https://gdpr-info.eu/`

frequent upload of gradients also risks exposing the raw training data [45, 46].

Federated Learning (FL) [47], originally proposed by Google, has emerged as a promising framework for learning from decentralised data in a privacy-preserving manner. The standard implement of FL [12] adopts the conventional parameter server (PS) architecture where end devices (clients) are connected to the central coordinator in a star topology. The coordinator can be a central server or a base station at the network edge. Without creating ambiguity, the central coordinator is also referred to as *the server* in the remainder of this thesis. Compared to traditional distributed learning methods, the key features of FL are i) **no data sharing** and the server stays agnostic of any training data, ii) **exchange of encrypted models** instead of exposing gradients, and iii) **sparing device–server communications** instead of batch-wise gradient uploads. Fig. 1.1 gives a schematic overview of the standard FL framework, which is also termed *Horizontal Federated Learning* since different paradigms have been developed under similar frameworks [48]. The figure shows a typical cross-device FL scenario [49] where a multitude of user devices (e.g., mobile phones and smart sensors) are coordinated by a central server (at the edge or on the cloud). The devices contribute by performing local training on their own data without sharing the data itself. The work presented in this thesis is focused on *Horizontal FL* because it well aligns with the aim of this research — learning from decentralised data at scale.

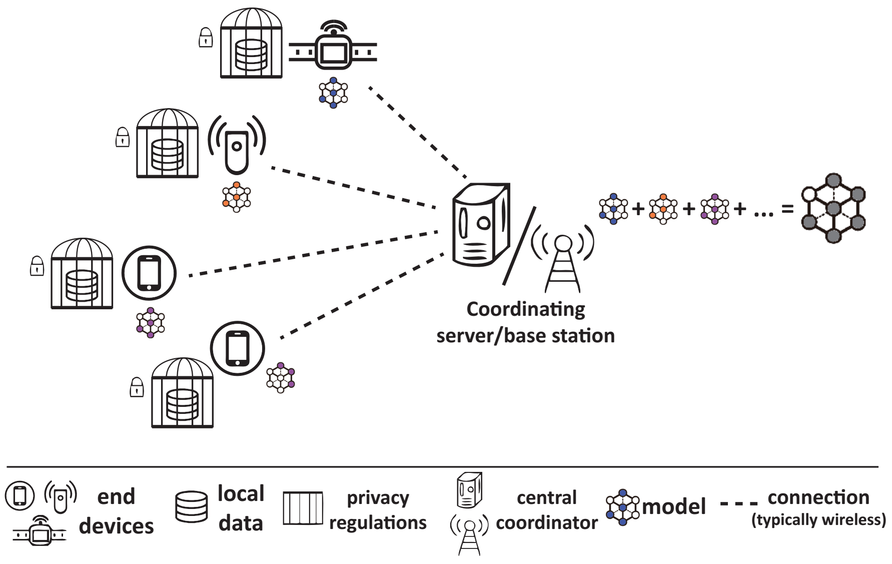


Figure 1.1: An overview of the (Horizontal) Federated Learning framework.

Horizontal federated learning was designed for training on distributed data sets that typically **share the same feature space whilst having little or no overlap**

in terms of their data instances. The term *horizontal* describes these properties from the perspective of tabular data — one sample per row and the columns stand for their features/attributes (Fig. 1.2). In September 2020, the Standard 3652.1-2020 for Federated Machine Learning (which is a synonym for federated learning) was approved by the C/AISC (Artificial Intelligence Standards Committee) of IEEE and published as an official *IEEE Guide for Architectural Framework and Application of Federated Machine Learning* in early 2021. According to the definition in the standard[3], 'Horizontal FML refers to building a model in the scenario where data sets have significant overlaps on the feature spaces $(X_1, X_2, \ldots)$ but not on the ID spaces. In this case, an FML model can be built as if the data is split and join horizontally.' An illustration (Fig. 1.2) is provided in the guidebook.

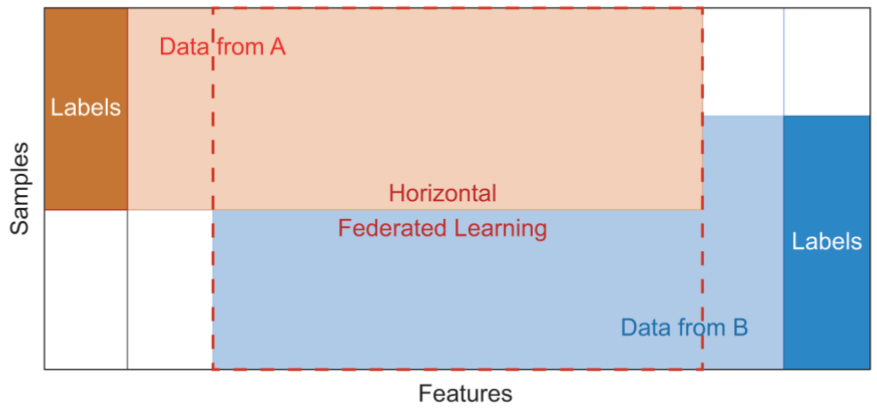


Figure 1.2: Horizontal federated learning from the perspective of data. (Source: IEEE Standard 3652.1-2020)

Despite the rigorous definition given by the IEEE standard, most of the recent studies (including the research by the author) on horizontal FL assume an *identical feature space* for all the local data partitions so that all the participants in the system are working on models of the same shape. Something that is not revealed in the data view (Fig. 1.2) is the specific distribution of data, which separates the settings of horizontal FL and conventional distributed learning. The distinction can be explained using the notion of domains. Let D denote the full set of data which actually corresponds to a domain that can be defined as a two-element tuple $(\chi, P(X))$ where $X = \{x_1, x_2, \ldots\}$ is any random sample from D, χ is the feature space and $P(X)$ is the marginal distribution of samples. Similarly, the local data D_k can be represented as a local domain $(\chi_k, P_k(X))$. With these notations, traditional distributed algorithms

[3]IEEE Standard 3652.1-2020. `https://standards.ieee.org/standard/3652_1-2020.html`

typically work on top of the following assumption for all the clients:

$$\chi_k = \chi, P_k(X) = P(X),$$

which means identical feature space and identical distribution of data.

By contrast, research on horizontal FL is usually developed on the following setting:

$$\chi_k = \chi, P_k(X) \neq P(X),$$

which means identical feature space but non-IID data in different local datasets.

The most representative use case for horizontal FL is `GBoard`, the Google keyboard, which is claimed to have been optimised over a global scale of user population (1.5 million clients in North America) through thousands of training rounds [15]. As shown in Fig. 1.3, users of the virtual keyboard do not share their text typed in but allow the programme to train the underlying model on their data. Some global configuration (e.g., dictionary size and embedding dimensions) will apply to ensure an identical feature space for all the local models so that the cloud can aggregate them using weighted average.

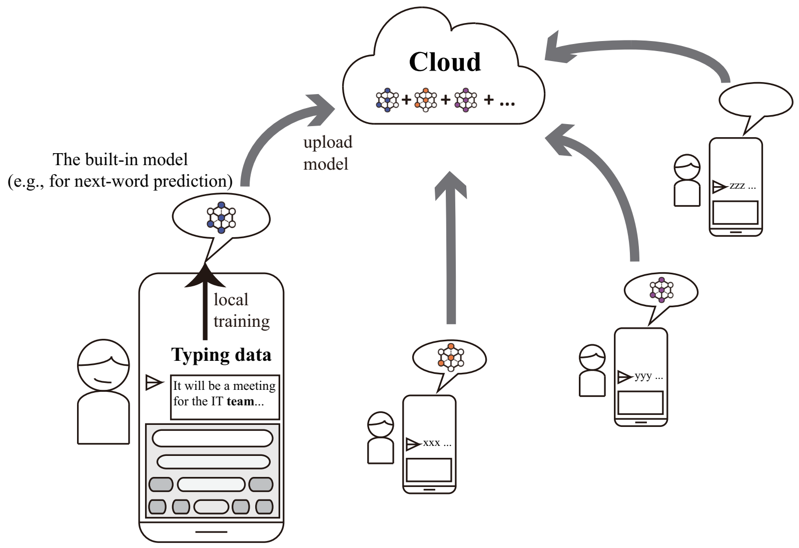

Cloud
+ + + ...
upload model
The built-in model (e.g., for next-word prediction)
local training
Typing data
It will be a meeting for the IT team...
xxx ...
yyy ...
zzz ...

Figure 1.3: Collaborative model training for virtual keyboards using horizontal FL.

In the following sections, the problem of 'Data Islands' in distributed machine learning will be discussed first, followed by a review of the emergence and evolution of the federated learning framework and its typical workflow. Then the motivation behind the research presented in this thesis is elaborated, followed by the definition of efficiency in the context of FL. A preview of contributions is given next about the research to be presented in this thesis including i) a failure-tolerant FL algorithm over heterogeneous devices, ii) a hierarchical FL algorithm for MEC systems, and iii) an optimised FL algorithm for data of varied quality. An outline of this thesis is provided in Section 1.7 to conclude this chapter.

1.1 Data Islands

There are two main factors that propel the rapid advances of AI technology: computing power and data. As pointed out in a recent study by the Brain team of Google research, 'the availability of larger datasets coupled with increased computational capacity often leads to a paradigm shift' (Tolstikhin et al. [50]), where the 'paradigm shift' refers to the development of large and complex machine learning models. In the meantime, networks have been prominently used to connect the computer resources and by scaling out with proper networking and parallelising techniques we can always obtain an equivalent computational capacity and the right results as a single supercomputer can provide. However, this is not the case for data in the context of machine learning. By connecting the sources of data we cannot expect to have an equivalent of the locally-stored complete set of data, which consequently leads to 'data islands' (Fig. 1.4). The reasons are two-fold. On the one hand, we have seen more and more restrictions on the storage and management of data as user privacy issues are drawing increasing attention. This makes gathering data for centralised training infeasible when transferring data out of user devices is prohibited or not authorised. For example, the General Data Protection Regulation (GDPR) enforces many restrictions on data transfer.

> *Any transfer of personal data which are undergoing processing or are intended for processing after transfer to a third country or to an international organisation shall take place only if, subject to the other provisions of this Regulation, the conditions laid down in this Chapter are complied with by the controller and processor, including for onward transfers of personal data from the third country or an international organisation to another third country or to another international organisation. All provisions in this Chapter shall be applied in order to ensure that the level of protection of natural persons guaranteed by this Regulation is not undermined.* (General

Principle for Transfers, Article 44, Chapter 5, GDPR[4])

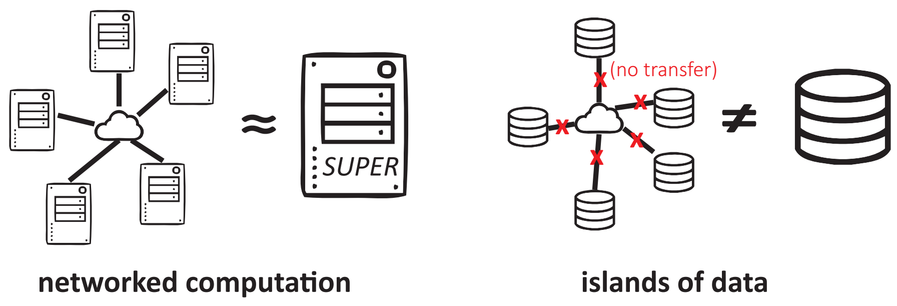


Figure 1.4: Data islands cannot be addressed by just networking.

On the other hand, there is a gap between learning from decentralised data and learning from a single, complete set of data using gradient-based methods (which is the basis of model training methods widely used in ML and DL [43]). This is naturally caused by the way how gradients are calculated. As a common hypothesis, this gap can be narrowed by algorithm design which, however, typically incurs the increase of communication costs. Nonetheless, in many scenarios, the gain (in approximating centralised training) may be marginal whilst the cost (of more frequent communication) could be prohibitive.

One can think of the problem from an opposite point of view — is it possible for the communication costs to be significantly reduced at the expense of a compromised guarantee on the gradients' quality? **The answer is positive for a wide range of realistic scenarios where the cost of communication dominates the gain of frequent information exchange.** This is one of the key ideas behind federated learning. A theoretical analysis will be provided in Chapter 2 (Section 2.1.2) for a better understanding of the problems caused by 'data islands'.

1.2 Emergence and Evolution of Federated Learning

Federated Learning (FL) was originally proposed by Google as a privacy-preserving distributed machine learning paradigm over decentralised data [47]. In the paper the authors first define *Federated Optimisation* as the implicit problem behind FL. In contrast to the traditional distributed optimisation problems, federated optimisation puts more restrictions on the data as well as the resources. Its key properties (or the key settings in FL) include:

[4]Art. 44 GDPR. `https://gdpr-info.eu/art-44-gdpr/`

- **No data sharing:** each local dataset is only accessible to the corresponding local device and data sharing is prohibited between any parties, i.e., 'data islands'.
- **Massive scale**: the data are distributed across a large number of devices and thus the optimisation needs to be performed at scale.
- **Imbalanced data sizes**: local datasets on the training devices are of different sizes because the devices collect or generate data independently and autonomously.
- **Statistical heterogeneity**: the data are typically non-IID across the devices since different devices are owned by different users who display different use patterns.
- **Device heterogeneity and limited communications**: local devices are heterogeneous in performance, bandwidth, availability and reliability while the connection between the devices and the coordinating server can be slow, unstable and expensive (e.g., via commercial cellular networks).

These properties differentiate federated optimisation/learning from existing distributed optimisation/learning paradigms and also make FL a better fit for practical scenarios especially with privacy requirements. In the design and evaluations of traditional distributed training algorithms (e.g., [42, 51, 52]), the full data are either accessible to all the workers or shuffled before partitioning to guarantee IID data across the workers for local training. But such an ideal distribution of data is not realistic as it has been discussed in Section 1.1. On the other hand, these training algorithms were mainly designed for and evaluated on high-spec machines or GPU/TPU clusters connected with high-speed network, where the workers are homogeneous and the communication does not cause prominent delays. These infrastructure settings are also too utopian for most of the real-world systems that consist of general-purpose devices such as mobile phones, sensors and wearable devices — they run the training over limited bandwidth and sometimes charged, poor-quality channels.

By contrast, the motivation behind Federated Learning is to effectively perform federated optimisation in a communication-efficient and usually privacy-preserving manner. The original framework of FL inherits the canonical Client–Server (C–S) architecture of distributed ML and stems from the multi-step distributed SGD training for communication efficiency (which will be elaborated theoretically in Section 2.1.2, Chapter 2). The standard FL algorithm developed by Google is called FederatedAveraging (FedAvg) [47]. Along with the algorithm, a realistic FL system is also supposed to incorporate some privacy protection feature such as Homomorphic Encryption (HE) [53] or Differential Privacy (DP) [54] to prevent (or reduce) information disclosure regarding the raw data on local devices [49].

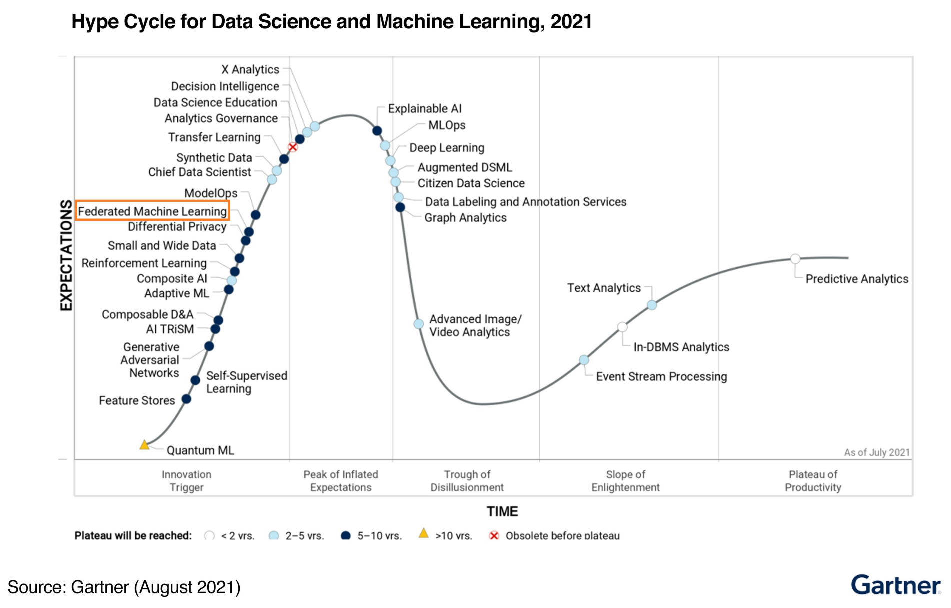
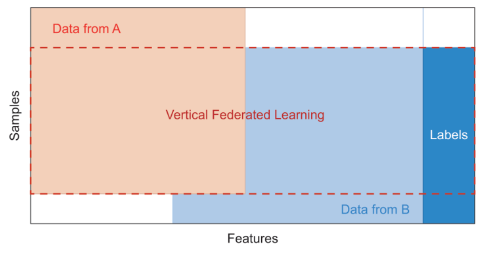


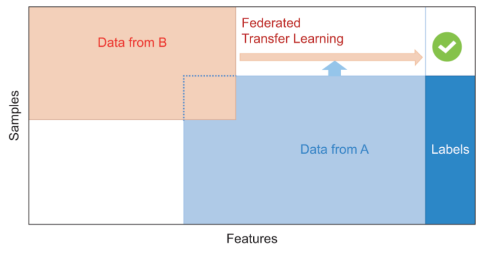
Figure 1.5: The 2021 Hype Cycle for Data Science and Machine Learning. (Source: Gartner)

The potential of FL has attracted lots of attention from both the academia and the industry. Gartner lists Federated (Machine) Learning as one of the promising technologies at the stage of 'Innovation Trigger' as part of the *Hype Cycle for Data Science and Machine Learning* for the year of 2021 (see Fig. 1.5). The prevalence of FL is largely attributed to its cross-disciplinary nature — the research spectrum of FL spans from distributed systems and machine learning to Secure Multi-party Computation (SMC) and wireless communication. The emergence of Federated Learning framework facilitates a diversity of studies that investigate the problem and optimise the system from different angles. In the domain of machine learning, extensive efforts have been made not only in the optimisation of FL, but also in the exploration of adapting the FL framework to multiple ML paradigms such as Transfer Learning, Multi-task Learning and Meta-learning. As a result, the FL framework has evolved into several different paradigms including Horizontal Federated Learning (HFL) , Vertical Federated Learning (VFL) and Federated Transfer Learning (FTL) [48]. These variants of FL are defined for different application scenarios based on the association between local data (namely, the way local datasets overlap with each other).

Different from horizontal FL that works in a common feature space (see Fig. 1.2), VFL was devised for collaborative training by multiple parties that desire a model for the same task but possess data with different features. As visualised in Fig. 1.6a, the situation can be considered as splitting the data vertically, which results in several sub-domains that have no or little overlap in the feature space. In the case of federated transfer learning, it usually involves two parties where party A (as the source domain) performs training to help party B (as the target domain, usually without labels) establish a model for the same machine learning task (Fig. 1.6b).



(a) VFL



(b) FTL

Figure 1.6: Vertical FL (VFL) and Federated Transfer Learning (FTL) (Source: IEEE Standard 3652.1-2020)

Table 1.1 categorises these paradigms from the perspective of data according to the *IEEE Guide for Architectural Framework and Application of Federated Machine Learning*[5] with some extension. More Details about VFL, FTL and other FL paradigms are introduced in the next chapter (Section 2.3.3) as related work.

Table 1.1: Categorisation of FL paradigms.

FL paradigms	Association of local data		
	Overlap of sample IDs	Overlap of features	labels
Horizontal FL	small	large (usually complete)	all
Vertical FL	large	small	partial
FTL	small	small	partial

Among these paradigms, horizontal federated learning is focused on the original problem of federated optimisation [47] where the data samples spread across a large number of clients but have the same feature space. As the prototypical paradigm, *horizontal federated learning is usually referred to as federated learning (FL for short) in the literature. Without ambiguity, the same convention is used in the rest of this thesis unless otherwise stated.*

[5]`https://standards.ieee.org/standard/3652_1-2020.html`

1.3 The Workflow of Horizontal FL

The goal of horizontal federated learning is to learn a global model by having the clients perform training on their local data collaboratively. The standard process of FL (e.g., FedAvg [47]) is organised in rounds. After initialisation, each round is comprised of the following steps:

1) the server selects a fraction of clients randomly to participate in this round of training;

2) the server distributes the latest global model to the selected clients;

3) the (selected) clients download the global model to overwrite their local models and perform training on their local data to update the models;

4) the (selected) clients upload their updated local models to the server;

5) the server aggregates the local models from the clients into a new global model.

The process repeats for a preset number of rounds or until the global model attains the desired level of quality (judged from the loss or accuracy in evaluations).

Fig. 1.7 gives an illustration of the workflow by showing the parameter server and clients in separate swim lanes with interactions.

1.4 Research Motivation

The most important hypothesis behind the research presented in this thesis is that *both device heterogeneity and data heterogeneity in an FL system can be addressed or mitigated by means of algorithm design.* On this basis, the motivation of this research is explained in this section with the emphasis on algorithmic improvements of the FL framework. The typical workflow of FL, as shown in Fig. 1.7, has three stages that are critical to the execution efficiency of a round: *client selection*, *local training* and *model aggregation*. Besides, the architecture and the synchronisation design of FL systems also play important roles.

Client selection

Client selection is the very first stage (after initialisation) on the server and usually a server-side parameter is needed for controlling the fraction of devices engaged in a round of training. Following the convention from [47], the parameter C is used to denote the client fraction which takes a value within $[0, 1]$ (where $C = 0$ means the server only selects one client per round).

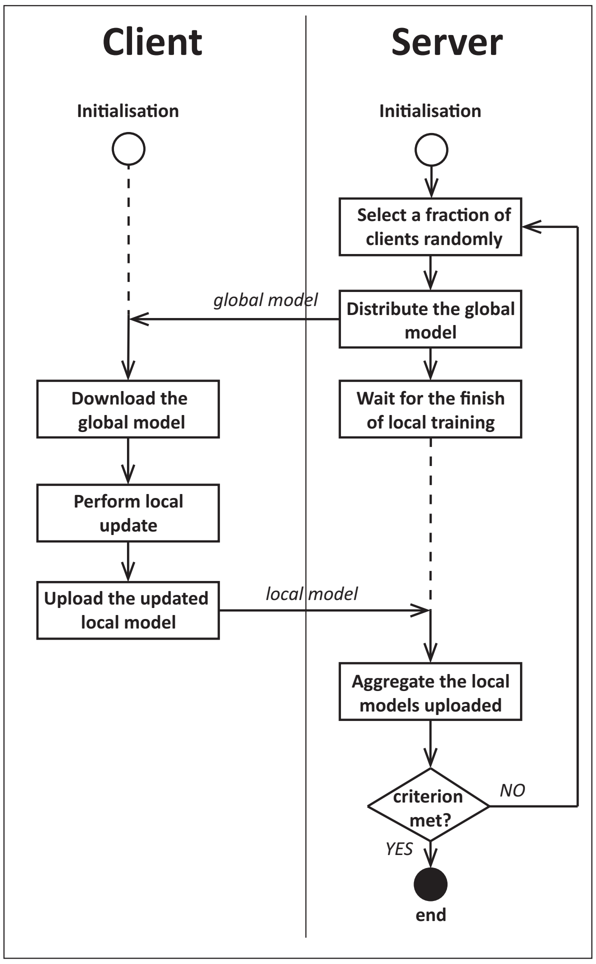

Client
Server
Initialisation
Initialisation
Select a fraction of clients randomly
global model
Distribute the global model
Download the global model
Wait for the finish of local training
Perform local update
Upload the updated local model
local model
Aggregate the local models uploaded
criterion met?
NO
YES
end

Figure 1.7: The workflow of a typical horizontal FL process.

In practice, it is suggested that C be set to small fraction (e.g., $C = 0.1$) [49]. The necessity of setting a small C rather than involving the entire fleet of clients can be explained from two perspectives. First, involving the entire set of clients does not necessarily lead to faster or better convergence of the global model. Experiments by McMahan et al. [47] with different models and data settings show that the global model can fail to converge (to a desired accuracy) by setting $C = 1$ (but manages to converge with a smaller C) with a large batch size. Their results also show some cases where the convergence of the global model benefits very little from a large client fraction when compared to smaller ones. The second reason for limiting the number of devices active in one round of training is about the cost. A higher value of the fraction C leads to increased involvement of each client by probability, which makes local training less resource-friendly. More importantly, more clients engaged in a round directly increases the traffic in the network as well as the burden of the server. For example, if each ML model exchanged between the clients and the server is 10

Megabytes in size and there are totally 1,000 clients, then running 100 rounds of FL over these clients incurs 1 Terabyte uplink traffic plus 1 Terabyte downlink traffic in total. Therefore, it is reasonable to set a relatively small fraction parameter.

Having a very limited portion of devices take part in a round of local training naturally adds subtleties to the stage of client selection. In the standard implementation of FL [12, 47], the server selects clients randomly at the beginning of each round. Random selection guarantees fairness of participation but ignores the heterogeneity of clients as well as their data. In certain situations, it is very likely that recruiting different clients in a certain round of training leads to complete different gains in terms of the quality and convergence of the global model. Also, from an engineering standpoint, choosing high-performance, reliable (i.e., less likely to drop out halfway) clients over the others may well speed up the whole training process. No matter from which perspective, there is a lot to explore in the design of client selection policies.

Local training

Once selected by and synchronised with the coordinating server, a client starts local training. The process of local training is basically no different than the traditional standalone gradient-based training except the range of data available. However, there are several challenges for local training in an FL system. First, the imbalanced size of local datasets introduces the following problem: with a constant and identical batch size, samples in a large local dataset get outweighted by the samples in a small local dataset, where a sample z_i outweights another sample $z_j (j \neq i)$ means z_i is sampled into a mini-batch (e.g., for SGD) by a higher probability (with random sampling) than z_j, as shown in Fig. 1.8. This consequently incurs two issues: i) local training on small datasets is more likely to cause over-fitting, and ii) data samples on smaller local datasets make more contribution than those on larger local datasets. Both issues can lead to a biased global model.

The non-identical distribution of local data brings up another challenge in local training. Training on ‘data islands’ leads to the divergence of local gradients (which will be analysed theoretically in Section 2.1.2, Chapter 2). The divergent gradients computed on different devices ‘drag’ the local models in different directions in the parameter space and consequently slows down the convergence of the global model. The phenomenon is quite common in realistic applications as the owners (or the devices themselves) behave differently and usually introduce biases into the data. For instance, if one wants to build a pet object detection app for smart phones via FL

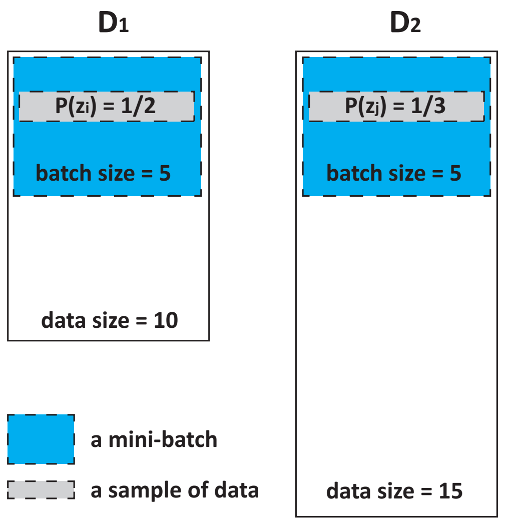


Figure 1.8: Local sampling on local datasets of different sizes.

then a prominent issue could be the discrepancy of data domains between users — some users may only take photos of cats and some may put funny clothes on their dogs. Therefore, it is worth exploring how to alleviate the negative impact of non-IID data on local training.

Model aggregation

Once the server has received a C-fraction of client models, the server enters the stage of model aggregation. The standard implementation of FL uses the FedAvg [47] algorithm that generates a new global model[6] $w(t)$ using the following rule at time step[7] t if t is an aggregation step:

$$w(t) = \sum_{k=1}^{|U|} \frac{n_k}{n} w_k(t), \text{ if } t \in T_A = \{nE|n = 1, 2, \ldots\} \tag{1.1}$$

where U is the set of clients, n_k is the size of client k's local dataset, n is the total data size (i.e., $n = \sum n_k$), E is the aggregation interval, and $w_k(t)$ is the updated local model based on $w_k(t-1)$ by client k. Throughout the thesis, a different but equivalent expression for the update rule may also be used:

$$w^{(r)} = \sum_{k=1}^{|U|} \frac{n_k}{n} w_k^{(r)}, \ r = 1, 2, \ldots \tag{1.2}$$

where the notation of round r is placed at the superscript. Note that each round usually involves multiple update steps.

[6]The model is actually a hypothesis parametrised by $w(t)$.
[7]Steps are counted as the total number of local updates performed.

The coefficient n_k/n is designed based on the relation between local training objectives $F_k(w)$ and the global objective $F(w)$. Eq. (1.3) formulates the global objective function $F(w)$ with more details presented in Section 2.1, Chapter 2.

$$F(w) = \sum_{k=1}^{|U|} \frac{n_k}{n} F_k(w), \tag{1.3}$$

As proven by Wang et al. [55], the FedAvg algorithm guarantees an upper bound on the error between the global model w and the centralised model trained by centralised SGD (Theorem 1, [55]). Also, the convergence upper bound of w (defined as the $F(w) - F(w^*)$ where w^* is the theoretically optimal model) is given as an expression of the total number of local iterations T, learning rate, aggregation interval and some assumed constants related to the properties of the objective functions (see [55] for details). Nonetheless, there are two main challenges for model aggregation.

The first challenge lies in the gap between the theoretical convergence rate and the actual convergence speed of the global model. The causes of the gap include the complexity of the loss function (e.g., non-convexity) and the discrepancy of local models' value in the aggregation. Efforts have been made in recent years towards analysing or bridging the gap for non-convex objective functions [56, 57] whilst it is relatively more sophisticated to assess the value of local models because i) the global model is evaluated after aggregation and ii) the improvement of the global model is very difficult to be traced down to any individual local model. This leaves a lot to desire in refining the aggregation rule.

The second challenge for model aggregation is about the timing. The standard FL algorithm uses synchronous control wherein the end of a round is triggered by the server upon receiving all the local models from the selected client subset. Although a preset round timer can be used in practice, the aggregation mechanism is still vulnerable to the failure of end devices. For example, given some unreliable clients each of which has a 20% probability to crash during local training, then involving 10 clients each round results in a merely $(1 - 0.2)^{10} \approx 10.7\%$ chance of successfully finishing the round in time. Therefore, it is necessary to consider ways to enable early aggregations for the sake of efficiency.

Architecture

FL inherits the classic client–server communication framework, in which the server does not take on the majority of computation but can possibly be overwhelmed in storage and network traffic when coordinating a massive group of clients. Basically there are three solutions to easing the burden of the server: reducing the size of models, changing the network topology and extending the architecture of the FL system.

Extensive studies have been carried out in the domain of model/gradient compression which was already a topic of interest since gradient–model exchange became the de facto paradigm for distributed machine learning. This branch of techniques can easily be applied to FL as compressing gradients and models are essentially the same.

The topologies of network can significantly influence the communication cost and efficiency of any distributed/parallel systems. The C–S structure of FL is equivalent to the spike–hub architecture where the central server handles all the traffic. It is very natural to think of more balanced, decentralised structures such as the Peer-to-Peer (P2P) network, which has already drawn attention from researchers [58–60]. However, the major problem of topologies without a central server role is the slowdown of information propagation. In machine learning systems, the convergence of the model may well be compromised if it takes longer (in time or in logical steps) to disseminate gradients/models over the whole network.

A promising direction is extending the layer architecture of the standard FL algorithm. Emerging techniques like Mobile Edge Computing and Internet of Things provide a natural platform for federated learning but most of them adopt the three-layer architecture where end devices are governed by edge nodes and edge nodes are connected to the cloud. In realistic systems, the layers can stack up and the nodes in the edge layer can interact with each other. There is certainly much potential in leveraging the multi-layer architecture to facilitate more efficient FL workflows.

Synchrony and Asynchrony

The concepts of synchrony and asynchrony differ in different domains of studies. In this thesis, these two terms are used to describe the behaviours of clients under the coordination by the server. Both synchronous and asynchronous algorithms for distributed learning have been well studied for conventional gradient-based training paradigms [61–64]. The main property that distinguishes synchrony and asynchrony is whether the server sets a 'synchronization barrier' to keep all the clients working in sync (i.e., always in the same logical round). Fig. 1.9 illustrates the difference with a case of two clients. Synchronous approaches barrier all the local updates to guarantee they are applied to a consensus model with regard to which the gradients are computed [63, 65, 66]. In contrast to that, asynchronous approaches adopt instant update on the server upon the arrival of any local update [67, 68]. It frees the server from long waiting at the cost of having to deal with stale updates [42, 69, 70]. Following the idea from asynchronous SGD algorithms [52, 67], Xie et al. [71] proposed *FedAsync*, an asynchronous federated optimisation scheme that regularises local optimisation and adopts a non-blocking update rule for the iterates of global model. A similar algorithm has been exploited by Sprague et al. [72] in a geo-spatial application for training a global model asynchronously, allowing the devices to join halfway. However, the main issue of the asynchronous approaches is that the server may receive too many local updates sent from a massive number of clients that remain active, which could overwhelm the server but with little benefit to the model convergence. Evidence from previous studies also shows that synchronous SGD can outperform asynchronous approaches in the data center setting [43], which to some extent inspired the synchronous design of FL [12]. Table 1.2 compares the upsides and downsides of synchronous and asynchronous training approaches over distributed data.

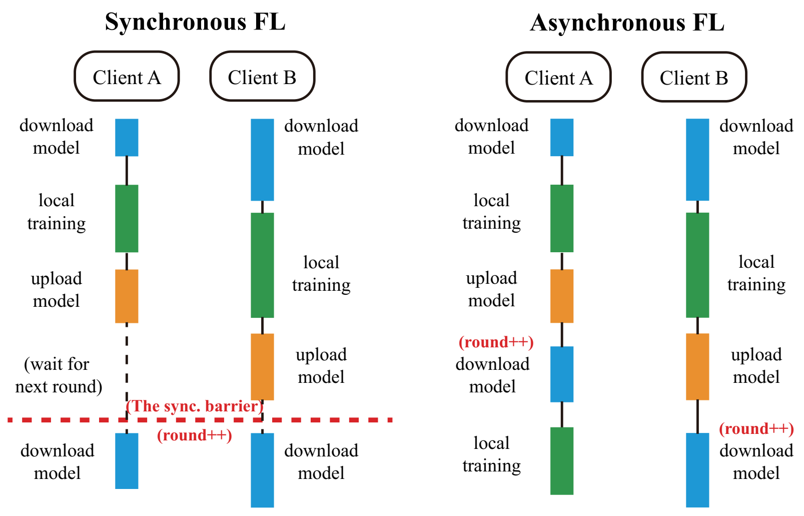


Figure 1.9: An illustration of the difference between synchrony and asynchrony in the context of FL.

Table 1.2: A comparison between synchronous and asynchronous approaches.

	Advantages	Disadvantages
Synchrony	•Stable convergence •Fewer communications	•Vulnerable to stragglers •Infrequent central updates
Asynchrony	•No sync. barriers •Frequent central updates	•Intensive communications •More local updates to converge

How to speed up the convergence rate of FL remains an open challenge. The optimisation mechanisms for traditional distributed SGD have great potential in FL. For example, gradient staleness control has been shown critical to guarantee convergence [73, 74]. Dutta et al. [74] theoretically characterised the trade-off between reducing error (by including more stragglers) and shortening run time (by bounding staleness). Wang et al. [75] refined ASGD by modulating the learning rate based on the staleness of incoming gradients. Smith et al. [76] proposed MOCHA, a fault- and straggler-tolerant multi-task learning method without forging a global model. Chen et al. [43] introduced backup workers to reduce server waiting time in synchronous stochastic optimisation. Inspired by these approaches, this work investigates the impact of straggling clients and model staleness on FL and presents a novel solution to the strong heterogeneity of participating devices by modifying the workflow of synchronous FL with asynchronous designs at a controllable cost of computation and communication.

1.5 Definition of Efficiency

Motivated by the aforementioned challenges, the research presented in this thesis is mainly focused on the improvement of horizontal FL in efficiency. The definition of efficiency for FL is much more intricate than the definition of accuracy for a single model. It is commonly recognised (especially in the field of machine learning) that one can use standard metrics such as top-k accuracy, precision, f1-score and mean error to evaluate, assess and compare models trained. But in the case of federated learning, the accuracy of the final global model contains way too limited information in terms of reflecting how good an FL algorithm is. Also, the value of FL is never merely about producing an accurate global model.

Therefore, efficiency-related metrics are defined in this work in order to assess the performance of FL from multiple perspectives. These metrics are *the number of rounds for convergence*, *total time for convergence*[8] and *average round length*[9].

Rounds for convergence

In this thesis, a round (or communication round) of FL is defined as the process spanning from client selection to model aggregation (see Fig. 1.7). As introduced in Section 1.3, the workflow of FL can go through any number of rounds until a stop

[8]The time here is recorded by the server no matter it is synchronous or asynchronous.
[9]Using different algorithms, rounds can differ in length.

criterion is met. In practice, the most convenient way is to preset a number of rounds to run but it is very hard to balance between the cost and gain as the rounds needed for a decent convergence is fairly unpredictable. Therefore, an alternative is to set a goal based on how good the model needs to be. Specifically, the goal can be an accuracy value (e.g., 90%) or a loss value with regard to the global model during evaluation. Then by running the FL process till the goal is achieved, one can assess the efficiency by the number of rounds needed:

$$\min_{r} Acc(w^{(r)}; D_{test}) \geq Acc\_goal \text{ or } F(w^{(r)}; D_{test}) \leq Loss\_goal, \tag{1.4}$$

where the Acc_goal and $Loss\_goal$ are the pre-specified accuracy goal and loss goal, respectively. D_{test} is a set of data for evaluating the global model and is typically possessed by the server. The superscript r denotes the round index as the version of the global model w.

Total time for convergence

The number of rounds is a logical measurement of how far the process of FL fares, but it does not reflect the actual cost of time. On this point, one can use the wall clock time to measure how long it takes for the global model to converge. Again, a preset goal is needed.

$$\min_{T} Acc(w^{(\Omega(T))}; D_{test}) \geq Acc\_goal \text{ or } F(w^{(\Omega(T))}; D_{test}) \leq Loss\_goal, \tag{1.5}$$

where $\Omega(T)$ denotes the number of training rounds performed given a time period of length T. For different FL algorithms, the process can go through different numbers of rounds given a same time budget T, which yields different values of $\Omega(T)$. It is noteworthy that the time for convergence is certainly associated with the rounds for convergence and may seem to be a more realistic (and thus valuable) metric than the latter, but in many cases the round needed could be more important as a reference of efficiency. This is mainly because the number of rounds also indicates the number of times the clients need to communicate with the server and thus the cost of communication throughout the system.

Average round length

The metric of average round length is derived from the number of rounds for convergence and the total time for convergence. It measures the efficiency of an FL process from the perspective of one round.

$$avg\_RT = \frac{\text{total time needed for convergence}}{\text{number of rounds for convergence}} \tag{1.6}$$

Average round length directly reveals how long on average it takes to finish a round of training, which includes a full cycle of client selection, model distribution, model downloading, local update, model uploading and model aggregation. It also indicates how fast the FL process is expected to proceed as the global model is updated once a round. However, the limitation of this metric is that it does not provide any information on the total time consumption or rounds needed to achieve the convergence.

In the evaluation of FL, it is suggested that all these metrics be taken into account. Beside these basic efficiency metrics, there are some auxiliary metrics that are related to or indicative about the efficiency of federated learning. For example, energy consumption of end devices reflects the cost of participating in the training as well as the efficiency of the FL process in terms of energy. The energy used by the clients is also a key factor that influences the willingness of the clients to participate because the global model is the only reward if no incentive mechanism (e.g., [77–79]) is applied. In addition to that, network traffic can also be used as a cost metric to characterise the burden on the devices and the channels caused by data transfer [80].

1.6 Summary of Contributions

In support of this dissertation, contributions are made by the author from the following perspectives: i) designing efficient algorithms for horizontal FL over heterogeneous devices, ii) integrating the two-layer FL framework with the three-layer architecture of edge computing systems, and iii) optimising the client selection policy for efficient FL over local data of varied quality.

1.6.1 FL over Heterogeneous Devices with Flexible Synchronisation

A typical use case of (horizontal) federated learning is model training in cross-device scenarios [49] where a vast number of end devices are recruited to contribute their computing power. However, it is very challenging to guarantee the efficiency of

FL considering the unreliable nature of end devices while the cost of device–server communication cannot be neglected. In view of this, a semi-asynchronous FL algorithm, SAFA, is presented in this thesis to address the problems in federated learning such as low round efficiency and poor convergence rate in extreme conditions (e.g., clients go offline frequently). Novel designs are introduced in the steps of model distribution, client selection and global aggregation to mitigate the impacts of stragglers, crashes and model staleness in order to boost efficiency and improve the quality of the global model. Results of extensive experiments with various machine learning tasks demonstrate that the proposed algorithm is effective in terms of shortening federated round duration, reducing local resource wastage and improving the accuracy of the global model at an acceptable communication overhead. This work is detailed in Chapter 3 corresponding to the published paper:

- Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A Jarvis. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70(5):655–668, 2021. [1]

1.6.2 Adapting Horizontal FL to the Edge Computing Architecture

The emerging Mobile Edge Computing (MEC) technique has shown great potential in narrowing the gap between where the data are processed and where the data reside. From this standpoint, MEC can be a natural fit for federated learning in terms of facilitating the training of AI models without breaching privacy regulations. However, it remains a big challenge to optimise the efficiency and effectiveness of FL when it is integrated with the MEC architecture whilst the unreliable nature (e.g., stragglers and intermittent drop-out) of end devices significantly slows down the FL process and affects the global model's quality in such circumstances. To address these issues, this research presents a multi-layer federated learning algorithm called HybridFL. The proposed solution adopts two levels (the edge level and the cloud level) of model aggregation enacting different aggregation strategies. Moreover, in order to mitigate stragglers and end device drop-out, controlling variables named Regional Slack Factors are introduced into the stage of client selection performed in the edge layer using a probabilistic approach without identifying or probing the state of end devices (whose reliability is unknown). Convergence analysis is presented for the proposed algorithm.

Extensive experiments were conducted with machine learning tasks in different scales of MEC system. The results show that HybridFL improves the FL training process significantly in terms of shortening the federated round length, speeding up the global model's convergence and reducing end device energy consumption. This work is detailed in Chapter 4 corresponding to the published paper:

- Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1539–1551, 2021. [2]

1.6.3 Selective Federated Learning over Data of Varied Quality

An issue that stands out in the settings of FL is that the whole system is agnostic about the quality of data that local models are trained on. In many scenarios, on top of the non-IID property of data, a large proportion of the clients are probably in possession of low-quality data that are biased, noisy or even irrelevant. As a result, they could significantly slow down the convergence of the global model we aim to build and also compromise its quality. In light of this, a novel algorithm called FedProf is presented with the aim of optimising FL under such circumstances without breaching data privacy. The key of the approach is a data representation profiling and matching scheme that uses the global model to dynamically profile data representations and allows for low-cost, lightweight representation matching. Based on the scheme the server adaptively scores each client and adjusts its participation probability so as to mitigate the impact of low-value clients on the training process. To evaluate the proposed solution, a series of experiments have been conducted by setting up both regression and image classification tasks under various FL settings. The results show that FedProf effectively reduces the number of communication rounds and overall time for the global model to converge while improving the accuracy of the final global model. This work is detailed in Chapter 5 corresponding to the paper:

- Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Fedprof: Selective federated learning with representation profiling. *arXiv preprint arXiv:2102.01733. Under review, ICLR'22*, 2021. [3]

1.7 Thesis Outline

This chapter provides an overview of the foundation, motivation and contributions of the research to be presented in this thesis. The remainder of this thesis is organised as follows:

Chapter 2 first introduces the preliminaries including the federated optimisation problem, a theoretical analysis on the key idea behind FL and the participation and model aggregation strategies. This part is a complement to the background knowledge provided in Chapter 1. Chapter 2 also provides an industrial vision by reviewing a wide range of applications and multiple existing FL systems and platforms. Then a literature review is presented discussing the relevant studies and recent advances in the research domain of FL, followed by a comprehensive list of FL datasets for facilitating relevant research.

Chapter 3 discusses the challenges in accelerating FL over a large group of heterogeneous devices. The impacts of involving heterogeneous, undependable clients are analysed, followed by the formulation of the system model. A semi-asynchronous FL algorithm SAFA with novel designs to address the heterogeneity of clients is presented in this chapter. Experimental results are provided to demonstrate the effectiveness of the proposed algorithm.

Chapter 4 investigates the emerging confluence of edge computing and AI applications followed by a breakdown of the benefits and challenges when it comes to federated learning in edge computing systems. In this chapter, a novel three-layer FL algorithm HybridFL is presented with the aim of performing FL efficiently in MEC systems over a massive group of clients of unknown reliability. Comprehensive results of experimental evaluation show significant boost of efficiency by using the proposed algorithm in a simulated MEC system.

Chapter 5 begins with the concept of representation learning for neural network models and studies why and how FL can be optimised with the exchange of representations. Besides, the problem of low-quality data on the training devices is discussed. Intriguing observations on the distribution of data representations from neural networks are given for which theoretically proofs are also provided. A representation profiling and matching scheme is further proposed and used to optimise the client selection strategy in FL. Based on the scheme, a novel FL algorithm called FedProf has been developed. Extensive experimental results demonstrate that the FedProf can leverage the learnt representations to differentiate the training value of different local datasets and facilitate fast convergence of the global model via selective client involvement.

Chapter 6 concludes this PhD thesis providing a summary of the research work presented in the previous chapters. This chapter also discusses the open challenges in developing FL systems and several possible research directions worth exploring in the future.

Chapter 2

Background and Literature Review

Since the emergence of federated learning, we have seen rapid development of the framework and relevant techniques thanks to the extensive efforts by the researchers and practitioners in this field of study. Fig. 2.1 visualises the research spectrum of FL with a word cloud generated from the titles of >200 *most-cited papers*. Most of these influential studies are included in the references of this thesis and will be discussed in this chapter.

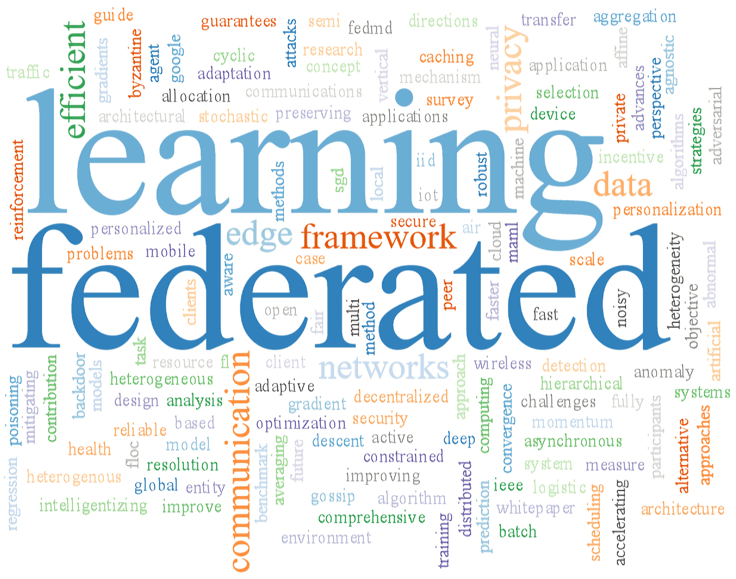


Figure 2.1: A word cloud of research keywords for visualising the spectrum of FL.

In this chapter, important preliminaries for FL are first introduced including the problem formulation of federated optimisation, the key idea behind FL (also its key difference from traditional distributed training), the participation control and model aggregation methods as well as the synchrony and asynchrony in the context of FL. The second section of this chapter casts light on the development of FL in the industry covering libraries, systems and applications. The third section discusses the related research work including distributed learning, various paradigms of FL, communication optimisation in FL and privacy preservation solutions in FL. The last section provides a list of public datasets for FL research.

2.1 Preliminary Knowledge

This section first mathematically formulates the objective of federated learning. Important prerequisites are then introduced for a better understanding of the key mechanisms in FL.

2.1.1 Problem Formulation

Federated learning was originally introduced by McMahan et al. [47]. The framework is termed federated learning because of its property of being a 'loose federation of participating devices'. In the seminal paper the authors 'refer to the optimisation problem implicit in federated learning as federated optimisation', where the problem and the settings thereof were first defined in [81]. Formally, the problem is to find a model that minimises the global objective (or empirical risk) $F(w)$ as follows:

$$\arg\min_{w} F(w) = \frac{1}{n} \sum_{(x_i, y_i) \in D} \ell(w; x_i, y_i), \tag{2.1}$$

where D contains all the data across the system, n is the size of D and $\ell(\cdot)$ denotes the loss function which can be any forms including non-convex neural network objectives [47]. The vector w parametrises the model (hypothesis) $h_w \in \mathcal{H} : \chi \to \mathcal{Y}$ one aims to establish.[1] Since the optimisation has to be done on decentralised data (D is a logical union of numerous distributed local datasets), each client is actually working on its respective local objective $F_k(w)$:

$$\arg\min_{w} F_k(w) = \frac{1}{n_k} \sum_{(x_i, y_i) \in D_k} \ell(w; x_i, y_i), \tag{2.2}$$

[1]Without ambiguity, w is also referred to as the model itself for brevity in this thesis.

where D_k is the local dataset on client k and $n_k = |D_k|$ is the size of D_k. Using the relations $D = \bigcup_{k \in U} D_k$ and $n = \sum_{k \in U} n_k$, the global objective (2.1) can be rewritten as:

$$\arg\min_{w} F(w) = \frac{n_k}{n} \sum_{k \in U} F_k(w), \tag{2.3}$$

where U is the set of clients.

Note the problem formulation given in this section is strictly consistent with the seminal work by McMahan et al. [47] and so are the majority of researches under the framework of federated learning. But actually the objectives can be adapted in order to fit different applications, system settings or algorithm designs. For example, local objectives can be averaged with identical weights [71, 82]. In the research work to be presented in Chapter 5, a different global objective is defined based on the idea that the local data on different clients differ in training value and thus should be differentiated in their weights.

2.1.2 The Key Idea behind FL

Decentralised data requires distributed training. Gradient-based methods (such as Gradient Descent (GD) and Stochastic Gradient Descent (SGD)) are currently the most popular choices for training machine learning models (especially deep neural networks). It is also true that training with gradient-based methods (including the way local models are trained in horizontal FL) on decentralised data (without any form of data exchange) always leads to the divergence of gradients (except the case of using GD on even sizes of local data). More specifically, the exact gradients (obtained by centralised training on full data) cannot be recovered using the gradients computed on several separate datasets in most of the practical situations, which is termed *Gradient Divergence* in this work. **Essentially, (horizontal) federated learning is a form of multi-step SGD over decentralised datasets with periodic aggregation**. The following section provides a theoretical analysis on how training on decentralised data leads to gradient divergence and its connection to the key value of FL.

By analysing why running distributed GD and SGD on 'data islands' introduces gradient divergence and how much the gradients deviate from the centralised, 'correct' gradients, **this section aims to reveal the impact of 'data islands' on gradient-based training and ultimately the key value of (horizontal) FL.**

In the work presented in this thesis, gradient divergence is defined as the deviation of the globally aggregated gradients (over decentralised data) from the centralised gradients (virtually over a full set of data). Without loss of generality, this definition is suitable for both GD and SGD (thus the terms *gradients* and *stochastic gradients* are generally interchangeable in the following analysis). More specifically, the aggregated gradients refer to the gradients calculated by the parameter server (PS) by aggregating all the locally computed gradients uploaded by the training devices (called workers in some studies and clients in the context of federated learning). The centralised gradients, on the other hand, are defined as the result of gradient computation over the union of data (i.e., a complete dataset containing all the data from all the clients) using sequential SGD. The centralised gradients are virtual and impractical (because of the data locality constraint in FL) but are usually used as a baseline that represents the ideal direction of each model update. In the following content, theoretical analysis on gradient divergence is provided.

Given a set of data holders (clients) U, let D_k denote the local dataset on client k, $D = \bigcup_{k \in U} D_k$ be the joint, complete dataset and $\tilde{w}$ be the virtual model trained on D (in a standalone manner). let G denote the aggregated gradient from locally computed gradients $\{G_k\}_{k \in U}$ and let $\tilde{G}$ denote the virtual gradient obtained from gradient computation on the complete dataset D. Both G_k and $\tilde{G}$ are calculated based on sample-wise gradients. Given a data sample (X_i, Y_i) and a model w (in the following content, w also represents the parameter set of the model), the sample-wise gradient $g_i(w)$ with regard to w can be formulated as:

$$g_i(w) \triangleq \nabla \ell(w; X_i, Y_i) = \frac{\partial \ell(w; X_i, Y_i)}{\partial w}, \tag{2.4}$$

where $\ell(\cdot)$ is the loss function (as the objective function).

Given a data sample (X, Y) randomly drawn from D, the centralised gradient with regard to $\tilde{w}$ is a random variable that has the expectation $\mathbb{E}_{\sim D}[g(\tilde{w})]$ (without ambiguity, $\mathbb{E}_{\sim D}$ means $\mathbb{E}_{(X,Y) \sim D}$ unless otherwise stated):

$$\mathbb{E}_{\sim D}[g(\tilde{w})] = \sum_{i=1}^{|D|} \frac{1}{|D|} g_i(\tilde{w}). \tag{2.5}$$

Similarly, the expectation of sample-wise gradient over the samples in a local dataset D_k can be formulated as:

$$\mathbb{E}_{\sim D_k}[g(w_k)] = \sum_{(X_i,Y_i)\in D_k} \frac{1}{|D_k|} g_i(w_k), \tag{2.6}$$

where w_k denotes the model snapshot on client k based on which the gradient is computed. Note that w_k is not necessarily same as $\tilde{w}$. In the following analysis, it is assumed that $w_k = \tilde{w}$ only for the case of one-step update.

In this work, G (the aggregated gradient) and $\tilde{G}$ (the centralised gradient) are defined *differently* for GD, SGD and multi-step SGD. In the following content, the gradient divergence introduced by distributed training is analysed in three cases: Gradient Descent update, Stochastic Gradient Descent update, and Multi-step SGD update (the basis of horizontal FL).

Case 1: Gradient Descent update

In this case, $\tilde{G}$ and $\{G_k\}_{k\in U}$ are computed using GD over full batches of data from the complete dataset and the local datasets, respectively. Using GD, gradients are deterministic and are computed as the average of all sample-wise gradients. By the definition of $\tilde{G}$ and G_k, Gradient Descent yields:

$$\tilde{G} = \frac{1}{|D|} \sum_{(X_i,Y_i)\in D} g_i(\tilde{w}), \tag{2.7}$$

and

$$G_k = \frac{1}{|D_k|} \sum_{(X_i,Y_i)\in D_k} g_i(w_k). \tag{2.8}$$

Further, the aggregated gradient G can be derived from G_k:

$$\begin{aligned} G &= \frac{1}{|U|} \sum_{k\in U} G_k \\ &= \frac{1}{|U|} \sum_{k\in U} \frac{1}{|D_k|} \sum_{(X_i,Y_i)\in D_k} g_i(w_k) \\ &= \sum_{(X_i,Y_i)\in D} \frac{1}{|U|} \frac{1}{|D_{i_k}|} g_i(w_k), \end{aligned} \tag{2.9}$$

where i_k denotes the index of the client/local dataset that the sample (X_i, Y_i) belongs to.

For a single step of gradient computation, the gradient divergence can be formally defined as $\mathbb{E}\|G - \tilde{G}\|$, where $\|\cdot\|$ denotes the 2-norm used to measure the distance between vectors. Due to the deterministic nature of full-batch Gradient Descent, the divergence between G and $\tilde{G}$ is given by:

$$\begin{aligned}\mathbb{E}\|G-\tilde{G}\| &= \|G-\tilde{G}\| \\ &= \|\sum_{(X_i,Y_i)\in D}\frac{1}{|U||D_{i_k}|}g_i(w_{i_k})-\frac{1}{|D|}\sum_{(X_i,Y_i)\in D}g_i(\tilde{w})\|,\end{aligned} \tag{2.10}$$

From Eq. (2.10), one can see that in the virtually centralised training on D (which produces the centralised gradient $\tilde{G}$), every sample shares the same weight $\frac{1}{|D|}$, whilst for the aggregated gradient G, the sample-wise gradients are weighted by the reciprocal of its corresponding local dataset's size times the population of clients.

Considering a single step of update from a fixed point in the parameter space (i.e., $w_k = \tilde{w} = w$, $\forall k \in U$), Theorem 2.1 is given as follows:

Theorem 2.1. *With a single step of update using Gradient Descent with regard to a same model (i.e., $w_1 = w_2 = \cdots = w_{|U|} = \tilde{w} = w$), the aggregated gradient G is equal to the centralised gradient $\tilde{G}$ over the full set of data when all the local datasets are of the same size or all the local data are IID, i.e.,*

$$G = \tilde{G}, \tag{2.11}$$

only if $|D_1| = |D_2| = \ldots = |D_{|U|}| = \frac{1}{|U|}|D|$ or $P\big((X,Y)|D_1\big) = P\big((X,Y)|D_2\big) = \ldots = P\big((X,Y)|D\big)$.

Proof. Given the first condition (i.e., even local data sizes) in Theorem 2.1 and recall that $w_k = \tilde{w} = w \, \forall k \in U$, the aggregated gradient can be rewritten as (2.12).

$$\begin{aligned}G &= \sum_{(X_i,Y_i)\in D}\frac{1}{|U|}\frac{1}{|D_{i_k}|}g_i(w_{i_k}) \\ &= \sum_{(X_i,Y_i)\in D}\frac{1}{|U|}\frac{1}{\frac{1}{|U|}|D|}g_i(w) \\ &= \sum_{(X_i,Y_i)\in D}\frac{1}{|D|}g_i(w), \\ &= \tilde{G}\end{aligned} \tag{2.12}$$

which proves Theorem 2.1.

Alternatively, given the second condition (i.e., IID local data) in Theorem 2.1 and recall that $w_k = \tilde{w} = w$, $\forall k \in U$, one can derive that:

$$\tilde{G} = \frac{1}{|D|} \sum_{(X_i,Y_i)\in D} g_i(\tilde{w}) = \mathbb{E}_{\sim D}[g(w)], \tag{2.13}$$

$$G_k = \frac{1}{|D_k|} \sum_{(X_i,Y_i)\in D_k} g_i(w) = \mathbb{E}_{\sim D_k}[g(w)], \tag{2.14}$$

where $\mathbb{E}_{\sim D}[g(\cdot)]$ and $\mathbb{E}_{\sim D_k}[g(\cdot)]$ are defined in Eqs. (2.5) and (2.6), respectively. The IID-data condition $P\big((X,Y)|D_k\big) = P\big((X,Y)|D\big)$, $\forall k \in U$ implies that the gradient distribution $P(g(w)|D_k)$ (where $g(w)$ denotes the gradient given a random sample (X,Y)) is identical to $P(g(w)|D)$ w.r.t. a same model snapshot w. Thus we have:

$$\mathbb{E}_{\sim D}[g(w)] = \mathbb{E}_{\sim D_k}[g(w)]. \tag{2.15}$$

Equivalently,

$$G_k = \tilde{G}. \tag{2.16}$$

Combining Eqs. (2.20) and (2.16), it follows that

$$\begin{aligned} G &= \frac{1}{|U|} \sum_{k\in U} G_k \\ &= \frac{1}{|U|} \sum_{k\in U} \tilde{G} \\ &= \tilde{G}, \end{aligned} \tag{2.17}$$

which proves Theorem 2.1. □

Remark. *Gradient Descent is a deterministic optimisation method that produces the exact gradients over a full batch of data samples. Nonetheless, successful recovery of the exact gradient by aggregating locally computed gradients is subject to two conditions as stated in Theorem 2.1. These conditions, however, are rarely satisfied in practical situations. For example, it is more common than not to have different sizes of local training data. Besides, GD is not a good option for modern deep learning models due to its low training efficiency (because each iteration of update involves an entire set of data).*

Case 2: Stochastic Gradient Descent update

SGD improves the efficiency of GD by picking one or a random subset of training samples from a dataset for gradient computation in each iteration. SGD is also referred to as mini-batch Stochastic Gradient Descent when gradients are computed over a subset (mini-batch) of data for parallelism. In the following content, gradient divergence for *a single step of SGD update* is analysed on the basis that local models $\{w_k\}_{k \in U}$ and the virtual model $\tilde{w}$ are at the same point in the parameter space (i.e., $w_1 = w_2 = \cdots = w_{|U|} = \tilde{w} = w$).

Let B_k denote a mini-batch of data drawn from D_k for one step of gradient computation and $\tilde{B}$ be a global mini-batch of data from D. To facilitate the analysis, let $|\tilde{B}| = \sum_{k \in U} |B_k|$ always hold (note that it does not necessarily mean $\tilde{B} = \bigcup_{k \in U} B_k$ as samples are randomly drawn) and let $b_k = |B_k|$ and $\tilde{b} = |\tilde{B}|$. Without generality, we also assume an identical local batch size b for the computation of local gradients G_k, i.e., $b_1 = b_2 = \ldots = b$, and thus the batch size $\tilde{b}$ for computing $\tilde{G}$ satisfies $\tilde{b} = |U| \cdot b$. Then by the definition of SGD, it follows that

$$\tilde{G} = \frac{1}{\tilde{b}} \sum_{(X_i, Y_i) \in \tilde{B}} g_i(\tilde{w}), \tag{2.18}$$

and

$$G_k = \frac{1}{b_k} \sum_{(X_i, Y_i) \in B_k} g_i(w_k), \tag{2.19}$$

where $1 \leq \tilde{b} \leq |D|$ is the global batch size for computing $\tilde{G}$ (on the full data w.r.t. the model $\tilde{w}$) and $1 \leq b_k \leq |D_k|$ is the local batch size for computing local gradients (on the local data of client k w.r.t. the local model w_k). With local gradients given by Eq. (2.19), the aggregated gradient G, given by Eq. (2.20), is the average across all the local gradients uploaded to the parameter server (PS).

$$\begin{aligned} G &= \frac{1}{|U|} \sum_{k \in U} G_k \\ &= \frac{1}{|U|} \sum_{k \in U} \frac{1}{b_k} \sum_{(X_i, Y_i) \in B_k} g_i(w_k) \\ &= \sum_{(X_i, Y_i) \in B} \frac{1}{|U|} \frac{1}{b_{i_k}} g_i(w_{i_k}), \\ &(\text{let}\, B = \bigcup_{k \in U} B_k). \end{aligned} \tag{2.20}$$

Combining Eqs. (2.18) and (2.20), the gradient divergence for the case of SGD can be formulated as:

$$
\begin{aligned}
\|G - \tilde{G}\| &= \|\frac{1}{|U|}\sum_{k\in U} G_k - \tilde{G}\| \\
&= \|\frac{1}{|U|}\sum_{k=1}^{|U|} G_k - \frac{1}{|U|}\sum_{k=1}^{|U|}\tilde{G}\| \\
&= \frac{1}{|U|}\|\sum_{k=1}^{|U|}(G_k - \tilde{G})\| \\
&\le \frac{1}{|U|}\sum_{k=1}^{|U|}\|G_k - \tilde{G}\| \\
&= \frac{1}{|U|}\sum_{k=1}^{|U|}\|\frac{1}{b_k}\sum_{(X_i,Y_i)\in B_k} g_i(w_k) - \frac{1}{\tilde{b}}\sum_{(X_i,Y_i)\in \tilde{B}} g_i(\tilde{w})\|, \qquad (2.21)
\end{aligned}
$$

which implies that the upper bound of gradient divergence for single-step SGD boils down to the randomness of SGD (in sampling) and the relation between the data distribution of each D_k and the population distribution of D.

Considering that every batch of samples are drawn randomly from the dataset D for computing $\tilde{G}$ (or from D_k for G_k), the expectation of $\tilde{G}$ can be formulated as:

$$
\begin{aligned}
\mathbb{E}[\tilde{G}] &= \frac{1}{\tilde{b}}\mathbb{E}_{\tilde{B}\sim D}[\sum_{(X_i,Y_i)\in\tilde{B}} g_i(\tilde{w})] \\
&= \frac{1}{\tilde{b}}|\tilde{B}|\mathbb{E}_{\sim D}[g_i(\tilde{w})] \\
&= \frac{1}{\tilde{b}}\tilde{b}\sum_{j=1}^{|D|}\frac{1}{|D|}g_j(\tilde{w}) \\
&= \sum_{i=1}^{|D|}\frac{1}{|D|}g_i(\tilde{w}) \\
&= \mathbb{E}_{\sim D}[g(\tilde{w})], \qquad (2.22)
\end{aligned}
$$

where $\mathbb{E}_{\sim D}[g(\tilde{w})]$ denotes the expectation of sample-wise gradient $g(\tilde{w})$ on the complete data. The result is intuitive because the gradient of a randomly drawn sample $g_i(\tilde{w})$ is a random variable, which means that the average gradient $\tilde{G}$ over a mini-batch of randomly drawn samples is also a random variable with the same expectation. As discussed in [83], $\tilde{G}$ actually follows the Gaussian distribution $\mathcal{N}(\mathbb{E}_{\sim D}[g(\tilde{w})], \frac{\Sigma(\tilde{w})}{\tilde{b}})$ (where $\Sigma(\tilde{w})$ denotes the covariance matrix of $g(\tilde{w})$ over D) according to the Central

Limit Theorem (CLT). In a similar way, the expectation of G_k can be formulated as:

$$\begin{aligned}\mathbb{E}[G_k] &= \mathbb{E}_{B_k \sim D_k}[\frac{1}{b_k}\sum_{(X_i,Y_i)\in B_k} g_i(w_k)] \\ &= \frac{1}{b_k}|B_k|\mathbb{E}_{\sim D_k}[g_i(w_k)] \\ &= \frac{1}{b_k}b_k \sum_{(X_i,Y_i)\in D_k} \frac{1}{|D_k|}g_i(w_k) \\ &= \sum_{(X_i,Y_i)\in D_k} \frac{1}{|D_k|}g_i(w_k) \\ &= \mathbb{E}_{\sim D_k}[g(w_k)], \end{aligned} \tag{2.23}$$

where $\mathbb{E}_{\sim D_k}[g(w_k)]$ denotes the expectation of sample-wise gradient over local dataset D_k with regard to the local model w_k.

Remark. *Comparing Eq. (2.22) and (2.23), one can find out that, for a single step of SGD, the difference between the expectations of the centralised gradient $\tilde{G}$ and the locally computed gradient G_k lies in i) the coverage of data samples and ii) the models. It is noteworthy that, even if $\tilde{G}$ and G_k are supposedly computed w.r.t. a same model snapshot (i.e., $\tilde{w} = w_k$), the two expectations are equal only when the data in D_k and D are Independent and Identically Distributed (IID). If the data are non-IID across the clients (which is the common case for FL), G_k cannot be expected as an unbiased estimate of $\tilde{G}$, i.e., $\mathbb{E}[G_k] \neq \mathbb{E}[\tilde{G}]$.*

Given the expression of the aggregated gradient G (Eq. 2.20), the expectation $\mathbb{E}[G]$ can be formulated as:

$$\begin{aligned}\mathbb{E}[G] &= \mathbb{E}[\frac{1}{|U|}\sum_{k\in U} G_k] \\ &= \frac{1}{|U|}\sum_{k\in U}\mathbb{E}[G_k] \\ &= \frac{1}{|U|}\sum_{k\in U}\mathbb{E}_{B_k \sim D_k}[\frac{1}{b_k}\sum_{(X_i,Y_i)\in B_k} g_i(w_k)] \\ &= \frac{1}{|U|}\sum_{k\in U}\frac{1}{b_k}|B_k|\mathbb{E}_{\sim D_k}[g_i(w_k)] \\ &= \frac{1}{|U|}\sum_{k\in U}\frac{1}{b_k}b_k\sum_{(X_i,Y_i)\in D_k}\frac{1}{|D_k|}g_i(w_k) \\ &= \sum_{i=1}^{|D|}\frac{1}{|U|}\frac{1}{|D_{i_k}|}g_i(w_{i_k}), \end{aligned} \tag{2.24}$$

where $|U|$ is the number of clients (or local datasets) and i_k represents the index of the client that sample (X_i, Y_i) resides on.

From the perspective of expectation, Theorem 2.2 is given as follows:

Theorem 2.2. *With regard to a same point w in the parameter space (i.e., $w_k = \tilde{w} = w$ for any $k \in U$), the aggregated gradient G is an unbiased estimate of the idealised gradient $\tilde{G}$ if at least one of the following conditions is satisfied: i) all the local datasets have the same size; ii) the data are IID in all the local datasets. Formally,*

$$\mathbb{E}[G] = \mathbb{E}[\tilde{G}] \tag{2.25}$$

only if $|D_1| = |D_2| = \ldots = |D_{|U|}| = \frac{1}{|U|}|D|$ or $P\big((X,Y)|D_1\big) = P\big((X,Y)|D_2\big) = \ldots = P\big((X,Y)|D\big)$.

Proof. From Eq. (2.22) and (2.24), it follows that

$$\mathbb{E}[G] - \mathbb{E}[\tilde{G}] = \sum_{i=1}^{|D|} \frac{1}{|U|} \frac{1}{|D_{i_k}|} g_i(w_{i_k}) - \sum_{i=1}^{|D|} \frac{1}{|D|} g_i(\tilde{w}). \tag{2.26}$$

Theorem 2.2 apparently holds with the first condition satisfied in which case $\frac{1}{|U|}\frac{1}{|D_{i_k}|} = \frac{1}{|D|}$ and given $w_k = \tilde{w} = w$ for any $k \in U$.

To prove Theorem 2.2 under the second condition that the data are IID across all the local datasets, one can use the same logic as in the proof of Theorem 2.1 and first rewrite Eq. (2.26) as:

$$\mathbb{E}[G] - \mathbb{E}[\tilde{G}] = \frac{1}{|U|} \sum_{k \in U} \mathbb{E}_{\sim D_k}[g(w_k)] - \mathbb{E}_{\sim D}[g(\tilde{w})]. \tag{2.27}$$

Recall that IID data implies IID sample-wise gradients for any $D_k (k = 1, 2, ..., |U|)$ and D with regard to an identical model snapshot w, i.e., $\mathbb{E}_{\sim D_k}[g(w)] = \mathbb{E}_{\sim D}[g(w)]$. Therefore, with $w_k = \tilde{w} = w$ for any $k \in U$, one can further derive that:

$$\begin{aligned}
\mathbb{E}[G] - \mathbb{E}[\tilde{G}] &= \frac{1}{|U|} \sum_{k \in U} \mathbb{E}_{\sim D}[g(w)] - \mathbb{E}_{\sim D}[g(w)] \\
&= \frac{1}{|U|} |U| \mathbb{E}_{\sim D}[g(w)] - \mathbb{E}_{\sim D}[g(w)] \\
&= 0,
\end{aligned} \tag{2.28}$$

which proves Theorem 2.2. □

Remark. *The gradient divergence for single-step SGD sources from i) the uneven distribution of local dataset size and ii) non-IID data distribution across the training clients. The discrepancy in the amount of data held by the clients results in the discrepancy of mini-batch sampling from the datasets. This can be explained intuitively from the perspective of a single sample* (X_i, Y_i)*. If* (X_i, Y_i) *resides on client* k*, then with a mini-batch size of* b*, probabilistically it has the chance* $\frac{b}{|D_k|}$ *that it will be selected into the mini-batch for a step of SGD update and contributes to* G_k *and* G*. On the other hand, recall that the complete dataset* D *has all the data samples and therefore, the sample* (X_i, Y_i) *is drawn from* D *by the probability* $\frac{\tilde{b}}{|D|}$*, where* $\tilde{b} = |U|b$ *is the centralised mini-batch size. Consequently, this leads to unequal likelihood of contribution by the sample* (X_i, Y_i) *to* G *and* $\tilde{G}$ *when* $|D_k| \neq \frac{1}{|U|}|D|$*.*

Non-IID data directly results in local gradients G_k *being potentially diverged from* $\tilde{G}$*. The expectation of gradients over a mini-batch of samples is exactly equal to that over the dataset whilst non-IID data indicates a different distribution of* D_k *from* D*. Consequently, local gradients deviate from the centralised, 'correct' gradients in expectation, which can possibly result in the gradient divergence between* G *and* $\tilde{G}$*.*

Case 3: Multi-step SGD update

Training machine learning models using distributed SGD on decentralised data is essentially equivalent to running multiple SGD chains, as opposed to the single SGD chain on the joint, complete dataset. In the case of multi-step SGD update, multiple steps of gradient computation are performed between two aggregation operations, i.e., periodic averaging. To facilitate the analysis, the notations of local gradient $G_k(t)$ and local model $w_k(t)$ at time point t are introduced, where $G_k(t)$ is computed with regard to $w_k(t)$ and by applying $G_k(t)$, $w_k(t)$ is updated to $w_k(t+1)$ using the following rule:

$$w_k(t+1) = w_k(t) - \eta G_k(t), \tag{2.29}$$

where η is the learning rate. Similarly, the virtual model $\tilde{w}$ trained on D is updated as:

$$\tilde{w}(t+1) = \tilde{w}(t) - \eta \tilde{G}(t), \tag{2.30}$$

Both single-step SGD and multi-step SGD compute gradients and update models in chains. The difference is that single-step SGD aggregates all the local gradients at every time step whilst multi-step SGD delays the aggregation. Fig. 2.2 illustrates the

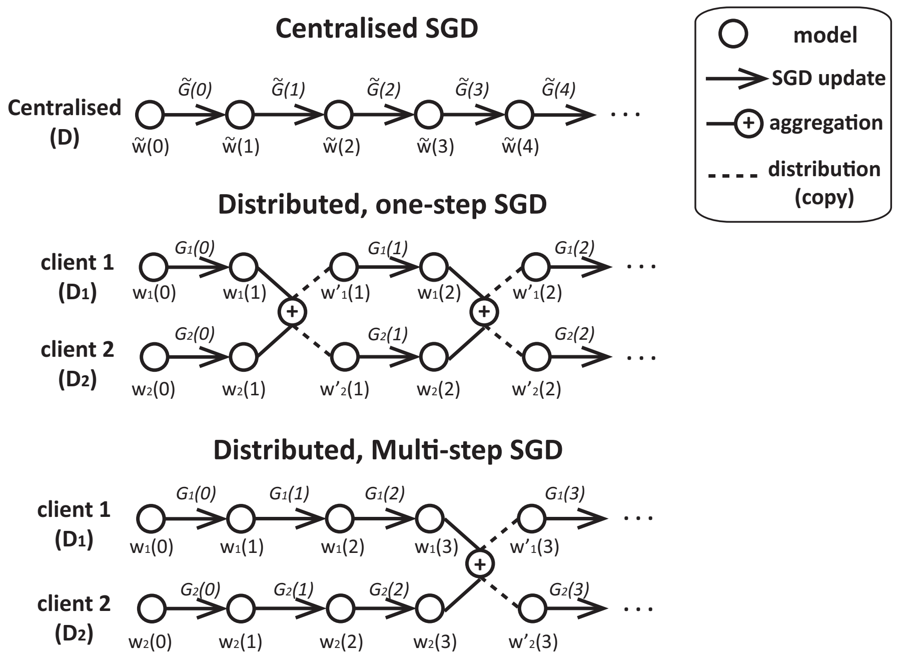

Centralised SGD
Centralised (D)
G̃(0) G̃(1) G̃(2) G̃(3) G̃(4)
w̃(0) w̃(1) w̃(2) w̃(3) w̃(4)
Distributed, one-step SGD
client 1 (D1)
client 2 (D2)
G1(0) G1(1) G1(2)
w1(0) w1(1) w'1(1) w1(2) w'1(2)
G2(0) G2(1) G2(2)
w2(0) w2(1) w'2(1) w2(2) w'2(2)
Distributed, Multi-step SGD
G1(0) G1(1) G1(2) G1(3)
w1(0) w1(1) w1(2) w1(3) w'1(3)
G2(0) G2(1) G2(2) G2(3)
w2(0) w2(1) w2(2) w2(3) w'2(3)
model
SGD update
aggregation
distribution (copy)

Figure 2.2: The chains of gradient computation.

difference between single-step SGD update and multi-step SGD update on decentralised data given two clients (i.e., two local datasets). With multi-step SGD update on the clients, local gradients are sent to the coordinating PS for aggregation at a fixed interval (i.e., a number of updates). In this part, the analysis presented is focused on one interval containing τ SGD updates (for example, $\tau = 3$ in Fig. 2.2) and one aggregation.

In the following content, the gradient divergence is analysed given the same number of SGD updates (i.e., τ) for calculating the centralised gradient $\tilde{G}$ and the aggregated gradient G. Both $\tilde{G}$ and G are accumulated through τ updates. First, by summing up Eq. (2.30) for all $t = 0, 1, \dots, \tau - 1$ one can derive that

$$\sum_{t=0}^{\tau-1} \tilde{w}(t+1) = \sum_{t=0}^{\tau-1} \big(\tilde{w}(t) - \eta \tilde{G}(t)\big), \tag{2.31}$$

which can be reorganised by eliminating the terms $\tilde{w}(1), \tilde{w}(2), \dots, \tilde{w}(\tau-1)$ on both sides of the equation:

$$\tilde{w}(\tau) = \tilde{w}(0) - \eta \sum_{t=0}^{\tau-1} \tilde{G}(t). \tag{2.32}$$

Similarly, using Eq. (2.29) one can formulate $w_k(\tau)$ as:

$$w_k(\tau) = w_k(0) - \eta \sum_{t=0}^{\tau-1} G_k(t). \tag{2.33}$$

From the above equations, one can see that multi-step SGD accumulates the gradients along the local update chains until aggregation happens. Aggregating the *accumulated* local gradients across the clients yields:

$$G^{cum(\tau)} = \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} G_k(t), \tag{2.34}$$

where $\sum_{t=0}^{\tau-1} G_k(t)$ is the accumulated gradients along the SGD chain of client k for τ steps and $G^{cum(\tau)}$ denotes the aggregated gradient.

Note that averaging accumulated local gradients and aggregating local models are equivalent, i.e.,

$$\begin{aligned} w(\tau) &= \frac{1}{|U|}\sum_{k\in U} w_k(\tau) \\ &= \frac{1}{|U|}\sum_{k\in U}\big(w_k(0) - \eta\sum_{t=0}^{\tau-1} G_k(t)\big) \\ &= \frac{1}{|U|}\sum_{k\in U} w_k(0) - \frac{1}{|U|}\eta\sum_{k\in U}\sum_{t=0}^{\tau-1} G_k(t) \\ &= w(0) - \eta G^{cum(\tau)}, \end{aligned} \tag{2.35}$$

where $w(\tau)$ is the aggregated model at time point τ and the last equality holds by ensuring a same base model $w(0)$ for all the clients before the first update is taken, i.e., $w_k(0) = w(0)$ for all $k \in U$.

Remark. *Aggregating local models is the standard practice in FL. Eq. (2.35) indicates that, to obtain an aggregated model (called the global model in FL), the coordinating PS can either directly aggregate local models or aggregate accumulated gradients first and then apply it to the latest consensus model. Besides, ensuring an identical base model for every τ steps of local training is naturally achieved by the PS that distributes the aggregated model (Eq. 2.35) back to the clients. The clients also typically have a common initial model at the very beginning of the whole process.*

Since model aggregation and gradient aggregation serve the same purpose, the following content of this subsection assumes that it is the (accumulated) local gradients that get uploaded to the PS for aggregation.

Without ambiguity, let $\tilde{G}^{cum(\tau)} \triangleq \sum_{t=0}^{\tau-1} \tilde{G}(t)$ representing the accumulated gradient in the centralised SGD chain. The expectation of $\tilde{G}^{cum(\tau)}$ can be formulated as:

$$\begin{aligned}\mathbb{E}[\tilde{G}^{cum(\tau)}] &= \mathbb{E}[\sum_{t=0}^{\tau-1} \tilde{G}(t)] \\ &= \sum_{t=0}^{\tau-1} \mathbb{E}[\tilde{G}(t)]. \end{aligned} \tag{2.36}$$

With the aggregated gradient $G^{cum(\tau)}$ given in Eq. (2.34), its expectation can be formulated as:

$$\begin{aligned}\mathbb{E}[G^{cum(\tau)}] &= \mathbb{E}[\frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} G_k(t)] \\ &= \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} \mathbb{E}[G_k(t)]. \end{aligned} \tag{2.37}$$

Combining Eq. (2.36) and (2.37), the expected difference between the aggregated gradient by multi-step SGD and the centralised gradient can be formulated as:

$$\begin{aligned}\mathbb{E}[G^{cum(\tau)} - \tilde{G}^{cum(\tau)}] &= \mathbb{E}[G^{cum(\tau)}] - \mathbb{E}[\tilde{G}^{cum(\tau)}] \\ &= \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} \mathbb{E}[G_k(t)] - \sum_{t=0}^{\tau-1} \mathbb{E}[\tilde{G}(t)] \\ &= \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} \mathbb{E}[G_k(t)] - \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} \mathbb{E}[\tilde{G}(t)] \\ &= \frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} \big(\mathbb{E}[G_k(t)] - \mathbb{E}[\tilde{G}(t)]\big) \end{aligned} \tag{2.38}$$

From the equation, one can discover that the fact that $G^{cum(\tau)}$ deviates from $\tilde{G}^{cum(\tau)}$ boils down to the difference (in expectation) between local gradients $\{G_k(t)\}_{k\in U}$ and the centralised gradient $\tilde{G}(t)$ at each time point before aggregation. Recall that the expectation of $\{G_k(t)\}_{k\in U}$ and $\tilde{G}(t)$ are given in Eq. (2.23) and (2.22) respectively for one SGD step, which can be rewritten with the notation of t for the case of multi-step SGD:

$$\mathbb{E}[G_k(t)] = \mathbb{E}_{\sim D_k}[g(w_k(t))], \tag{2.39}$$

and

$$\mathbb{E}[\tilde{G}(t)] = \mathbb{E}_{\sim D}[g(\tilde{w}(t))]. \tag{2.40}$$

Remark. *From the equations above and recalling the update rules, one can learn that generally* $\mathbb{E}[G_k(t)] \neq \mathbb{E}[\tilde{G}(t)]$ *for multi-step SGD update even with a common* *initial model $w(0)$. The reason behind is that $G_k(0) = \tilde{G}(0)$ cannot be guaranteed no matter the data are IID or not across the clients. Both gradients are random variables (due to the nature of SGD) and the IID assumption, if it holds, can only guarantee $\mathbb{E}[G_k(0)] = \mathbb{E}[\tilde{G}(0)]$. Consequently, $w_k(1) = \tilde{w}(1)$ cannot be guaranteed for any $k \in U$, which means $\mathbb{E}[G_k(t)] \neq \mathbb{E}[\tilde{G}(t)]$ since $t = 1$ because the gradients are computed w.r.t. different models. In other words, the local SGD chains and the centralised sequential SGD chain diverge at the second update.*

To facilitate further analysis of gradient divergence for multi-step SGD, Assumption 2.1 is made to bound the divergence of any local gradients in expectation. Similar assumptions are also made in the literature [55, 84–87].

Assumption 2.1 (Bounded local gradient divergence)**.** *For any clients in U and models $w_k(t)$ and $\tilde{w}(t)$ at time point t, there exists an upper bound δ of the divergence between $G_k(t)$ and $\tilde{G}(t)$ in expectation:*

$$\mathbb{E}\|G_k(t) - \tilde{G}(t)\| \leq \delta. \tag{2.41}$$

For multi-step SGD (equivalently, periodic model averaging as in FL) with a full cycle of τ local updates and one global aggregation, the following Theorem holds:

Theorem 2.3. *With Assumption 2.1 satisfied, the divergence between the aggregated gradient $G^{cum(\tau)}$ and the centralised gradient $\tilde{G}^{cum(\tau)}$ is upper bounded by $\tau\delta$:*

$$\mathbb{E}\|G^{cum(\tau)} - \tilde{G}^{cum(\tau)}\| \leq \tau\delta, \tag{2.42}$$

where τ is the number of SGD steps.

Proof. From the definitions of $G^{cum(\tau)}$ and $\tilde{G}^{cum(\tau)}$ and using triangle inequality, one can derive that

$$\begin{aligned}\|G^{cum(\tau)} - \tilde{G}^{cum(\tau)}\| &= \|\frac{1}{|U|}\sum_{k\in U}\sum_{t=0}^{\tau-1} G_k(t) - \sum_{t=0}^{\tau-1}\tilde{G}(t)\| \\ &= \frac{1}{|U|}\|\sum_{k=1}^{|U|}\sum_{t=0}^{\tau-1}\big(G_k(t) - \tilde{G}(t)\big)\| \\ &\leq \frac{1}{|U|}\sum_{k=1}^{|U|}\sum_{t=0}^{\tau-1}\|G_k(t) - \tilde{G}(t)\| \end{aligned} \tag{2.43}$$

Then by taking expectation of both sides and using Assumption 2.1, it follows that

$$\begin{aligned}\mathbb{E}\|G^{cum(\tau)} - \tilde{G}^{cum(\tau)}\| &\leq \frac{1}{|U|}\sum_{k=1}^{|U|}\sum_{t=0}^{\tau-1}\mathbb{E}\|G_k(t) - \tilde{G}(t)\| \\ &\leq \frac{1}{|U|}\sum_{k=1}^{|U|}\sum_{t=0}^{\tau-1}\delta \\ &= \tau\delta. \qquad (2.44)\end{aligned}$$

□

Theorem 2.3 gives the divergence upper bound for multi-step SGD update with τ as the aggregation interval. By revisiting the case of single-step SGD update, one can also derive the corresponding upper bound $\mathbb{E}\|G - \tilde{G}\| \leq \delta$ based on Assumption 2.1 from Eq. (2.21). By now one can discover the association between single-step SGD and multi-step SGD: the gradient divergence upper bound is relaxed by a factor of τ (i.e., the steps of update between two aggregations) using multi-step SGD update as compared to the single-step SGD update based on a consensus model.

The result of the analysis reveals that for gradient-based training, having each client/worker download the latest consensus model from the PS and limiting local SGD update to one step can minimise the gradient divergence (and thus achieve faster model convergence). This is the reason why most of the state-of-the-art distributed training methods (e.g., SyncSGD [43], AsyncSGD [52], DC-ASGD [42], etc.) use single-step SGD for each local update.

However, multi-step SGD update has its advantages in *training and communication efficiency*. Clients/workers perform gradient computation and model update for several mini-batches without communicating with the coordinating PS. This effectively saves network traffic in the system. In particular, for synchronous training approaches, reduced communication frequency means fewer synchronisation barriers and generally higher overall efficiency of the training process.

Remark. *The comparison between single-step SGD and multi-step SGD over decentralised data shows a trade-off between resource efficiency and model accuracy in practical distributed machine learning systems. Exchanging gradients and the model after each gradient computation (i.e., SGD update) theoretically yields the least error of gradients. However, the benefit diminishes in the case of non-IID data where Theorem 2.2 does not hold and the gradient divergence is more potentially unpredictable.*

This in some ways tips the scale towards the focus on communication efficiency and motivates the study of (horizontal) federated learning. FL is essentially a multi-step SGD paradigm with more flexibility in the number of steps, batch sizes as well as the optimisation algorithm for local update. ***The key idea of FL is to relax the gradient divergence in local training in exchange for significantly improved communication efficiency.*** *The advantages in communication efficiency can be crucial in many resource-limited circumstances (e.g., MEC and IoT) where model transfer takes much longer than local update on the training devices [80].*

2.1.3 Participation and Aggregation

This section introduces the ways clients get selected to participate in a round of training and subsequently how local models get aggregated into the global model.

Client participation can be flexible in an FL system. The standard algorithm FedAvg uses a fraction parameter C to control the proportion of clients allowed to take part in one round of training. In other words, given a set U of clients (which is usually assumed unchanged throughout the process or at least determined for any specific round), the parameter $C \in [0, 1]$ pre-set by the server defines the quota $C \cdot |U|$. With $C = 1$ the recruiting mechanism is called *full participation*, otherwise ($C < 1$) indicates *partial participation.* A special case where $C = 0$ is defined in [47], wherein only one client is selected for local training each round regardless of the size of the population.

Even though FL was initially devised with communication-efficient properties for mobile and edge devices [12], various application scenarios are emerging where the device population can range from a couple to more than a million. For this reason, both full participation and partial participation can make sense depending on the scenarios. For example, the comprehensive survey by Kairouz et al. [49] specifies two typical scenarios of FL: *cross-device FL* and *cross-silo FL.* Cross-device FL is in line with the original settings oriented to relative large population of participating devices, whilst the cross-silo FL is defined as performing FL over a small number of data silos each of which contains relatively large amount of data and is typically held proprietary by an organisation such as an e-business company. In the case of cross-silo FL, it is reasonable to allow full participation by all the parties because the clients are limited in numbers but sufficient in computational and network resources.

The operation of model aggregation is executed by the server and can also take two different forms: *full aggregation* and *partial aggregation.* Full aggregation refers to the aggregation rules that take into account all the clients when aggregating the their local models. Every local model accounts for a term in the aggregation rule no matter if the corresponding client participated in the round or not. Actually there are two ways to perform full aggregation. The first way is termed *keep-local aggregation* in this thesis and can be formulated as:

$$w^{(t)} = \sum_{k \in U} \frac{n_k}{n} w_k^{(t)}, \tag{2.45}$$

where $w^{(t)}$ denotes the global model produced at the end of round t. Formally, $w^{(t)} := w(tE)$ with tE being the final time step of round t and the same rule applies to $w_k^{(t)}$. The local models are weighted by the local dataset size n_k as the numerator and the total size n as the denominator. For brevity, the value $\frac{n_k}{n}$ is termed *relative data size* or *data ratio* in the following content. Note that the formula averages across all the local models but one does not necessarily need to have them uploaded. From the engineering perspective, the server can keep a cache of local models and every round of local training results in a partial or full update of the cache. Then server only needs to do aggregation based on the cache. This should be feasible as long as the old entries (unchanged local model files) are compatible to the new entries (updated local model files) when encryption or other privacy protection techniques apply. Details on how techniques such as Homomorphic Encryption and Differential Privacy may affect model aggregation are provided in the next section. Another full aggregation implementation is *keep-global* aggregation which, as opposed to *keep-local*, replaces the model entries of inactive clients with the existing global model when performing the aggregation. Here inactive clients refer to those which did not participate in the round. The rule of *keep-global* aggregation can be written as:

$$w^{(t)} = \sum_{k \in S(t)} \frac{n_k}{n} w_k^{(t)} + \sum_{k \notin S(t)} \frac{n_k}{n} w^{(t-1)}, \tag{2.46}$$

where $S(t)$ is the selected clients (here drop-outs are not taken into account) for round t and $w^{(t-1)}$ denotes the global model obtained at the end of round $t-1$, i,e., the consensus model at the start of round t. Technically, *keep-global* aggregation can also be realised via a cache on the server as long as the encrypted (by HE) or masked (by DP) models are cross-round compatible. The standard implementation of FL (i.e., FedAvg) adopts full aggregation but the authors did not clarify whether the inactive clients' local models are replaced in the aggregation step.

Partial aggregation, as it literally indicates, is a class of aggregation rules that only account for the updated local models in each round. Again, let $S(t)$ denote the set of clients that participate in round t and assume no crash or network failure. The general rule of partial aggregation is formulated as follows:

$$w^{(t)} = \sum_{k \in S(t)} a_k w_k^{(t)}, \tag{2.47}$$

where a_k is the weight of client k's model and generally should add up to 1 because the global model is expected to be at the same scale as the local models. For example, Li et al. [88] adopt partial aggregation in their approach and use $1/|S(t)|$ (where $|S(t)| = C \cdot |U|$ is a constant in the paper) to weight the local models submitted each round. From the theoretical perspective, the reason why they use equal weights is that they select $C \cdot |U|$ clients using weighted random with n_k/n being the probability for client k to be selected. Besides, the work [85] proposed a distinctive aggregation rule that weights each client model with its relative data size multiplied by a constant $\frac{|U|}{|S(t)|} = \frac{1}{C}$. The constant rescales the relative data sizes so that the sum of weights adds up to 1 in expectation. Comprehensive convergence analysis is also provided in their paper for all the three partial aggregation rules mentioned above.

A special case of partial aggregation is *instant* update of the global model, which is designed to support more flexible pace control in asynchronous FL algorithms. As opposed to the synchronous implementations of FL discussed above, asynchronous FL algorithms remove the barrier of each round and allow the server to update the global model upon the arrival of any local model. As an typical example, the *FedAsync* algorithm [71] periodically selects a random subset of clients and distributes the global model to them, after which the server launches an updater thread to keep listening and performs the following update whenever a local model is uploaded:

$$w^{(t)} = \big(1 - \alpha(t, t_k)\big) w^{(t-1)} + \alpha(t, t_k) w_k^{(t_k)}, \tag{2.48}$$

where $w_k^{(t_k)}$ is the client model trained based on a previous global model from some logical time point t_k before, and $\alpha(t, t_k)$ is a staleness function that adaptively determines the weights based on the difference between t_k and the current (logical) time

t. In the paper, the time stamp t is a server-side counter and increments whenever the global model gets updated. The biggest advantage of asynchronous FL is that it significantly reduces the interval between two global updates by making the model aggregation an incremental operation. But the downside of asynchronous approaches is also outstanding — the convergence of the global model largely depends on the form of the staleness function (which has tunable parameters) and thus can be less stable if the function is not properly set.

Table 2.1 categorises relevant studies of FL that adopt different participation and aggregation rules in their implementations. Details of these studies will be discussed in the next section.

Table 2.1: Categorised studies by participation and aggregation rules.

Refs.	Model aggregation	
	Full aggregation	Partial aggregation
Full participation	Wang et al. [55], Konečný et al. [81], Ma et al. [89], Mothukuri et al. [90], Truex et al. [91], Yang et al. [92], Ang et al. [93], Amiri and Gündüz [94], Shlezinger et al. [95], Qu et al. [96], Song et al. [97], FEDL [98], FedCOM [99], SlowMo [82], FedNova [100], EDM [101], FedMA [102], PEFL [103], FedPer [104], MOON [105], FedMD [106], NbAFL [107], L2GD [108], FedAwS [109], UVeQFed [110], HFL [111], MFL [112], FedRobust [113], C-DDPG [114]	Sprague et al. [72], Kall and Trabelsi [115], Ghosh et al. [116], Yang et al. [117], Lalitha et al. [118], So et al. [119], Chen et al. [120], Liu et al. [121], Chen et al. [122], CMFL [123], VerifyNet [124], FeSEM [125], HybridAlpha [126], CFL [127], FedMe [128], LG-FedAvg [129]
Partial participation	Bonawitz et al. [12], Ramaswamy et al. [16], McMahan et al. [47], Konečný et al. [130], Preuveneers et al. [131], Geyer et al. [132], Chen et al. [133, 134, 135], Jiang et al. [136], Chai et al. [137], Li et al. [138], FedCS [139], FedMom [140], SplitFed [141], Favor [142], FL+HC [143], huang2020RBCS-F [144], SAFA [1], HybridFL [2], FedProf [3]	Li et al. [85] (2019), Li et al. [145] (2020), Niu et al. [146], Yao et al. [147], Sattler et al. [148], Yang et al. [149], Zhu and Jin [150], Nguyen et al. [151], Pillutla et al. [152], Sun et al. [153], Bonawitz et al. [154], Tolpegin et al. [155], Shi et al. [156], Sarkar et al. [157], Ye et al. [158] AFL [159], FedAdam [160], FedAsync [71], FedFusion [161], FedProx [88], Power-of-Choice [162], *FedGLOMO* [163], SCAFFOLD [164], FedPAQ [165], MIME [166], MUSCS [167], APFL [168], TiFL [169], IFCA [170], Per-FedAvg [171], FetchSGD [172], FedBoost [173], FedDyn [174], Ditto [175], FedGRU [20], VIRTUAL [176], FedProf [3]

Note that the partial model aggregation may sound weird intuitively in the case of full client participation, but it is actually reasonable for algorithms with asynchronous nature, selective aggregation rules or failure-tolerant designs. For example, Dhakal et al. [127] Coded Federated Learning (CFL) that preemptively calculates an estimate of the gradient aggregate so that it allows the server to perform aggregation without having to receive all the local updates. Kall and Trabelsi [115] adopt an asynchronous FL algorithm for building code scanners wherein all devices are allowed to participate at any time. The server is designed to perform instant model update upon the arrival of any local update, which makes it a case of partial aggregation. A special case of partial aggregation under full participation is client clustering (e.g., [116, 125], where the clients perform local training, get clustered by some metrics and make contributions to their corresponding cluster models only.

In the table, SAFA, HybridFL and FedProf are devised by the author and will be introduced in details in Chapters 3, 4 and 5, respectively. Note that the proposed FedProf is applicable to both full aggregation and partial aggregation.

2.2 Industrial Practice

The boom of research on FL has stimulated the development of prototype FL systems, integrated FL platforms as well as a wide range of applications. In this section, first provided is a brief review of existing FL systems and open platforms designed and developed for FL. Then a number of emerging or potential application scenarios for FL (especially horizontal FL) are introduced.

2.2.1 Open Platforms and Systems for FL

Platforms/Libraries

The implementation of a realistic FL system requires a comprehensive collection of techniques spanning from networking and machine learning to encryption and scheduling. It is usually not plausible for individuals or small groups of researchers to build such a system from scratch. Fortunately, a number of open platforms and libraries have been developed and released.

TensorFlow Federated

`TensorFlow Federated (TFF)`[2], maintained by the `TensorFlow`[3] community, is an open-source framework[4] developed for facilitating research and experimentation of federated learning algorithms and protocols. `TFF` was built on top of the TensorFlow engine and provides a rich set of interfaces enabling developers to run the FL algorithms, which can be built-in or customised, with their own models and data. It is worth mentioning that the `TFF` provides two levels of APIs (fig. 2.3), Federated Learning (FL) API and Federated Core (FC) API, to facilitate both easy plug-in of user models for basic tasks and lower-level flexibility for customised FL algorithm reproduction. `Keras` is the recommended base framework for model implementation on `TFF`.

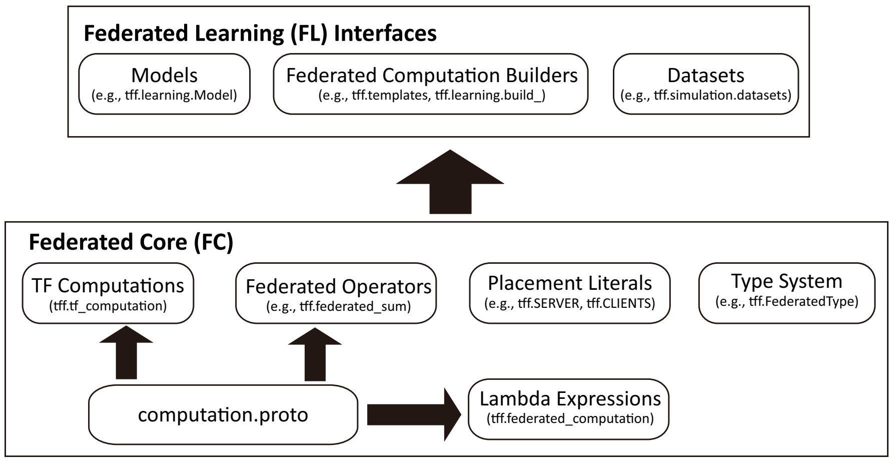

Federated Learning (FL) Interfaces
Models
(e.g., tff.learning.Model)
Federated Computation Builders
(e.g., tff.templates, tff.learning.build_)
Datasets
(e.g., tff.simulation.datasets)
Federated Core (FC)
TF Computations
(tff.tf_computation)
Federated Operators
(e.g., tff.federated_sum)
Placement Literals
(e.g., tff.SERVER, tff.CLIENTS)
Type System
(e.g., tff.FederatedType)
computation.proto
Lambda Expressions
(tff.federated_computation)

Figure 2.3: The architecture of `TensorFlow Federated`. (based on the official documentations and the author's understanding)

PySyft

`PySyft`[5] is a Python library for secure and private Deep Learning. `PySyft` integrates the commonly used privacy-preserving techniques for FL such as Homomorphic Encryption and Differential Privacy. The library also provides a bunch of interfaces for performing federated learning over data of different owners securely. Note that for complete implementation of FL, `PySyft` needs to be used within a main machine learning framework such as `TensorFlow` or `PyTorch`[6].

FATE

`FATE`[7], developed and maintained by WeBank, is an industrial-grade, open-source project that provides a secure computing framework to support the federated AI

[2]`https://www.tensorflow.org/federated/get_started`
[3]`https://www.tensorflow.org/`
[4]GitHub link: `https://github.com/tensorflow/federated`
[5]GitHub link: `https://github.com/OpenMined/PySyft/blob/dev/packages/syft/`
[6]`https://pytorch.org/`
[7]`https://fate.fedai.org/`

ecosystem. FATE provides developers with out-of-the-box usability and built-in secure computation protocols. It also features scalable modelling pipeline, clear visual interface and flexible scheduling system. Researchers and developers can perform agile experiments with pre-configured tasks like logistic regression, tree-based algorithms, deep learning and transfer learning.

FATE itself has actually developed into a comprehensive ecosystem that encompasses a pipelined high-performance FL system (FATE-serving) for production environment, a module-based FL library (FederatedML) for developers as well as a bunch of tools for building key components (e.g., networking, containerisation and visualisation) of an FL system. The FederatedML library[8] in the FATE project is developer-oriented as it offers standard implementations of common algorithms and utilities in the form of APIs. The architecture of the FederatedML library is illustrated in Fig. 2.4.

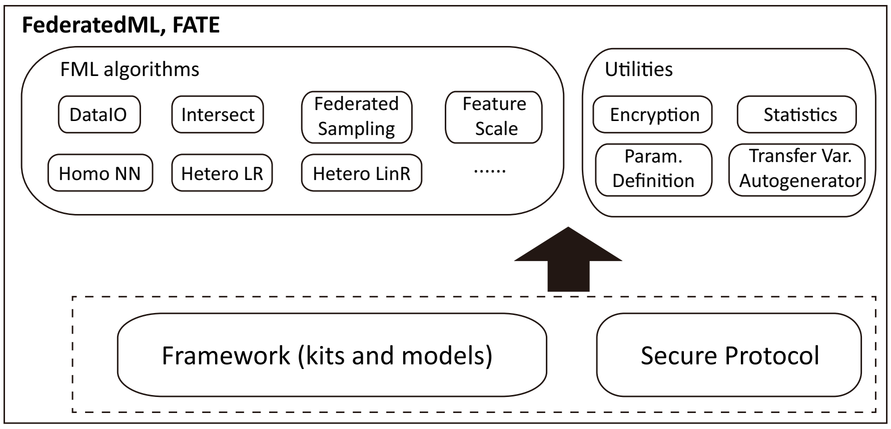


Figure 2.4: The architecture of FederatedML, FATE (based on the official documentations and the author's understanding)

FedML

FedML[9] is an open-source library for FL research and benchmarking [11]. The library follows a versatile design and supports three different running mode: FL over edge devices, distributed computing and standalone simulation. It also integrates a variety of FL applications and state-of-the-art FL training algorithms. The library is developed based on PyTorch as its training engine and uses MPI and MQTT for its main backend module for communication. FedML also features cross-platform compatibility with support of both mobile devices (Android and iOS) and IoT devices such as NVIDIA Jetson Nano and RaspBerry Pi. Deployment on these specific hardware requires two extension frameworks, FedIoT and FedMobile, on top of FedML

[8] https://fate.fedai.org/federatedml/
[9] https://fedml.ai

(please refer to the official documentations for more details). Fig. 2.5 shows a structural overview of the `FedML` library.

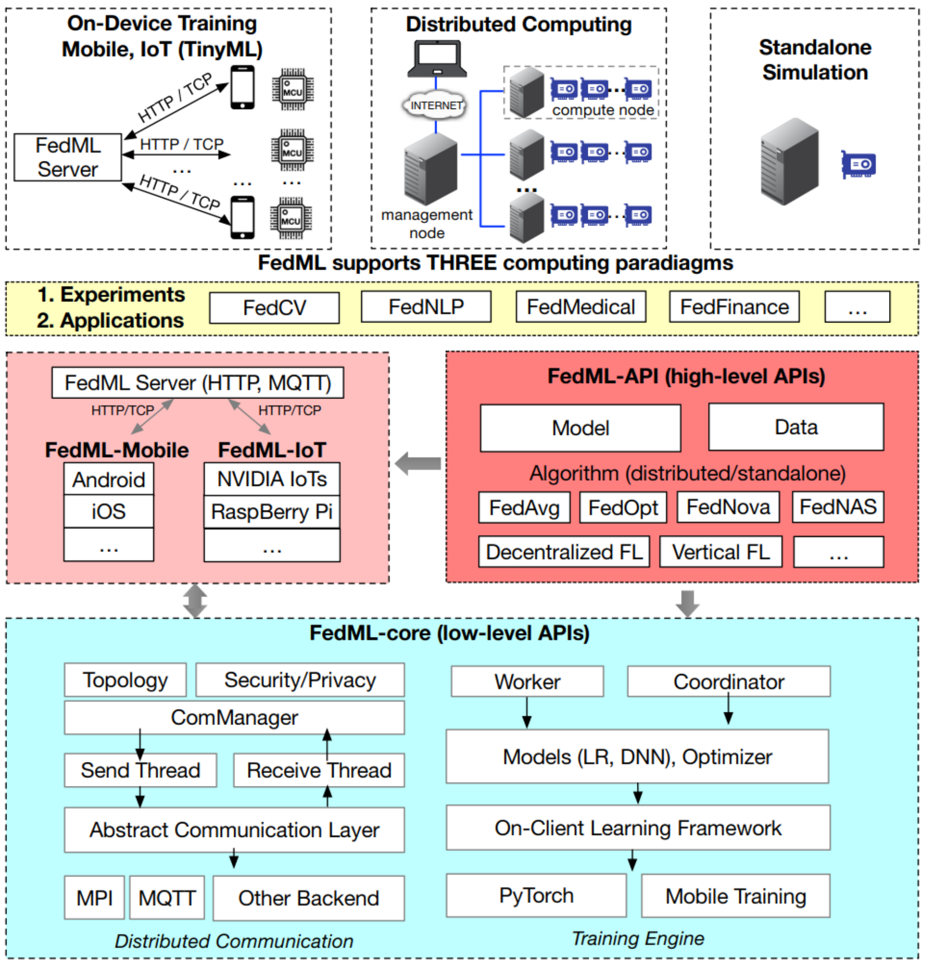


Figure 2.5: The architecture of `FedML`. (Source: He et al. [11])

An ecosystem has been built around `FedML` where a variety of domain-specific system frameworks are provided. These systems use `FedML` as the core library and mainly focus on models and datasets for specific tasks such as Natural Language Processing (NLP), graph processing and Computer Vision (CV), or specific environments such as IoT and MEC. These extension frameworks will be detailed later.

PaddleFL

`PaddleFL`[10], or Paddle Federated Learning (PFL), is an open-source FL framework developed on the basis of the scalable deep learning platform `PaddlePaddle` [177]. `PaddleFL` can be used to build easy-to-deploy horizontal FL systems over a large-scale cluster or a two-party vertical FL system. Along with built-in FL algorithms, it also

[10]`https://github.com/PaddlePaddle/PaddleFL`

provides a wide range of applications from CV and NLP to recommendation tasks and elastic scheduling on Kubernetes.

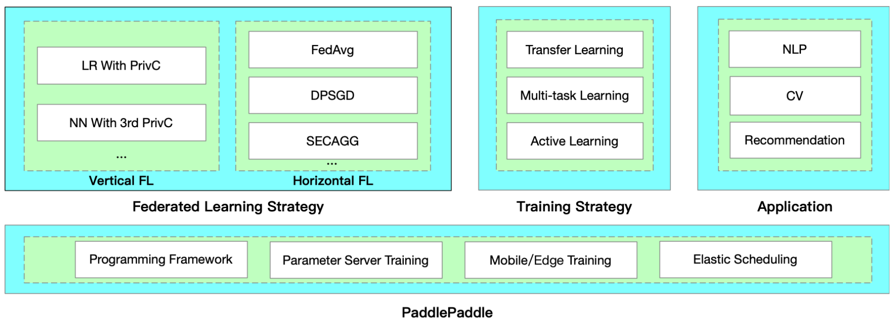


Figure 2.6: An overview of the `PaddleFL` framework. (Source: `PaddleFL` documentation at `https://paddlefl.readthedocs.io/en/stable/`)

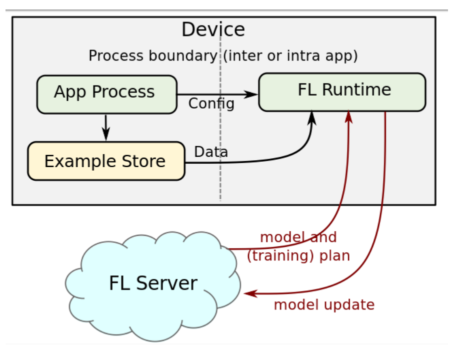
Frameworks/Systems

Though we have seen the emergence of both research-facilitating platforms (discussed above) and conceptual FL system designs [121, 178, 179], production FL systems *at scale* are still scarce due to the underlying engineering complexity. Only Google and a few tech giants have the user coverage and social influence to facilitate and benefit from a large-scale FL system.

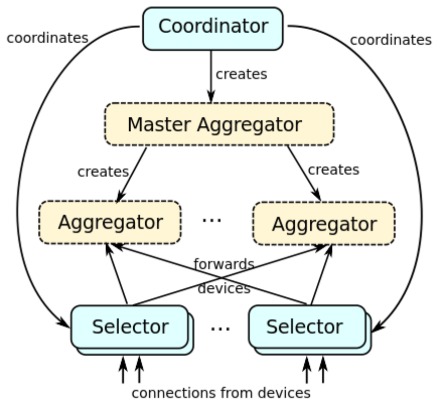
Google

Google built a scalable production system for federated learning and introduced its high-level designs in their published paper *Towards federated learning at scale: System design* [12]. The system was developed based on TensorFlow in the domain of mobile phones. According to the authors, the system has reached the standard of production with capability to run federated learning over tens of millions of real-world devices. The system uses a learning algorithm that basically follows FedAvg [47] and also incorporates several engineering techniques and tricks to address practical issues. For instance, a mechanism called *pace steering* is adopted along with partial participation to control the pattern of device connections. An *FL plan* needs to be generated for each task and the plan describes the model (in the TensorFlow graph format) and configuration for on-device training as well as the aggregation logic for the server. Figs. 2.7a and 2.7b depict the software architectures for the device side and the server side, respectively.

FATE-Serving



(a) device architecture



(b) server architecture

Figure 2.7: The software architecture of Google's FL system implementation. (Bonawitz et al. [12])

Within the `FATE` ecosystem, `FATE-Serving`[11] is a highly-integrated FL system that supports federated learning and online reasoning in production environments. `FATE-Serving` not only provides out-of-the-box models, datasets and FL algorithm implementations, but also has a built-in pipeline that connects basically all the inference procedures from post-processing to A/B testing. Fig. 2.8 shows the overall architecture.

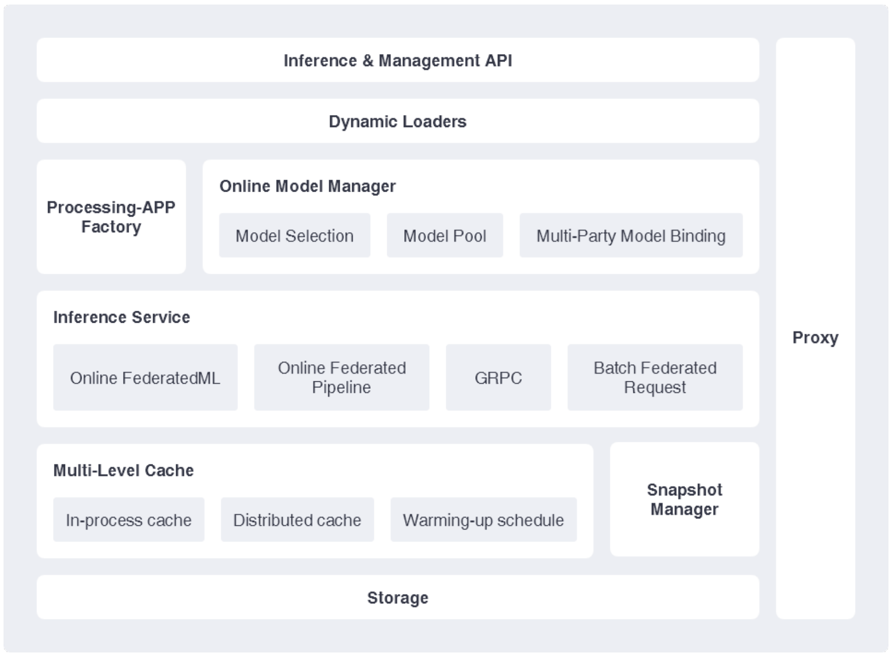


Figure 2.8: The architecture of the `FATE-Serving` system. (Source: official documentation at `https://fate.fedai.org/fate-serving/`)

`FATE-Serving` features a centralised job scheduling subsystem `FATE-Flow`, which

[11]`https://fate.fedai.org/fate-serving/`

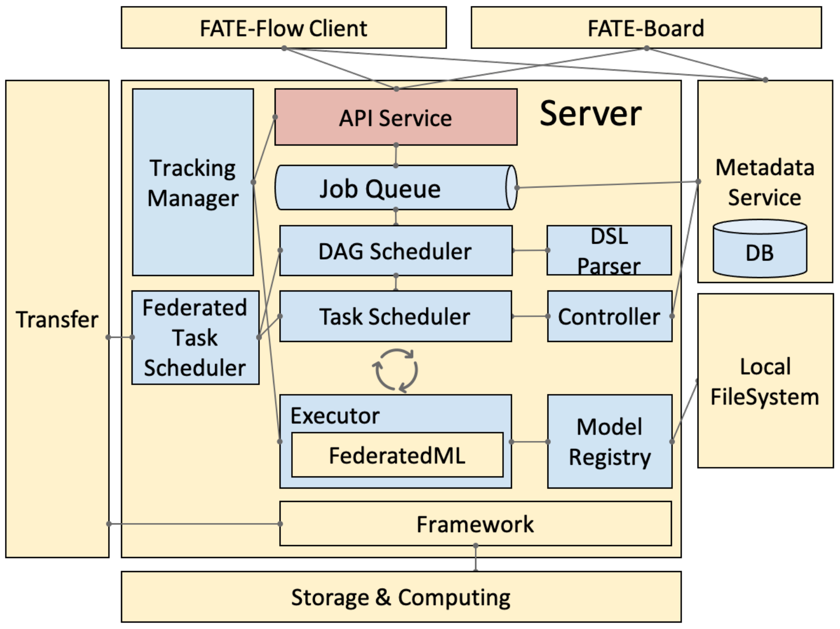


Figure 2.9: The architecture of the `FATE-Flow` scheduling subsystem. (Source: official documentation at `https://github.com/FederatedAI/FATE/tree/master/python/fate_flow`)

manages the entire federated learning job cycles including data pre-processing, feature engineering, statistics tracking, fine-grained task scheduling and resource allocation. According to the official documentation, `FATE-Flow` now supports both single-party scheduling and multi-party coordinated scheduling of FL jobs described in the form of Directed Acyclic Graphs (DAGs). Fig. 2.9 shows the architecture of the scheduling subsystem `FATE-Flow`. Trained models can be published via the model registry module on the server and be tracked through HTTP APIs. In addition, the `FATE-Serving` system provides an online inference pipeline based on the gRPC protocol that allows for request-driven remote reasoning.

Domain-specific Frameworks

FedNLP

`FedNLP` is an open-source framework built on top of `FedML` specifically for research on NLP tasks under FL settings. The framework has been released as a GitHub repository and officially introduced in [180]. 10+ benchmark datasets for experimental evaluation on NLP tasks are provided within the framework in a pre-partitioned format. The overall structure of `FedNLP` is shown in Fig. 2.10.

FedGraphNN

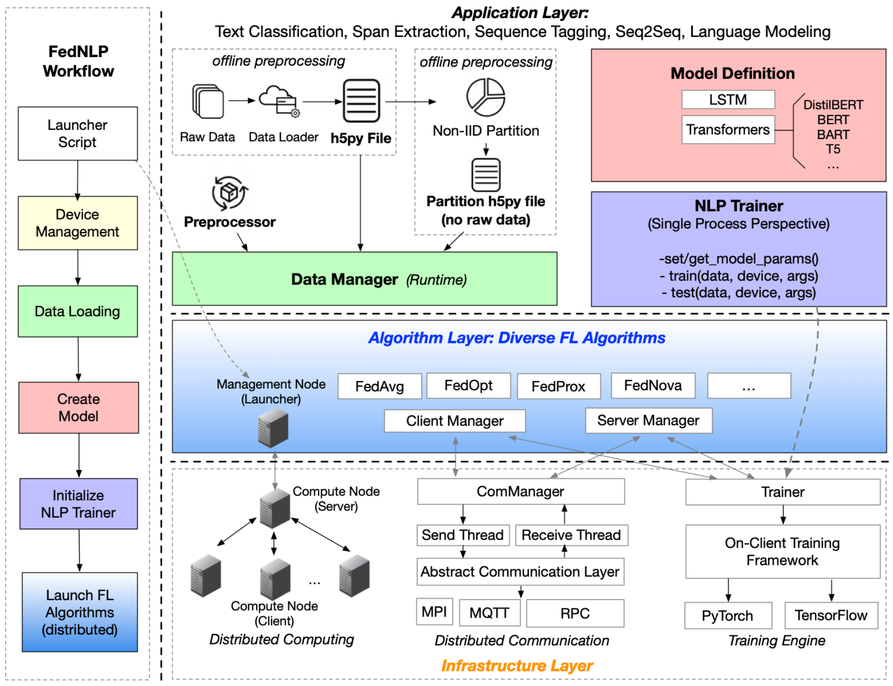


Figure 2.10: The structure of the FedNLP framework. (Source: GitHub documentation at https://fate.fedai.org/fate-serving/)

FedGraphNN was built on top of the core APIs of FedML. It is an open framework for facilitating research on Graph Neural Networks (GNNs) in federated learning systems. A number of benchmark datasets are also incorporated covering several molecular ML tasks in quantum mechanics, physical chemistry, biophysics and physiology domains. The framework, federated GNN algorithms and the datasets are detailed in the paper *FedGraphNN: A Federated Learning System and Benchmark for Graph Neural Networks* [13].

FedCV

FedCV is another FedML-based, research-oriented FL framework specifically developed for evaluating and benchmarking FL algorithms on computer vision tasks. At the time of writing, four popular datasets and six state-of-the-art models are integrated within the framework for three categories of CV tasks (i.e., image classification, image segmentation and object detection), together with a non-IID pre-processing module (implemented based on Latent Dirichlet Allocation). These datasets will be introduced in Section 2.4. Fig. 2.12 illustrates the overall structure of FedCV.

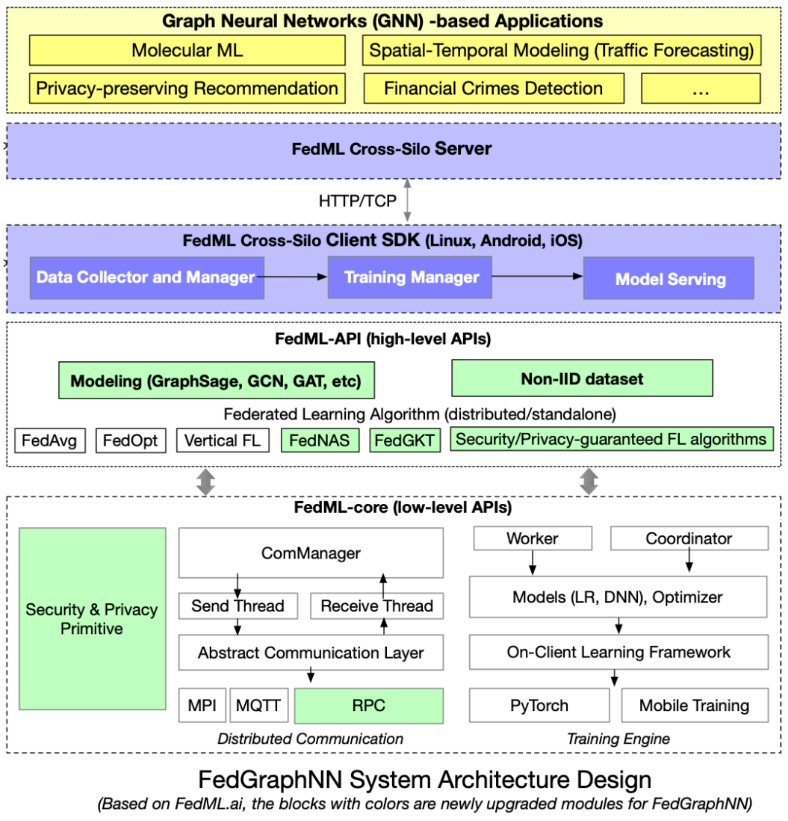

Graph Neural Networks (GNN) -based Applications
Molecular ML
Spatial-Temporal Modeling (Traffic Forecasting)
Privacy-preserving Recommendation
Financial Crimes Detection
...
FedML Cross-Silo Server
HTTP/TCP
FedML Cross-Silo Client SDK (Linux, Android, iOS)
Data Collector and Manager
Training Manager
Model Serving
FedML-API (high-level APIs)
Modeling (GraphSage, GCN, GAT, etc)
Non-IID dataset
Federated Learning Algorithm (distributed/standalone)
FedAvg
FedOpt
Vertical FL
FedNAS
FedGKT
Security/Privacy-guaranteed FL algorithms
FedML-core (low-level APIs)
Security & Privacy Primitive
ComManager
Send Thread
Receive Thread
Abstract Communication Layer
MPI
MQTT
RPC
Distributed Communication
Worker
Coordinator
Models (LR, DNN), Optimizer
On-Client Learning Framework
PyTorch
Mobile Training
Training Engine
FedGraphNN System Architecture Design
(Based on FedML.ai, the blocks with colors are newly upgraded modules for FedGraphNN)

Figure 2.11: FedGraphNN architectural design. (Source: He et al. [13])

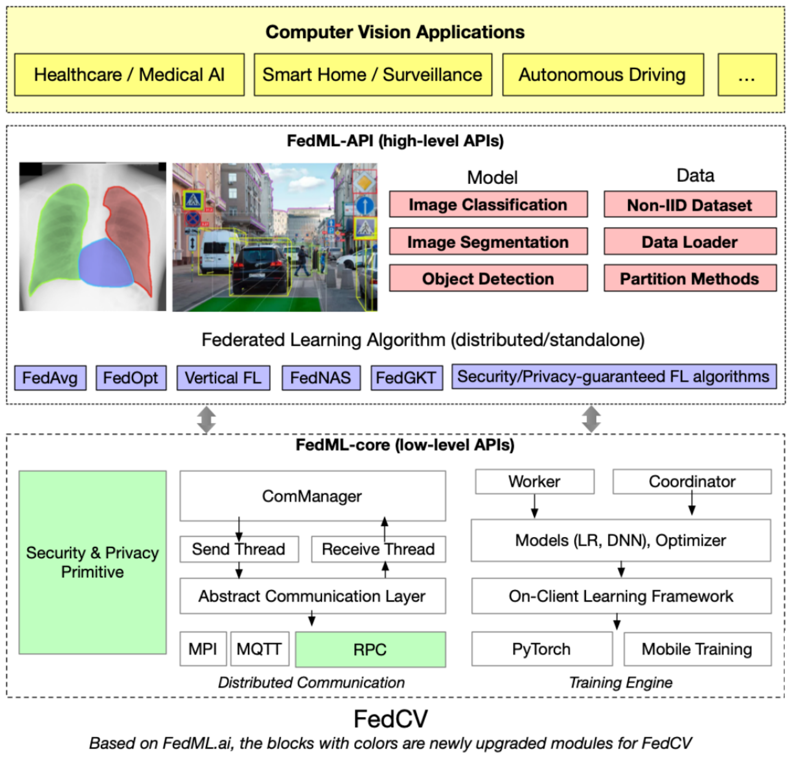

Computer Vision Applications
Healthcare / Medical AI
Smart Home / Surveillance
Autonomous Driving
...
FedML-API (high-level APIs)
Model
Data
Image Classification
Non-IID Dataset
Image Segmentation
Data Loader
Object Detection
Partition Methods
Federated Learning Algorithm (distributed/standalone)
FedAvg
FedOpt
Vertical FL
FedNAS
FedGKT
Security/Privacy-guaranteed FL algorithms
FedML-core (low-level APIs)
Security & Privacy Primitive
ComManager
Send Thread
Receive Thread
Abstract Communication Layer
MPI
MQTT
RPC
Distributed Communication
Worker
Coordinator
Models (LR, DNN), Optimizer
On-Client Learning Framework
PyTorch
Mobile Training
Training Engine
FedCV
Based on FedML.ai, the blocks with colors are newly upgraded modules for FedCV

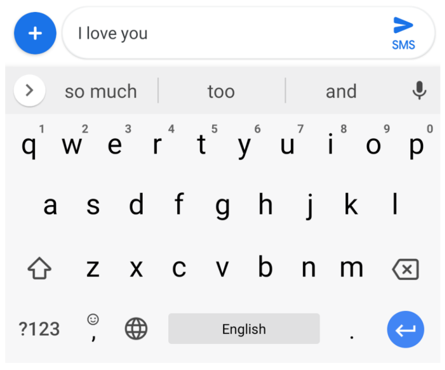
Figure 2.12: FedCV architectural design. (Source: He et al. [14])

2.2.2 Application Scenarios

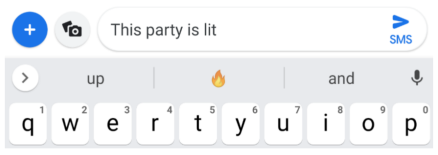
FL has great potential to drive a diversity of real-world applications including navigation optimisation, content suggestion [12], health care (e.g., NVIDIA Clara Imaging[12]) and service recommendation [181]. This section introduces a number of representative applications driven by FL.

Commercial Applications

As the initiator of FL, Google has created or optimised quite a few commercial applications through the deployment of federated learning. The most famous example is Google Keyboard (`GBoard`), which is claimed to have been optimised over a global scale of user population. Researchers and engineers of Google first targeted at the three basic features of `GBoard`: next-word prediction, auto-correction and word completion. As revealed in [182, 183], the old-version prediction mechanism for `GBoard` was based on an n-gram finite state transducer (FST) and a static Katz smoothed Bayesian interpolated 5-gram language model. To improve the prediction quality with constrained memory and data, they switched to Coupled Input-Forget Gates (CIFG) for the language model and deployed the application in an FL setting to realise large-scale training in their live production environments. The training process involved 1.5 million clients in North America over a course of 3000 FL rounds (4–5 days), as reported in their paper [15].



(a) Next-word predictions



(b) Emoji predictions

Figure 2.13: `GBoard` virtual input assistant. (Source: Hard et al. [15] and Ramaswamy et al. [16])

The adoption of FL for user experience enhancement is not just limited to the next-word prediction function in `GBoard`. Research efforts have also been made in

[12]`https://developer.nvidia.com/clara-medical-imaging`

dealing with out-of-vocabulary words and emoji predictions [16, 134]. Further, Google optimised the use case of search query suggestions for `GBoard` by integrating an *triggering model* trained on the user behaviour data under the federated learning framework [17]. The *triggering model* learns from users' reaction to the suggested queries (generated by a baseline model, stored locally in a SQLite database) and acts as an filter to keep useful queries only. This optimised search query suggestion solution is depicted in Fig. 2.14.

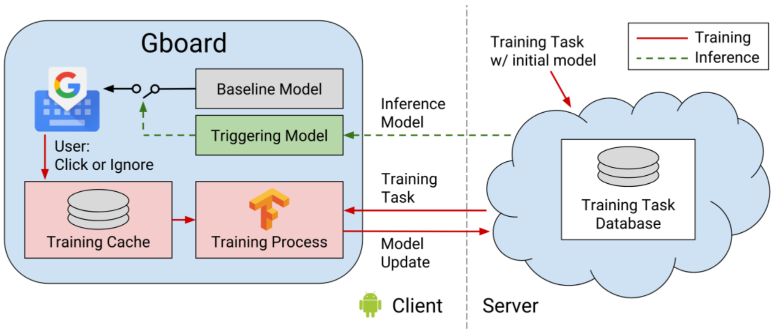


Figure 2.14: `GBoard` search query suggestion architecture. (Source: Yang et al. [17])

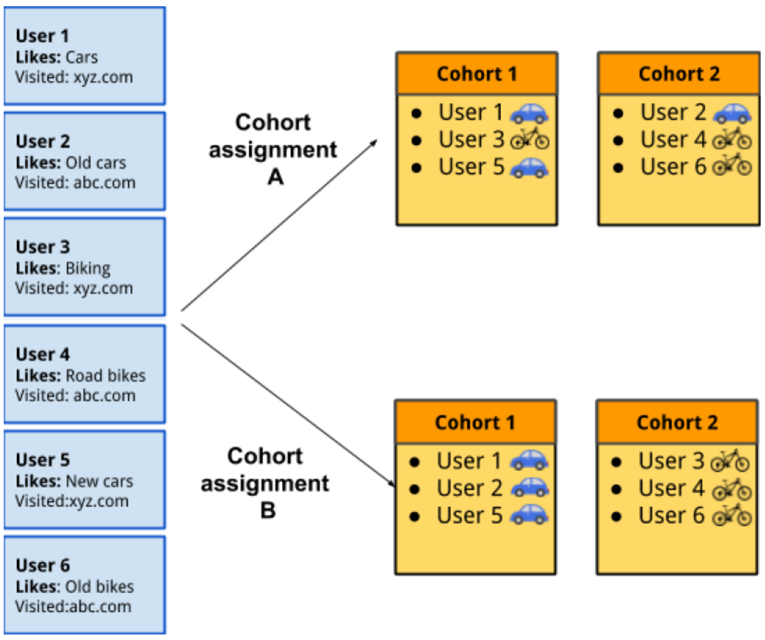


Figure 2.15: An example of how FLoC groups users into cohorts to guarantee k-anonymity ($k = 3$ in this example). The grouping is based on the 'likes' of users according to their browsing history. For more details, see the FLoC whitepaper. (Source: Ravichandran and Vassilvitskii [18])

Apart from typing assistant applications, Google has also made ambitious attempts to replace the 'reigning' third-party cookies with a new cohort-based user grouping technique called FLoC [184]. FLoC is designed for better privacy protection of users' browsing behaviours by means of hashing, clustering and FL so that individual users' properties are less exposed (see Fig. 2.15 for an example). Google Research & Ads realised a whitepaper titled *Evaluation of Cohort Algorithms for the FLoC API* [18] where part of the technical details for the FLoC technique is revealed. As an addition to the proposals, the adoption of federated learning is expected in the future to address the privacy concerns of the clustering-based algorithms introduced by FLoC.

With the rising concerns of user privacy, more and more commercial applications, such as content recommendation and advertising, may have to follow the paradigm shift from mining centrally stored data to learning over decentralised data. Federated learning has a great potential in leading the trend.

Healthcare and Medicine

Among all the potential applications, healthcare and medical informatics draw the most attention [185, 186]. This is mainly because clinic data are very sensitive whilst biomedical analytics heavily relies on colossal volumes of data for healthcare applications [59]. Federated learning enables inter-institutional collaboration for such applications and is thus believed to be the future direction for biomedical research [187].

Substantial evidences for FL's efficacy in training diagnostic models have been reported by Sheller et al. [19] in their article published in *Scientific Reports, Nature*. The authors evaluated different model training approaches on a medical imaging task of brain tissue labelling. Fig. 2.16 outlines their federated learning based approach. The experiments were conducted using multi-institutionally collected glioma data from the BraTS 2017 dataset plus clinically-required brain tumor MRI scans from the University of Texas MD Anderson Cancer Center (MDACC) and the Washington University School of Medicine in St. Louis (WashU). Their empirical results report that federated learning can achieve a competitive model quality (measured by Dice Similarity Coefficient [188]) to centralised training.

Security and Smart City

Federated learning can also have catalytic impact on corporate applications where a joint model is much needed but data sharing is prohibited between multiple organisations. Typically cross-silo FL applications include cyber security [90], smart city [20],

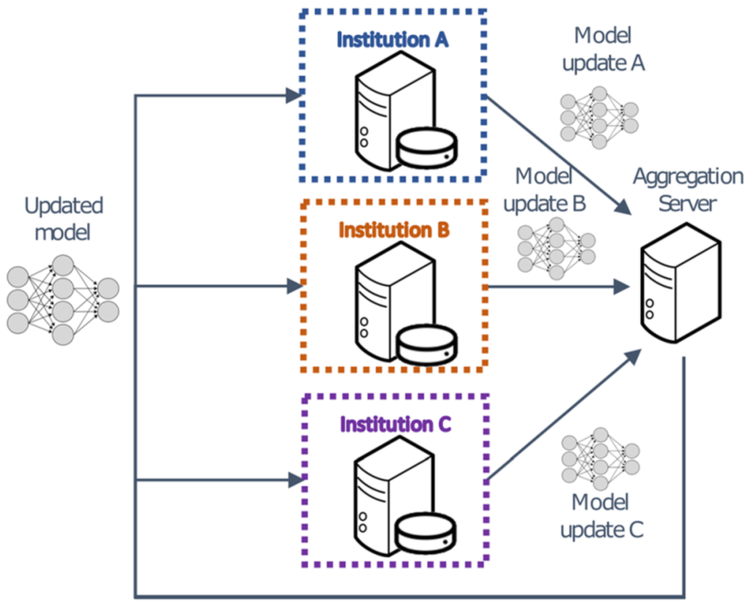


Figure 2.16: The FL-based, data-private collaborative training approach evaluated in [19].

intelligent surveillance [189, 190] and health informatics [186].

Security is a long-term concern across multiple domains whilst conventional data-sharing solutions usually achieve security by compromising privacy. In this aspect, federated learning can be a promising next-generation solution. For example, existing cyber-security companies are working independently on their own risk detection models using their proprietary data; but with FL, they can probably find a way to enhance every party's model through collaborative training without revealing their own data. The German corporation SAP developed a code scanning tool named Credential Digger [115], which adopts asynchronous FL to train a sensitive code token detection model over private code repositories. As an increasing number of risk, leakage and anomaly detection methods are model-driven and built on a large amount of log data, federated learning may well be the key technology for next-generation security solutions on the basis of cross-organisational collaboration.

Learning from big data is a critical part of realising smart cities where many everyday applications such as delivery service (e.g., UberEats[13]) can be optimised with FL-based technologies [158, 191]. Traffic management is a long-time predicament for metropolitan development and also a principal topic in the course of realising smart cities. Liu et al. [20] proposed a traffic flow prediction (TFP) solution FedGRU using the Gated Recurrent Neural Network (GRU) as the prediction model. To address

[13]https://eng.uber.com/michelangelo-machine-learning-platform/

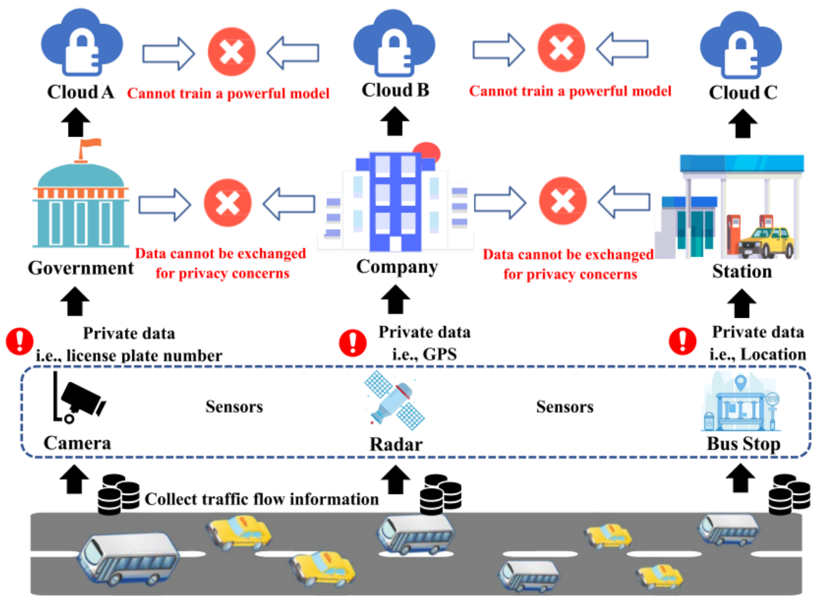

Cloud A
Cannot train a powerful model
Cloud B
Cannot train a powerful model
Cloud C
Government
Data cannot be exchanged for privacy concerns
Company
Data cannot be exchanged for privacy concerns
Station
Private data i.e., license plate number
Private data i.e., GPS
Private data i.e., Location
Camera
Sensors
Radar
Sensors
Bus Stop
Collect traffic flow information

Figure 2.17: Different sources of traffic data kept by separate organisations. (Source: Liu et al. [20])

the problem of 'data islands' in TFP (Fig. 2.17), they built a privacy-preserving training framework based on an improved federated learning algorithm via client clustering. The developers also adopt a joint announcement protocol to coordinate the participating organisations in each round of training.

2.3 Related Studies

This section reviews the relevant studies of federated learning including the distributed machine learning algorithms that cornerstone the development of FL, state-of-the-art algorithms and protocols for horizontal FL, variants of FL paradigms for specific learning problems or network topologies, communication optimisation in resource-limited FL systems, privacy protection and anti-adversary solutions for FL, and real-world application scenarios of FL.

2.3.1 Learning from Decentralised Data

The roots of machine learning are entangled with the studies on convex optimisation problems. Among the big family of optimisation algorithms [192–194], gradient-based training methods are arguably the most important building blocks of modern machine learning (especially deep learning) models. In the meantime, the increasing volume of data needed for training deeper and wider models has motivated the research on parallelising gradient-based model training over multi-core systems (e.g., IBM Power9 [195]), GPU clusters or inter-connected machines. Therefore, it is necessary to first discuss the algorithms for parallel and distributed optimisation.

Stochastic gradient descent is naturally suited for data parallelism where the process of inference (for example, forward propagation through layers in a neural network) for each sample within a mini-batch of data runs in parallel and so does the back propagation [196, 197]. Zinkevich et al. [198] discussed a very simplistic approach to performing distributed SGD over a multitude of machines. Their training algorithm only requires a 'one-shot' averaging of models that are trained locally by separate workers using SGD. Shamir et al. [63] proposed a distributed optimisation algorithm based on an approximate Newton-like method (DANE), which takes multiple synchronous steps to gather local gradients, broadcast the gradient of the global objective function and perform model aggregation. DANE performs local updates with second-order gradient information (Hessian) but does not need to exchange it. Mann et al. [65] discussed three distributed SGD-based training algorithms including synchronous gradient computation, majority vote and mixture weight methods. With conditional maxent models the training algorithms were evaluated in terms of prediction accuracy and CPU and memory consumption. Note that the mixture weight method is essentially the same as the one-shot averaging method [198] and the parameter mixing method [62]. They incur very little network usage but potentially lead to a biased final model over non-IID data.

Asynchronous approaches have also drawn lots of attention due to better scalability and flexibility [52]. For example, ASAGA [67] is a distributed version of the SAGA [199] optimiser with modifications to the update rule for gradient sparsification and an asynchronous online algorithm for shared-memory architectures with potential read inconsistency. Reddi et al. [200] proposed a generic framework for running variance-reduced gradient-based optimisation (including SAGA and GD) on multiple processors in parallel. Each processor works asynchronously on four key operations: read, read schedule iterate, update and schedule update. The schedule here refers to the historical gradients/weights stored in a shared memory. Variance reduction is an effective method for optimising local SGD by eliminating the gradient variance among workers. Liang et al. [201] introduced VRL-SGD, a distributed update scheme that has each worker track the average global–local gradient deviation which is used to rectify stochastic gradients of local objectives. Duchi et al. [61] studied the optimisation problem over sparse data and proposed an asynchronous training algorithm named AsyncDA. This method shares the dual vector (i.e., the accumulated gradient) among the workers each of which works out the model parameters before gradient computation and update. Another example of fully asynchronous update scheme is called Hogwild! [68], which is a lock-free distributed training method that sparsifies the local SGD updates by computing gradients with regard to the subvectors instead of the full vector of the central model's parameters. On the contrary, Langford et al. [69] constrained the update order of data processors to a round-robin fashion which allows every processor to access the latest model parameters (so that gradients can be computed) but only applies the gradient after a fixed delay (i.e., number of updates). Agarwal and Duchi [202] adopted a similar delayed SGD approach using a master node for model distribution. Though theoretical proof has been provided to support the convergence, delayed update can still be 'poisoning' for the asynchronous setting. To address it, researchers came up with delay-compensated update schemes [42, 70] that 'compensate' the out-of-date gradients or losses (from slow workers) via mathematical approximation or prediction before being applied to update the model.

Synchrony in distributed training avoids stale gradient poisoning but can be slow in practice due to device heterogeneity. Nonetheless, various mechanisms can be applied to make synchronous training more efficient. Chen et al. [43] figured out the negative impact of stale gradients caused by asynchrony and introduced backup workers into the standard implementation of synchronous SGD for straggler mitigation. By experiments they demonstrated faster and better convergence than the asynchronous optimisation. Synchronous implementations can also be flexible. The DC-S3GD algorithm [64] uses a stale-synchronous mechanism that parallelises gradient computation and the non-blocking version of `AllReduce` operation. The algorithm also incorporates a step for gradient correction with the distance between local update and the average update. Zhang et al. [51] introduced a proximal term (as the distance between each local model and a center model) to the overall objective function so that both types of models can update according to their respective gradients. The scheme, namely Elastic Averaging SGD (EASGD), was further extended by the authors for both synchronous training and asynchronous training.

In terms of system design, Agarwal et al. [66] developed a scalable linear learning system compatible to large-scale data storage systems with MapReduce logic such as

Hadoop[14] and provides the MPI–style `AllReduce` operation[15] for running gradient-based optimisation algorithms (such as GD and L-BFGS [203]) in parallel. Similarly, Distributed GraphLab [204] is a machine learning and data mining framework built based on a shared-memory system. It supports a high degree of parallel performance by virtue of its distributed data graph representations and distributed lock engine.

2.3.2 Horizontal Federated Learning

Horizontal federated learning was (and in a sense still is) the de facto implementation of FL before a number of variants evolved from the original framework. The horizontal FL paradigm stems from the traditional data-parallel distributed learning paradigm (which has been extensively studied in the last decade as discussed in the last section), where the data are distributed across multiple workers (namely clients in the context of FL) and the local data sets i) are non-overlapped or have little overlap in sample IDs, ii) share the feature space, i.e., have identical sample attributes, and iii) have labels for their own samples.

In this section, relevant studies on optimising horizontal FL are reviewed from eight perspectives.

Update rules

The process of FL entails frequent local model update (i.e., client-side update) and less frequent global model update (i.e., server-side update). Generally, the clients and server need to exchange intermediate results, such as stochastic gradients or models, so as to benefit from each other's update. Both types of update are vital to the performance of FL in terms of the global models' convergence and local models' utility.

A branch of studies, inspired by distributed learning algorithms [200, 201], are focused on refining the update rules for FL. The common motivation behind is to realise variance reduction in each update step by introducing server-side incremental update [71, 99], local and/or global momentums [82, 112, 140, 160, 163, 172, 174] or general control variates (i.e., state variables that facilitate the update of models) [166]. These methods typically have improved convergence guarantees towards heterogeneous data distribution.

Special update rules are also employed in many other approaches. To compensate

[14]`http://hadoop.apache.org/`
[15]`https://www.open-mpi.org/`

the missing updates from inactive clients, Dhakal et al. [127] use a novel gradient aggregation formula that encompasses two types of partial gradients that respectively come from participating devices over the network and the composite parity data shared by the clients. FedBoost [173] replaces the traditional consensus model on the server with an ensemble model built upon a group of pre-trained base models, wherein the corresponding update rule is devised for finding the optimal ensemble coefficients. Wang et al. [102] studied the permutation invariance properties for popular neural networks and proposed a layer-wise model aggregation strategy that aligns the neurons (for MLPs), channels (for CNNs) or states (for RNNs) before averaging the parameters.

Objective adaptation

All gradient-based optimisation algorithms are tightly coupled with the form of the objective functions (also termed empirical risk in the literature), which is intuitive because gradients, subgradients and Hessians are defined based on the objective function with regard to the model that we aim to optimise. In the case of horizontal FL, the global objective (see the previous section for the standard form) is defined as a weighted average of the local objectives. Every participating client takes the responsibility of local training, e.g., using SGD or momentum methods, with the aim of minimising its local objective function [47, 81, 85].

Many researchers have investigated the relation between the global objective $F(w)$ and the local objectives $\{F_k(w)\}_{k\in U}$ of FL and figured out their inconsistency over heterogeneous data [88, 100, 102, 116], which slows down the convergence of the global model and compromises its accuracy [101, 137]. Essentially, this is caused by the discrepancy between the local data distribution $\mathcal{D}_k$ and the population distribution $\mathcal{D}$, which consequently yields divergent local optima in the parameter space (as illustrated in Fig. 2.18) or mathematically,

$$\underbrace{\arg\min_{w} F(w)}_{w^*} \neq \underbrace{\arg\min_{w_k} F_k(w_k)}_{w_k^*},$$

and typically the optimal combination of local optima is not aligned with the aggregation rule:

$$w^* \neq \sum_{k\in U} \frac{|D_k|}{|D|} w_k^*,$$

which results in the sub-optimality of the aggregated global model in heterogeneous settings.

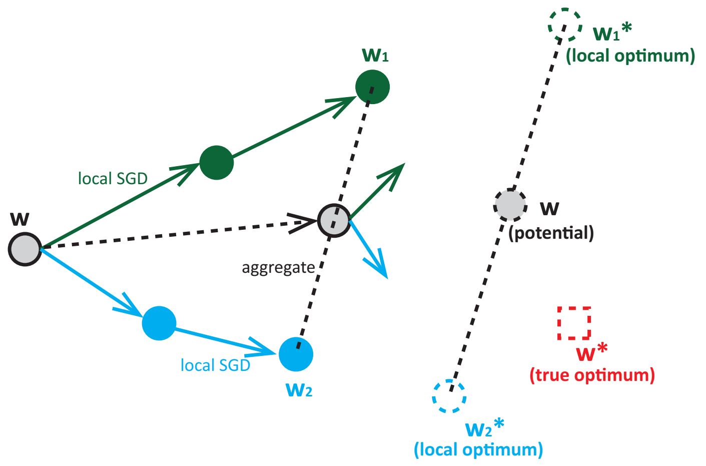


Figure 2.18: Illustrating the drift of local optima from the true global optimum in FL.

To address the aforementioned issue, the local and/or global objectives need to be ‘corrected’. A common technique is to introduce a proximal term [205–207] to the local objectives [71, 88, 151, 174]. The idea was initially adopted in the approaches to optimising centralised SGD such as SAGA [199] , which again can be viewed as an effective way to reduce gradient variance [208]. In the case of FL, the global model is suited for being the reference point for the local model in the parameter space, such that a proximal term can be added to the original local objective, i.e.,

$$F'_k(w_k; w) = F_k(w_k) + \underbrace{\frac{\mu}{2}\|w_k - w\|^2}_{\text{proximal term}}, \tag{2.49}$$

where w is the global model, w_k is the local model and μ is a tunable parameter. The key benefit of introducing the proximal term to local update (for example, as in FedProx [88]) is a soft restriction that pulls local models closer to the global model to mitigate heterogeneity. Besides, some approaches such as FedDyn [174] and FOLB [151] make further modifications to the local objectives to achieve faster convergence whilst some reconstruct the entire optimisation problem [113, 145, 171, 209] or redefine the relation between local update and global update [51].

Adaptive steps

The standard implementation of FL adopts the synchronous design because it provides better convergence property than asynchronous approaches in terms of the logical steps of updates needed [43]. But the practical issue of device heterogeneity stands out in realistic systems and makes FL less efficient [137]. To this end, efforts have been made in the design of adaptive configurations for local updates.

Flexible participation schemes were proposed to accelerate each round of local training by allowing (the selected) clients to submit incomplete updates [210] or opt out of a round [127] without hindering the global aggregation. The study by Wang et al. [100] takes into account situations where clients may take inconsistent number of steps and the authors further proposed to mitigate the problem by using normalised gradients and effective steps.

In most cases, it is the server that is responsible for setting up the training parameters (or termed hyper-parameters as they are used by the training algorithms). Realistic systems (e.g., IoT, MEC and crowdsensing) with limited resource budgets are usually sensitive to the trade-off between model convergence and resource usage. Therefore, in such circumstances it is imperative to consider the heterogeneity of devices in performance including and not limited to their data size, computational power and network throughput. Studies have shown that convergence improvement can be made through intelligent scheduling and configuring schemes such as adaptive aggregation frequency control [55] and adaptive (local) batch sizes and learning rates [89, 211].

Asynchrony

In the context of FL, asynchronous mechanisms are tempting due to the potential efficiency boost just like they are for traditional distributed learning paradigms. The main reason why asynchrony is usually thought to be advantageous over synchrony boils down to the 'straggler effect' which means that the overall efficiency of the learning process is basically decided by the slowest members among the participating clients [169]. Therefore, it is natural for the researchers to seek asynchronous alternatives [71, 72, 212, 213] that can release the server from awaiting stragglers.

A representative implementation of asynchronous FL is the FedAsync [71] algorithm which inherits the instant global update mechanism from AsyncSGD approaches [52, 61, 67]. FedAsync features incremental update with staleness control at the server-side and proximity operator SGD update at the client side. The concept of rounds (or epochs as termed in the paper) is kept for evaluating staleness but a separate thread is used for client selection periodically. The work by Kall and Trabelsi [115] adopts the FedAsync algorithm for training sensitive code token scanners collaboratively in an FL setting. They introduced extra personalisation mechanism to the local models, namely model interpolation and data interpolation, so as to refine local models' performance on local data.

Even though convergence guarantee is provided by Xie et al. [71], asynchronous approaches are empirically expected to take more update steps than synchronous FL to achieve the global model's convergence [43, 55]. Asynchrony may lead to less stable convergence over non-IID data because of the increased divergence of local models [101] and consequently brings about more communication costs. Besides, commonly-used privacy protection techniques such as DP [154, 214] are incompatible with asynchronous model or gradient uploads [169].

Client selection

As discussed in the research motivation (Section 1.4), client selection is a stage of the standard FL process and can be key to the optimisation of FL's efficiency. The random selection method [12, 47, 55] implicitly assigns equal importance to every client, which could be problematic from two different perspectives. First, the heterogeneity of devices in performance directly causes potentially huge difference in local training speed and consequently brings about the 'straggler effect'. The slow devices always have a fixed probability to be selected with the aforementioned random selection policy and cause a 'slow round'. Picking (potential) stragglers into a round of training directly hurts the FL's efficiency. Second, from the perspective of data heterogeneity, random selection is susceptible to the 'model poisoning' attacks by malicious clients or clients with low-quality data (e.g., noisy and irrelevant samples) [215]. Also, despite the quality of device-held data, the statistical heterogeneity of data can lead to large divergence between some local models and the others, which degrades the efficacy of model aggregation.

Straggler mitigation is an important topic for realistic environments. A straightforward solution is to preemptively exclude the potential stragglers in the selection stage [139]. Chai et al. [169] provide a different view of the problem by grouping clients into several 'tiers' according to their performance level. They proposed a tier-based solution (TiFL) that performs tier-wise client selection and incorporates a dynamic mechanism for updating the weights of the tiers based on accuracy evaluation.

Taming the statistical heterogeneity of data has attracted much attention from researchers in the field of FL. A variety of selection-based approaches are adopted such as weighted random selection based on data sizes [88], error/loss-oriented selection policies [159, 162], and Reinforcement Learning (RL) based selection policies [142, 216]. Since each client is bound to a specific local dataset, selecting the 'right' clients can facilitate the convergence of the global model. This can be achieved by involving an extra set of clients to correct the local updates before aggregation [151], or dynamically adjusting the participation probability of clients based on the norm of their updates [122].

FL has a lot of potential as a means of model training in modern crowdsourcing and crowdsensing systems [217, 218]. In such scenarios, an incentive mechanism is much needed [78, 79]. Besides, from the standpoint of the task publishers (who publish the machine learning task over the network and recruit participants), it is of great importance to dynamically assess the reputation and actual contribution of the participating devices in the system [179, 219].

Representation exchange

The magic of deep learning is in a sense attributed to the power of representation learning. It is the ability of the networks to learn multi-level latent representations that enables deep models to work well on data with complexity [220].

Extensive exploration has been taken from the perspective of how to learn more useful representations. On this point, most of the efforts were made by the researchers with interests in maximising the performance of a single model. However, with the surging need for distributed machine learning, the investigation of representations as an exchangeable information is still lacking. Note that the representations here refer to the hidden activations of any (multi-layer) ANNs rather than the specific distributed (word) representations [221–223] (or termed *embeddings*) for NLP tasks.

The representations (of data) produced by the deep models can be used to prevent the local model divergence in FL. Based on this idea, Li et al. [105] proposed (MOON), a Model-Contrastive Federated Learning framework that rectifies the directions of local training by constraining the dissimilarity between the representations from local models. This is realised by introducing the representation similarity between three types of models (current local model, previous local model and the global model) into the local loss function. Representations are also useful for building personalised local models, which will be discussed in the next category — *Specialised local models.*

Although representation exchange is widely adopted in Federated Transfer Learning

(FTL), it is barely explored in the context of horizontal FL. As a contribution of this thesis, Chapter 5 investigates the value of representations and the utilisation thereof for efficient horizontal FL.

Specialised local models

Specialised local models are a result of running FL for maximising the performance of local models on individual devices instead of building a generic global model (which could be a by-product though). They are defined as opposed to the general-purpose global model and developed for situations where the users want to have unique and 'strong' local models customised for their own data.

The motivation of specialisation is well supported by the heterogeneity of local data [101, 115, 224]. Model personalisation can be very necessary when local performance is a paramount demand. Tan et al. [225] reviewed the existing approaches to personalised FL (PFL) and categorise them into data-based methods and model-based methods. The majority of personalised FL approaches are model-based methods where the local objectives and update rules are tailored [128, 168, 175, 226], the models are split to keep local knowledge [104, 129] or the global model is mixed with local models [108]. Clustering is a promising solution to data heterogeneity [227]. By dynamically grouping clients into clusters, e.g., based on their model parameters [125, 143, 170] or their affinity to the cluster models [170], each client is assigned a model that is still shared (among a group of similar clients) but more fitted to its own data. In addition, researchers have found the intrinsic association between the personalisation of FL and emerging ML paradigms such as Knowledge Distillation [106, 228, 229], Multi-Task Learning (MTL) [76, 176, 228] as well as Model-Agnostic Meta Learning (MAML) [171].

Extreme conditions

Federated learning is meant for large-scale distributed machine learning, so extreme conditions should be expected. These conditions include but are not limited to strongly biased data distribution [109], pervasive data of low quality [3, 230], the existence of unreliable devices and adversaries (also termed Byzantine machines in the literature) [116, 119] and extremely limited resources available [229].

Among these performance-affecting conditions, the scarcity of communication-related resources can be mitigated through various techniques that will be discussed later in the Section 2.3.4 *Communication in FL systems.* Methodologies for addressing potential adversarial participants will be reviewed in the Section 2.3.5 *Privacy and Security in FL.*

2.3.3 Other FL paradigms

The paradigm of horizontal FL covers the majority of use cases from cross-device learning systems to cross-silo collaborative scenarios [49]. Yet some situations are out of the scope and require specifically designed learning paradigms.

Vertical Federated Learning (VFL) and Federated Transfer Learning (FTL) are popular variants that follow the primary principles of FL, namely no raw data exchange, and have the potential to make great impacts in many industrial applications. Both VFL and FTL are formally defined and included in the IEEE Standard 3652.1-2020. Fig. 1.6 in the previous chapter illustrates their difference from the perspective of data domains.

Vertical FL was developed for training across different feature spaces and has much in common with some branches of existing ML paradigms including Split Learning (SL) [104, 141], Multi-View Learning (MVL) [231] and ensemble learning [173, 229]. In the IEEE standard, the paradigm is described in the following extract:

> *Vertical FML*[16] *refers to building a model in the scenario where data sets have significant overlaps on the sample space D, but not on the feature spaces $(X_2, X_2, \ldots)$. In this case, an FML model can be built as if the data is split and joined vertically. Vertical FML may apply to scenarios where there are insufficient features or labels to build a high-quality model.* (p.17, IEEE Standard 3652.1-2020)

Comparing to VFL, federated transfer learning targets at more special scenarios where the goal is to help *one* of the multiple (usually two) parties establish a model with the knowledge from the other party/other parties that have a fairly different set of samples and features with labels. The IEEE standard describes FTL in the following way:

> *Federated Transfer Learning (FTL) refers to the federated machine learning technique designed for application scenarios where data sets have no significant overlap on neither the sample space nor the feature space. FTL takes*

[16]Synonymous for Vertical FL

advantage of transfer learning techniques to exploit reusable knowledge across different feature domains, and consequently, results in high-quality FTL models despite small data and weak supervision difficulties. (p.19, IEEE Standard 3652.1-2020)

Both VFL and FTL have drawn attention from researchers in recent years. Yang et al. [48] presents a survey on the definitions, workflows and applications of these FL paradigms. Due to the constraints on loss and gradient encryption, most of the researches on Vertical FL are mainly confined to relatively simple models (such as logistic regression) that take the form of an additive combination of the sub-models' outputs (i.e., intermediate results produced by the models on different devices) [48, 232–235]. Only a few stand out and seek to generalise VFL to general-purpose ML models and loss functions [212, 236], multi-party scenarios [236, 237] or coordinator-free architectures [233, 234]. Since a common intersection of data entities is usually needed for both VFL and FTL, entity alignment (also termed Entity Resolution in the literature) is a vital pre-processing work [233, 238]. Errors (especially cross-class mismatching) in entity alignment can have a big impact on the models to be learnt [239]. Compared to VFL, the investigation of FTL is relatively lacking. This is in some ways due to the inherent similarity between the two paradigms and in most cases, cross-feature space learning algorithms for VFL can be easily adapted to FTL scenarios [240, 241].

The pervasive nature of FL also facilitates researches on adapting emerging machine learning paradigms to the framework. Wu et al. [242] explore the possibilities of realising personalised FL at the network edge via Federated Transfer Learning, Federated Meta Learning, Federated Multi-task Learning, Federated Distillation and data augmentation. Moreover, fully decentralised FL in Peer-to-Peer (P2P) networks has also become a topic of interest [58–60, 118, 243].

2.3.4 Communication in FL systems

Communication is always reckoned as the bottleneck of networked systems and should be a primary concern in the design of distributed learning algorithms [244]. FL is of no exception and the time cost of communication usually plays the main factor in terms of FL's efficiency [245, 246].

The most straightforward method for reducing communication cost is payload compression [247], which can be applied to the transmission of models and model updates (e.g., accumulated gradients). In the context of FL over a wireless network, the two most popular compression methods are sparsification and quantisation. Sparsification, or termed top-k sparsification, refers to a class of compression methods that sparsify the vector (e.g., a flatten model) by only keeping the k largest entries with all the rest omitted [86, 148, 248, 249]. Theoretical analysis provided in Stich et al.'s work [86] proves that sparsification does not degrade the convergence rate of distributed SGD. Liu et al. [250] further improved the efficacy of sparsification by virtue of a hierarchical structure.

In contrast to sparsification, quantisation does not change the length of the vector but instead does a $\phi : \mathbb{R} \to \mathbb{F}_q$ mapping for every component in the vector. Here $\mathbb{F}_q$ is a finite field with only q elements. Since FL typically works at scale, the error caused by model or update quantisation is usually controllable whilst the reduction of traffic is substantial [80, 110, 135]. The compression rate can be further improved by combining the two methods together [148]. As an alternative option, the number of parameters to transfer can be reduced by directly splitting the models and having each client only exchange a part of them with the server [141, 251].

The complexity of optimising the communication in FL boils down to the trade-off between faster convergence and lower communication costs. When it comes to the network edge, resource allocation becomes critical [178]. Based on this fact, some studies seek to find joint solutions to the problems of client selection/scheduling and network resource allocation [120, 135, 156, 252]. Partial participation is very necessary in resource-limited edge environments, and it has been proven, both theoretically and empirically, that precluding some 'less useful' updates can save traffic without affecting convergence [120, 123].

Federated learning is naturally suited for collaborative machine learning over mobile devices through wireless connections [245, 246, 253]. The advances in wireless communication technology have facilitated the development of FL in mobile edge networks [94, 117, 149]. Nonetheless, there is still much to explore for the application of FL in the 5G era and beyond [252, 254].

2.3.5 Privacy and Security for FL

Privacy and security are two prominent issues in FL systems that have raised increasing concerns [255]. Conceptually the two terminologies have overlaps with each other, but in the context of FL privacy and security are often studied from different perspectives [256]. In general, privacy issues often refer to the disclosure of user-relevant information including models, gradients and most importantly, raw data. On the other hand, security problems in an FL system are usually caused by malicious parties who intentionally launch attacks against the system to backdoor the global model, slow down the convergence or achieve other adversarial purposes. In some cases, security breach can also lead to compromised privacy [124].

The primary concern of privacy is whether the raw data held by participating users can be reconstructed. A number of studies have raised the alarm by showing that gradients can be used to recover the raw input of even deep models in both CV and NLP tasks [257–259]. On the other hand, model poisoning attacks are regarded as an outstanding security risk in the collaborative training process of FL [155, 215]. Fortunately, a variety of solutions have been developed and the majority of them root from cryptography [103, 119], masking schemes [124, 154] and Secure Multi-party Computation (SMC) theories [91].

Among the popular approaches, Homomorphic Encryption (HE) provides the theoretically most secure solution. However, the ultimate generation of HE, namely Fully Homomorphic Encryption (FHE) [53], is not very possible to be realised realistically. In the context of FL, additive HE schemes are sufficient for the operation of model/update aggregation [103, 119]. It is also noteworthy that HE is usually indispensable for VFL and FTL [48].

Secure multi-party computation is a widely adopted framework when it comes to collaborative computing and machine learning. The primary principle for SMC is only allowing each party to know the system output and its own input. This mechanism proves very helpful in a lot of situations where some parties in the system can be potentially curious about other parties' 'secrets'. Some methods have been developed and applied to FL based on Secret Sharing theories [154, 260]. Another promising roadmap to privacy-preserving FL is Differential Privacy (DP), which is a class of methods that use artificial noises to 'perturb' models/updates locally and probabilistically recover the sum at the server side [103, 107]. DP is computationally lightweight and can be robust to inference-based attacks and failures [91, 121, 126, 132]. Apart from passive defence, some researches resort to proactive detection and preclusion of malicious or unreliable contributors in the system [138, 261].

In order to achieve a desired level of privacy and/or security, multiple techniques can combined in practice [103, 121, 126]. But it is always necessary to account for the costs and potential impacts (e.g., performance loss) [262].

2.4 Datasets for FL Research

Any design of algorithms and protocols for FL needs to be evaluated under FL settings. The selection of datasets is pivotal in such benchmarking process. A straightforward option is to manually partition centralised data into decentralised shards. For example, popular image classification datasets such as MNIST [263], CIFAR-10/100 [264] and ImageNet [28] can be used for evaluation after splitting them by some rules. The partitioning rules determine the data distribution across the clients. For instance, non-IID settings can be created by forcing class imbalance for local data [47]. An example of manually partitioned CIFAR datasets based on a Dirichlet distribution (with the concentration parameter α set to an all-one vector) can be found in a GitHub repository created by the author at this link: `https://github.com/wingter562/LEAF_prepartitioned/tree/main/CIFAR_Dirichlet`.

A more natural way is to experiment on FL–style datasets (FL datasets for short). By the time of writing, only a few FL datasets are publicly available, as summarised in Table 2.2. The 'Domains' column gives the number of unique data sources that can be, for example, writers (for FEMNIST) or twitter users (for Sentiment140). Note that the last column of Table 2.2 shows the parties who released the *partitioned* datasets rather than the original creators of the source data. The source data contributors can be found in the references given.

Table 2.2: Public (pre-partitioned) datasets for FL research. Meta-data are retrieved from the repositories or papers referenced where the source of the data can be found.

Dataset name	**Task**	**Size**	**Domains**	**Released by**
Street-5/20	Object detection	956	5/20	FedAI[265]
Synthetic(α, β)	classification	8977	30	Li et al. [88]
FEMNIST	Image classification	805,263	3,550	LEAF[266]
Shakespeare	Next character pred.	422,615	1,129	LEAF
Sentiment140	Sentiment analysis	1,600,498	660,120	Kaggle, LEAF
Celeba	Image classification	200,288	9,343	LEAF
Reddit	Language modelling	56,587,343	1,660,820	LEAF
BigQuery	Multi-purpose pred.	146,404,765	342,477	StackOverflow
AGNews	Text classification	127,600	1,000	FedNLP[180]
20News	Text classification	18,842	100	FedNLP
SST2	Text classification	8,742	30	FedNLP
PLONER	Sequence Tagging	17,501	50	FedNLP
W-NUT	Sequence Tagging	4,681	30	FedNLP
WikiNER	Sequence Tagging	286,495	1,000	FedNLP
SQuAD	Question Answering	122,325	300	FedNLP
Movie-Dialogs	Text generation	221,634	617	FedNLP
CNN/DM	Text generation	312,085	100	FedNLP
hERG	Regression (Biophy.)	10,572	4	FedGraphNN[13]
QM9	Reg. (Quan. Mech.)	133,885	8	FedGraphNN
ESOL	Reg. (Phy. Chem.)	1,128	4	FedGraphNN
FreeSolv	Reg. (Phy. Chem.)	642	4	FedGraphNN
Lipophilicity	Reg. (Phy. Chem.)	4,200	8	FedGraphNN
BACE	Classification (Biophy.)	1,513	4	FedGraphNN
BBBP	Classification (Biophy.)	2,039	4	FedGraphNN
SIDER	Classification (Physio.)	1,427	4	FedGraphNN
ClinTox	Classification (Physio.)	1,478	4	FedGraphNN
Tox21	Classification (Physio.)	7,831	8	FedGraphNN
CIFAR-100	Image classification	60,000	100	FedCV[14]
GLD-23K	Image classification	23,080	233	FedCV
PASCAL VOC	Image classification	11,355	8/4	FedCV
COCO	Image classification	328,000	10	FedCV

Chapter 3

Driving Horizontal FL with Flexible Synchronisation

The rapid advances in machine learning techniques are catalysing a broad range of applications which more or less integrate AI components into user devices to empower their underlying business logic. As predicted by Gartner, more than 80% of enterprise IoT projects are expected to have built-in AI components by 2022 [267]. Also, it has been an emerging trend that users are becoming more sensitive to the data privacy protection mechanism of AI applications, while their performance, in many cases, is still expected to be guaranteed in the first place.

As it has been analysed in the previous chapters, decentralised data requires decentralised machine learning (with privacy protection) whilst the heterogeneity of data and devices poses great challenges both theoretically and practically. The training of deep models in such environments has to deal with i) *imbalanced and biased data distribution*, where devices may have various volumes data and the distribution of data may differ from device to device, ii) *a massive scale* of disparate devices at the network edge as participants, and iii) *heterogeneity and unreliability* exhibited by the participating clients and the communication channels.

3.1 Motivation

Generally, the heterogeneity of FL clients refers to their discrepancy in processing speed (i.e., how fast a model can be trained) and network throughput (i.e., how long it takes to upload/download models). In this work, an extra factor of device heterogeneity is taken into account, namely, the reliability of devices. It is of great necessity to be aware that end devices in FL are usually not obligated to participate. Devices such as mobile phones can be FL clients while running many other applications and they occasionally opt out or drop out for reasons like low battery or poor network condition. Most of the recent studies investigate FL over a cluster of heterogeneous but reliable devices [88, 139, 145, 159]. However, devices such as mobile phones intermittently dropping out halfway in local training can have a strong impact on the overall efficiency of an FL system [49]. Unreliable clients can cause:

- **'Missing' participants**: in each round, the server selects a fraction of clients randomly to perform local training and expects them to commit their local training results. However, the number of clients which manage to commit their results are very uncertain given the unreliable nature of end devices;
- **Long wait**: To aggregate the local results at the end of each round, FedAvg has to wait for all selected clients to finish, among which there may be stragglers while the crashed ones may never respond. Consequently, the global learning progress is suspended until a timeout threshold is reached ;
- **Under-utilisation of clients**: With random selection, many capable clients remain idle even if they are willing to participate and ready to do more training;
- **Progress in vain**: When not being able to finish local training in time, the progress made is wasted as the client will be forced to synchronise by the server.

3.2 Contributions

This chapter presents a Semi-Asynchronous Federated Averaging (SAFA) algorithm enabling fast, lag-tolerant federated optimisation. The algorithm is *semi-asynchronous* because a special mechanism is adopted to allow specific clients to work asynchronously with the others. SAFA takes advantage of several efficiency-boosting features from asynchronous machine learning approaches (e.g., [268][71]) while making use of a refined pace control mechanism to mitigate the impact of straggling clients and stale models (i.e., staleness [71]) on the global learning progress. Moreover, a novel aggregation algorithm is adopted using a cache structure (in the cloud) to bypass a fraction of client updates so as to improve convergence rate at a low cost of communication. The main contributions of this work are outlined as follows:

- By taking into account the unreliability and heterogeneity of end devices, this work proposes a Semi-Asynchronous Federated Averaging (SAFA) algorithm to alleviate the staleness, boost efficiency and better utilise the progress made by stragglers.
- A simple parameter, *lag tolerance*, is introduced to flexibly control the behaviour of SAFA algorithm. The impact of lag tolerance on SAFA is empirically analysed by observing how it affects the critical metrics such as synchronisation ratio and version variance.
- Extensive experiments were conducted with several typical machine learning tasks in multiple FL settings varying from tiny to relatively large-scale environments. SAFA is evaluated in terms of several important metrics such as model accuracy, round efficiency and communication cost.

3.3 System Model

In the context of FL, the aim of this work is to solve the global optimisation (i.e., empirical risk minimisation) problem as below:

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(w; X_i, Y_i), \tag{3.1}$$

where w denotes the parameters of the global model (the number of parameters $= d$), $\ell(w; X_i, Y_i)$ represents the loss of the inference on sample (X_i, Y_i) made by the model with w as its parameters. Note that data samples are distributed among disparate end devices and always remain local. Let U denote a set of N clients, and D_j the partition of data residing on client j, then the target function can be rewritten as:

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{j=1}^{N}\sum_{i\in D_j} \ell(w; X_i, Y_i). \tag{3.2}$$

Note that the problem definition here is in accordance with [47], but is different from [71]. Xie et al. [71] define their target function as the average of the average loss (on local partitions), which is fair at the local partition level but not the case at the sample level, because data samples in small local partitions take larger weights in their target function. For clarity, Table 3.1 lists the symbols frequently used in this chapter.

Table 3.1: List of symbols

Symbol	Description
D	the complete dataset
n	the size of D (i.e., $n = \|D\|$)
D_i	the data partition on client i
n_i	the size of client i's local partition
U	the set of clients (i.e., end devices)
N	total number of clients (i.e., $N = \|U\|$)
v_i	the version of client i's local model
U_v	the set of clients whose model version is v
S	the set of selected clients
S_v	the set of selected clients of version v
Z	the set of crashed clients
Z_v	the set of crashed clients of version v
X	the set of clients that complete local training
Q	the set of undrafted clients
Q_v	the set of undrafted clients of version v
$w^{(t)}$	the global model at round t
$w_k^{(t)}$	the local model on client k at round t

An important property of end devices is unreliability, which means that they occasionally drop offline for some reasons such as power outage (or low battery level), inaccessible network or manual shutdown/opt-out of training. In this work, these temporarily unavailable states are referred to as *crashed*. Every client has a certain probability to crash in each round of training. For clients that stay active and connected to the central server (throughout a round of training), it is assumed they are always able to finish the task assigned within a certain period of time (otherwise they are also reckoned crashed). Without loss of generality, in this work clients are assumed independent of each other regarding their behaviours.

The population of committed updates should be carefully limited considering a huge fleet of end devices [49]. McMahan et al. [47] use a parameter C to control the maximum fraction of clients allowed to participate in one round of training. Moreover, C serves as the criterion in the FedAvg [47] algorithm by which the server keeps waiting for selected clients to end a global round. In the proposed approach, this hyper-parameter is retained but no longer applied as a hard constraint. Instead, the restriction is relaxed to allow all clients to participate if they are willing to, and enable the central server to end a round once a C-fraction of updates have been received.

It is notable that the fraction of selected clients (called selection fraction) is not equivalent to the actual fraction of clients that finish local training and commit their models in time. In an unreliable environment, picked clients can crash halfway in their training progress or fail to upload their trained models. This work defines a metric termed Effective Update Ratio (EUR) to measure the fraction of effective updates from the local (i.e., all clients) to the cloud (i.e., central server(s)).

$$EUR = \frac{|S - S \cap Z|}{|U|}, \tag{3.3}$$

where S and Z are the sets of picked and crashed clients, respectively. Obviously EUR is positively correlated with the size of S and negatively correlated with that of Z.

As discussed in Section 1.5, average round length (i.e., the time needed to complete a round of FL) is one of the key metrics that reflect the efficiency of the FL process. The length of a federated round can be modelled by considering both local training time and communication overheads, which is captured by Eq. (3.4).

$$T = \min\left\{T_{lim}, T_{dist} + \max_k\{T_k^{down} + T_k^{up} + T_k^{train}\}\right\} \tag{3.4}$$

where T_{lim} is the preset upper limit of round length. T_k^{train}, T_k^{down} and T_k^{up} denote local training time, model download and upload time for client k, respectively. T_k^{down} and T_k^{up} depend on model size and device bandwidth. For client k, its local training time (i.e., T_k^{train}) is determined using Eq. (3.5):

$$T_k^{train} = \frac{|B_k| \cdot E}{s_k} \tag{3.5}$$

where E is the number of local epochs and $|B_k|$ is the number of batches for device k. A client's performance s_k is modelled as the number of batches the client is able to process per second.

For a federated round, T_{dist} denotes the server-side overhead for distributing the global model to the end devices. In this work it is assumed that the server can fully utilise its bandwidth to send models in parallel via intermediate network elements [55] to the clients. Thus T_{dist} depends on the number of model copies to distribute (denoted by N_{sync}) and the communication bandwidth of the server (denoted by bw). T_{dist} is formulated in Eq. (3.6).

$$T_{dist} = \frac{N_{sync} \cdot model\_size}{bw} \tag{3.6}$$

where the server bandwidth *bw* is a constant considering that the server and base stations linked with wired connection. The size of a model is also a constant as FL does not change the models' shape.

3.4 SAFA: A Semi-Asynchronous Algorithm for fast FL

This section details the key designs in SAFA including lag-tolerant model distribution (Section 3.4.1), post-training client selection (Section 3.4.2) and discriminative aggregation (Section 3.4.3). These designs enable a semi-asynchronous control over the clients by allowing specific clients to work asynchronously with the others.

3.4.1 Lag-tolerant Model Distribution

In the federated setting, an important guarantee for the convergence of the global model is the quality of local models. The quality of a local model depends on whether local training is sufficient (i.e., training sufficiency) and on the starting model on which local training is based. Training sufficiency can be achieved by allowing adequate local iterations (i.e., epochs), while the version control is a non-trivial task. The original FL algorithm [47] prevents outdated clients with stale models from committing, which simplifies FL process but also throttles the potential of accelerating convergence.

Motivated by the problem, a lag-tolerant model distribution algorithm is presented which does not always enforce synchronisation (i.e., allows some clients to stay asynchronous with the others and unsynchronized with the cloud) and is tolerant to outdated local models (i.e., staleness). The key idea is to develop a better way to get the stragglers (i.e., clients with stale models) involved in the model aggregation and leverage their progress for faster federated learning. This work refers *stragglers* to the clients who are slow and still conducting local training based on an outdated model. Normally, the clients are supposed to start epochs of training based on the latest global model received from the server. However, device crashes or network problems generate the stragglers inevitably.

With a version-based criterion, SAFA only requires specific clients to retrieve the latest global model from the server. Before a round of local training starts, the server classifies all clients into three states (or categories) based on their current versions:

Up-to-date, *deprecated* and *tolerable*, which are defined as follows.

Up-to-date clients: the clients that have completed the previous round of local training (and submitted models successfully) are reckoned up-to-date at the start of this round.

Deprecated clients: the clients that still base local training on the models that are too stale (given a threshold) compared to the version of the global model.

Tolerable clients: the clients that do not base local training on the latest global model, but the model version they are based on is not too old either. This is a state that stands between *Up-to-date* and *Deprecated*.

SAFA only requires the up-to-date and deprecated clients to synchronise with the server, while the tolerable clients stay asynchronous with the server. This is why SAFA is called a semi-asynchronous distributed training scheme. SAFA let up-to-date clients synchronise with the server in order to prevent model divergence [47]. Deprecated clients are forced to synchronise so that the global model will not be poisoned by the seriously outdated local models.

After a round of local training is completed on device, the clients will then be labeled *picked*, *undrafted* or *crashed* based on the result of client selection in SAFA, which is a post-training process. The server tags clients with these labels after the selection quota is met or the round time limit is reached. The picked clients are those whose local training results in this round are selected to be used in the following aggregation step. The undrafted clients are those whose local training results are not selected but still get cached by the server for future use. Crashed clients are those who fail to finish a round of local training — clients can either opt out or drop offline intermittently (i.e., any time during training) with a certain probability (which is referred to as crash probability).

A typical FL process driven by SAFA is shown in Fig. 3.1. This diagram will be used as an example throughout this chapter to illustrate the FL training process.

In Fig. 3.1 the workflow of SAFA is illustrated with four end devices: clients A to D, with which the system is to perform several federated rounds of training. Clients start local training from their local model versioned A0 to D0 (the initial model version for each round of local training is depicted in the figure by a single circle with the model version in the middle). After a client completes its local training, local parameters are updated (depicted by double circles with the model version in it) and are uploaded to the server (depicted by upwards arrows). The server selects

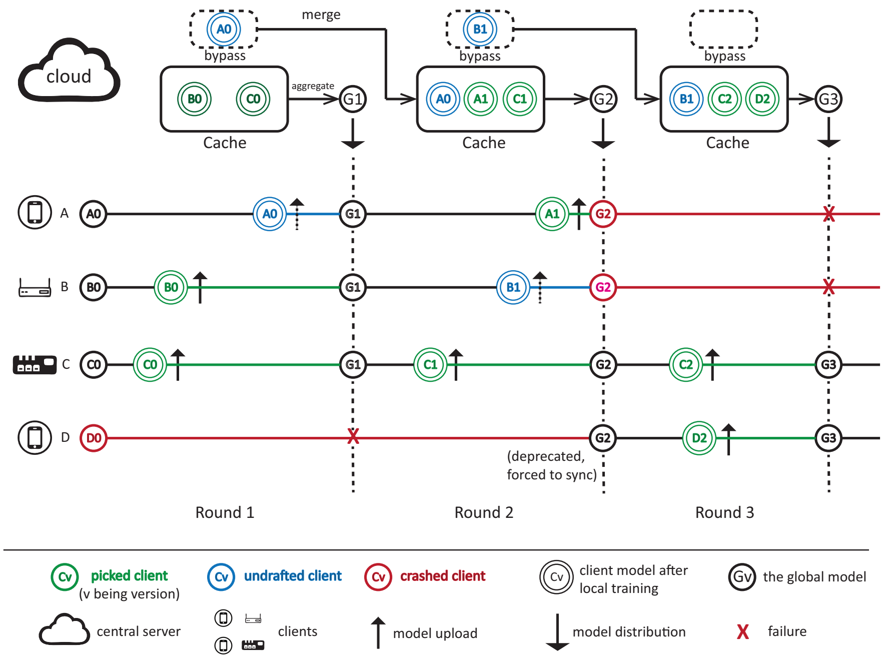


Figure 3.1: The diagram of SAFA algorithm showing the interaction between the cloud and end clients in different states

submitted results only from a portion of clients (50% in this example) to update the global model. The clients whose updates are selected are tagged as picked clients (colored green), for example, clients B and C in the 1st round. The selected updates are placed in a cache structure by the server. The cache maintains the entries of the latest local models uploaded from the picked clients and will be used for aggregation. The clients whose results are not selected are undrafted clients (colored blue), e.g., client A in round 1 and client B in round 2. Updates from these clients are stored in the bypass structure to avoid futile work locally. The clients who cannot complete their local training due to any reason (such as opt-out or network failure) are crashed clients (highlighted red), such as client D in the 1st round.

Each round ends with a new version of global model (i.e., G1 to G3 in this diagram), which, at the start of the next round, will be distributed to (i.e., synchronised with) the up-to-date and deprecated clients. In the first round, for example, A, B and C successfully complete local training and upload their updates (in spite of A being undrafted), thus they become up-to-date clients (i.e., tagged *up-to-date* by the server). The results of undrafted clients will not be merged into the global model in the upcoming aggregation step, but may take effect in future rounds via a bypass structure (squares with dashed lines) that saves these updates temporarily. The bypass will merge with the cache right after the current aggregation step before the next round starts.

In this example, it is assumed the maximum tolerable version lag is 2. In Fig. 3.1, client D does not manage to finish local training in two rounds. Therefore, it is tagged *deprecated* and forced to synchronise with the server, which means client D needs to replace its stale local model with the latest global model. To decide whether a client's model is *deprecated*, here a simple criterion is adopted based on the difference between the versions of the global model and the local model, which is called *lag tolerance*. Therefore, the deprecated clients are those whose local version lags behind the version of the global model by more than the specified *lag tolerance*. Specifically, the proposed lag-tolerant distribution principle can be formulated as follows:

$$w_k^{(t)} = \begin{cases} w^{(t-1)} & \text{if } k \in \bigcup_{v=t-1} U_v, \text{or } k \in \bigcup_{v<t-\tau} U_v, \\ & \text{// up-to-date or deprecated clients} \\ w_k^{(t-1)} & \text{if } k \in \bigcup_{t-\tau \le v < t-1} U_v \\ & \text{// tolerable clients} \end{cases} \tag{3.7}$$

where $w^{(t-1)}$ denotes the latest global model parameters (i.e., the aggregation result from last round) upon the start of round t, and w_k denotes the parameters of client k's local model; τ stands for *lag tolerance*, which is the only parameter for SAFA. The lag-tolerant model distribution forces the up-to-date and deprecated clients to adopt the latest global model as the base model for the next round of training, while the tolerable clients can continue to work on their previous local results. The algorithm parameter *lag tolerance* in some ways controls the tradeoff between communication overhead and the convergence rate of federated optimisation. If it is set too small, the server may suffer heavy downlink transmission as the portion of deprecated clients increases. If it is set too large, the convergence of the global model could be unsteady. The impact of *Lag tolerance* will be analysed later with empirical studies.

3.4.2 Client Selection

Apparently, the efficiency of federated optimisation is closely associated to the fraction of picked clients. One may think that we can set C to a large value (e.g., close to 1.0) and pick as many clients into each round as possible. However, it is neither realistic nor beneficial to do so. On the one hand, allowing more clients to participate increases the potential risk of uplink congestion and the communication cost as well. In each round, the server may have to wait for more clients among which some may never respond (because picked clients could crash midway). On the other hand, involving a large number of updates leads to limited benefit to the global model especially in the last few rounds before convergence [47].

As mentioned, simply increasing the pick fraction can bring about problems in the FL context, while the crash of clients is not predictable or controllable (improving client stability is beyond the scope of this work). As a solution, one can let the central server collect local updates after local training instead of randomly selecting clients at the very beginning of a global round. This means the server does not need to wait for those designated clients for aggregation but is able to execute the aggregation step once it has received a C-fraction of update. The post-training selection effectively decouples the server with the selected clients and consequently improves EUR, which facilitates faster convergence of the federated optimisation. Another advantage of doing so is a significant boost of round efficiency in the case where clients drop out with fairly high probability. Based on the outcome of selection (before the aggregation step is carried out), the server tags the clients with three different labels: *crashed*, *picked* and *undrafted*. Only picked clients are eligible to update its corresponding cache entry right before the aggregation conducted by the central server. Undrafted clients also commit their updates but their updates will bypass the following aggregation.

Considering the 'selection-ahead-of-training' scheme used in the synchronous FL (e.g., FedAvg [47]), its effective update ratio, according to Eq. (3.3), is $C(1 - \frac{|Z|}{|U|})$. By contrast, SAFA adopts a 'selection-after-training' scheme that theoretically yields the value of EUR as follows:

$$EUR = \begin{cases} 1 - R & \text{if } C \geq 1 - R, \\ C & \text{if } C < 1 - R \end{cases} \tag{3.8}$$

where C is the selection fraction and R denotes the crash ratio over all the clients (i.e., $R = \frac{|Z|}{|U|}$). Fig. 3.2 demonstrates how SAFA promotes the effective update ratio in FL.

From Fig. 3.2, one can see a clear improvement of EUR by SAFA, which minimises the negative impact of clients' failure. Nevertheless, extremely high crash ratio of clients will still cause a low value of EUR even with the proposed selection method. The phenomenon will be analysed later in the experiment section.

As figured out by Bonawitz et al. [12], bias is introduced if each device is equally

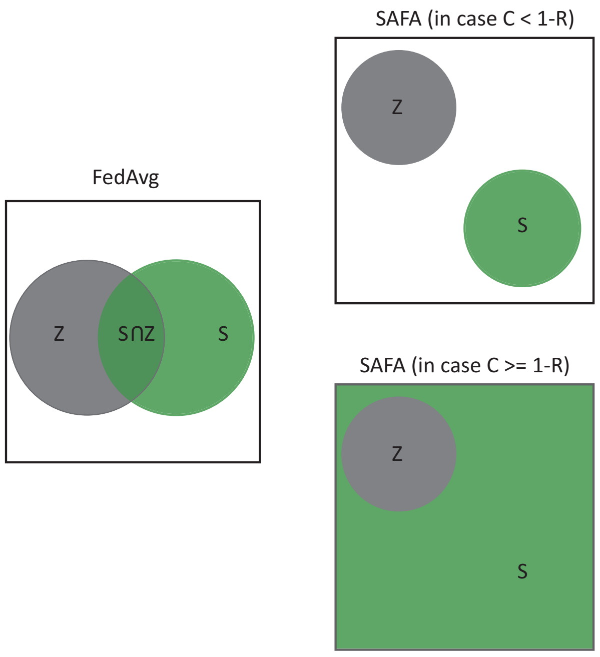


Figure 3.2: An illustration of the client selection policies of FedAvg and SAFA where the squares represent the full client set U. Note that an extra structure (the bypass buffer) will be introduced later to further improve the EUR.

likely to participate each round because they differ in performance and network access privilege. The problem remains despite the client selection method mentioned above. Therefore, a novel method is further proposed to alleviate the bias using a compensatory client selection algorithm. The principle is simple — higher priority is given to those clients that are less involved. In each round the server maintains a list of IDs of clients that missed the previous round of training, and their updates will be picked prior to others for the coming aggregation. The pseudo-code of the proposed selection policy is shown in Algorithm 1.

In the selection, the server stops involving more clients once the quota has been met, namely a C-fraction of clients have been selected from $S^{(t-1)} \cap X^{(t)}$. Otherwise the algorithm continues to wait and accept the updates (until a deadline is reached) from the rest of clients which, in practice, will arrive at the cloud successively.

3.4.3 Discriminative Aggregation

After a round of local training completes, the server has received a collection of updates from the end devices. Three steps are taken to aggregate local updates. The first step is the pre-aggregation cache update, which overwrites the corresponding entries (for storing model parameters) of the selected clients in the cache. In the second step, the

Algorithm 1: Compensatory First-Come-First-Merge (CFCFM) client selection

```
Input  : round number $t$, client set $U$, last-round picked clients $S^{(t-1)}$,
         selecting fraction $C$, round time limit $T_{lim}$
Output : clients to pick $S^{(t)}$
$S^{(t)}$ = models in the bypass buffer
$Q^{(t)} = \emptyset$
$quota = C \cdot |U|$
while $|S^{(t)}| < quota$ and $T_{round} < T_{lim}$ do
    Await new updates
    $w'_k \leftarrow$ update arrives from client $k$
    if $k$ not in $S^{(t-1)}$ then
        add $k$ to $S^{(t)}$
    else
        add $k$ to $Q^{(t)}$
    end
end
if $|S^{(t)}| < quota$ then
    Sort $Q^{(t)}$ by arrival time
    $q \leftarrow quota - |S^{(t)}|$
    $[q] \leftarrow$ first $q$ clients in $Q^{(t)}$
    $Q^{(t)} \leftarrow Q^{(t)} \setminus [q]$
    $S^{(t)} \leftarrow P^{(t)} \cup [q]$
end
return $S^{(t)}$
```

updates stored in the cache are aggregated. In the third step, the undrafted updates (in the bypass buffer) are moved to the cache, which can be used in the next round of global model aggregation. Since the picked and undrafted updates are treated in a different manner in the aggregation, it is called the three-step discriminative aggregation, which is formally formulated as follows:

(1) *Pre-aggregation Cache Update*:

$$c_k^{(t)} = \begin{cases} v_k^{(t)} & \text{if } k \in S^{(t)}, \\ w^{(t-1)} & \text{if } k \in \bigcup_{v<t-\tau} U_v^{(t)}, \\ c_k^{(t-1)} & \text{otherwise} \end{cases} \tag{3.9}$$

where $c_k^{(t)}$ denotes the k-th entry of the cache structure (see Fig. 3.1), and $v_k^{(t)}$ denotes the trained local model at round t. Entries of deprecated clients will be replaced with the global model $w^{(t-1)}$.

(2) *SAFA Aggregation*:

$$w^{(t)} = \sum_{k=1}^{N} \frac{n_k}{n} c_k^{(t)} \tag{3.10}$$

(3) *Post-aggregation Cache Update*:

$$c_k^{(t+1)} = \begin{cases} v_k^{(t)} & \text{if } k \in Q^{(t)}, \\ c_k^{(t)} & \text{otherwise} \end{cases} \tag{3.11}$$

where $S^{(t)}$, $Q^{(t)}$, and $Z^{(t)}$ denote the sets of picked, undrafted, and crashed clients, respectively in round t.

For SAFA, there are three cases of changes in the cache after a global around t. For picked clients, their updates will be kept in the cache after being merged into the global model. For undrafted clients, the updates will not take effect in this round but will be carried to the next round by the post-aggregation step. For the crashed clients, their entries stay unchanged only if they have not been deprecated. Otherwise these entries will be replaced by the global model (i.e., $w^{(t-1)}$ in Eq. 3.9) to avoid heavy staleness.

Algorithm 2: Semi-Asynchronous Federated Averaging (SAFA) algorithm

```
Input  : maximum number of rounds $r$, client set $U$, local mini-batch size $B$,
         number of local epochs $E$, learning rate $\eta$, lag tolerance $\tau$
Output : finalised global model
Server process:        // running on the central server
Initialises client connections
Initialises global model $w$ and the cache
for round $t = 1$ to $r$ do
    Distributes $w$ according to Eq. (3.7) given $\tau$
    for each client $k$ in $U$ in parallel do
        $w'_k$ = client_update($k, w$)
    end
    Collects and selects client updates using CFCFM
    Updates cache according to Eq. (3.9)
    Performs aggregation and get $w$ using Eq. (3.10)
    Updates cache according to Eq. (3.11)
end
return $w$
Client process:        // running on the client $k$
client_update($k, w_k$):
$B_k \leftarrow$ batches from $D_k$ of size $B$
for epoch $e = 1$ to $E$ do
    for batch $b$ in $B_k$ do
        $w_k = w_k - \eta \nabla f(w_k; b)$
    end
end
$w'_k = w_k$
return $w'_k$ to the server
```

The complete workflow of the proposed SAFA algorithm is outlined in Algorithm 2.

The server orchestrates the process holistically in rounds. At the beginning of each round, the server first checks the version of clients and distributes the latest global model in a lag-tolerant manner (see Eq. 3.7) given the parameter τ. Then the server begins to listen and collects the updates (i.e., trained local model) from clients. Clients train their native models on local datasets using the gradient descent method. Based on Algorithm 1, the clients missing the previous round will have the priority to be selected to meet the preset fraction C. Following the client selection, the server then executes the three-step discriminative aggregation, which merges all the entries in the cache into the global model, i.e., $w^{(t)}$, and updates the cache entries of the undrafted clients.

3.5 Analysis of Lag tolerance

This section analyses the impact of *lag tolerance* from different perspectives. As mentioned, this parameter is crucial to the pace control of the SAFA algorithm. When *lag tolerance* is small, clients/models become deprecated frequently, resulting in relatively high cost in model distribution. If it is set to a big value, the server will be very tolerant to the stragglers, which will probably cause high variance in the versions of local models and consequently slow down the convergence of the global model. Thus, this work introduces two holistic metrics: Synchronisation Ratio (SR) and Version Variance (VV). *SR* measures the usage of downlink by which the global model is distributed to the devices. *VV* is defined based on the version distribution of local updates. For SAFA,

$$SR_{SAFA} = \frac{1}{rm}\sum_{t=1}^{r}(|\bigcup_{v=t-1} U_v^{(t)}| + |\bigcup_{v<t-\tau} U_v^{(t)}|) \tag{3.12}$$

where r is the number of global rounds and N is the number of clients. *SR* is calculated based on the proposed lag-tolerant distribution rule (Eq. 3.7).

$$VV_{SAFA} = \frac{1}{r}\sum_{t=1}^{r}\text{var}(V_t) \tag{3.13}$$

where V_t is the version distribution of trained clients at round t, i.e., $V_t = \{v_1, v_2, ..., v_N\}$.

Lag tolerance (i.e., τ) varies from 1 to 10 and several groups of FL tests are set up for a regression task on the Boston Housing dataset. The maximum number of global rounds is set to 100. Apart from the best loss achieved (i.e., the minimum loss by the global model in 100 rounds), this work also presents the statistical results in the metrics including EUR, SR and VV.

Fig. 3.3(a) draws the best loss of the global model in the FL environment where the selection fraction C is set to 0.1, 0.5 and 1.0, and the expectation of client drop-out probability cr is set to 0.3 and 0.7, respectively. Fig. 3.3(b) shows the resulting synchronisation ratio (SR). Apparently small values of *lag tolerance* show a clear advantage in terms of loss. However, the overhead of communication (revealed by SR) is relatively large in the case where τ is set too small (e.g., 1, 2 or 3). This is expected because more clients will become deprecated and be forced to synchronise when the proposed algorithm are less tolerant to the stragglers with stale models.

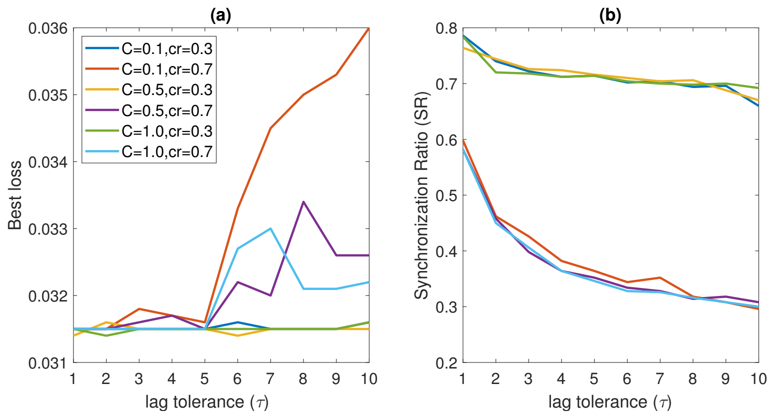


Figure 3.3: (a) Best loss achieved by the global model and (b) the synchronisation ratio over the federated optimisation with SAFA algorithm under different lag tolerance settings.

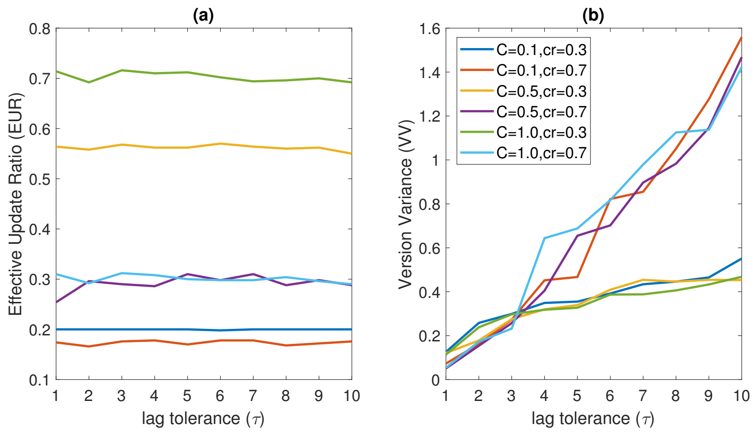


Figure 3.4: (a) Effective Update Ratio (EUR) and (b) Version Variance (VV) over the federated optimisation with SAFA algorithm under different lag tolerance settings.

There are multiple factors that can affect the best global model one can obtain by federated learning. This work analyses it based on the effective update ratio

(EUR) and the variance of version (VV) under different FL settings — these are two important metrics that well reflect the quality of the aggregation step, which is vital for the accuracy of the global model. From Fig. 3.4(a) one can see EUR basically remains level as the lag tolerance changes, and that EUR depends on both the client fraction C and the client crash probability cr. When cr is low (e.g., $cr = 0.3$), EUR is slightly above the percentage quota of the clients specified by C, which is because of the contribution by undrafted clients. In the case of a high crash rate (e.g., $cr = 0.7$), EUR is restricted at a low level as it is impossible to be higher than $\mathbb{E}(|U - Z|)$, which in theory is equal to $1 - cr$, i.e., the portion of clients with successfully committed updates. In addition, the plot of Version Variance in Fig. 3.4(b) reveals part of the reason why the quality of the global model degrades when *lag tolerance* is set too large (see Fig. 3.3(a)). In general VV increases if SAFA is more tolerant to the stragglers (i.e., a larger value of τ). It can be further observed from Fig. 3.4(b) that as τ increases, VV goes up at a much slower rate in relatively stable FL settings (e.g., $cr = 0.3$) than in the extreme settings (e.g., $cr = 0.7$). Combining Fig. 3.3(a) and Fig. 3.4(b), a clear correlation between VV and the quality of global model can be observed especially in an unstable environment where the clients disengage frequently.

Based on the observations, one conclusion can be drawn that a moderate *lag tolerance* can largely restrain the loss of the global model below a desired level and avoid high communication costs (indicated by SR) in sending out the global model. Therefore it is suggested that *lag tolerance* be set to 5 rounds in general.

3.6 Bias Analysis

This section theoretically analyses the bias in client selection introduced by the discrepancy of performance and reliability between clients. Here the bias between two clients (e.g., clients A and B) refers to the ratio of client A's chance of contributing to the global model to client B's chance. It is worth mentioning that FedAvg also incurs bias (even though it uses random selection before the training starts) because the clients drop or opt out with different frequencies.

In the analysis, an extreme case is considered to represent the worst bias between the clients. In this case, clients A and B are assumed to the most and least capable clients, respectively. Namely, clients A and B yield the shortest and longest local training time, respectively. Further, let us assume they behave independently and have the probabilities of dropping out in any round of training denoted by cr_A and cr_B, respectively. For the entire set of clients, we assume an overall crash ratio, denoted by R, which is the expected proportion of clients that drop out in an FL round. At the r-th round of training, the bias between clients A and B can be represented by:

$$bias^{(r)} = \frac{P^{(r)}(A)}{P^{(r)}(B)} \tag{3.14}$$

where $P^{(r)}(A)$ (or $P^{(r)}(B)$) denotes the probability that the local update of client A (or B) is successfully aggregated in the global aggregation step in round r.

First presented is the analysis of the bias generated by FedAvg, which selects clients at the beginning of each round and the server will wait for all those selected clients to submit local updates. The local update of a selected client will always be aggregated in this round unless this client crashed. Therefore, the bias in FedAvg only depends on the clients' crash rates, which can be modeled by:

$$bias^{(r)}_{FedAvg} = \frac{1 - cr_A}{1 - cr_B} \tag{3.15}$$

In SAFA, the situation is different. C percentage of the clients are selected from all clients that committed their local updates at the end of this round. The bias in SAFA not only depends on the crash rate, but also on the performance of the clients. A more powerful client can complete their local training faster and therefore its local update has a higher chance to be used in a round.

There are two possible cases where a local update can be used in the current round: i) when a client is selected by the server, its local update will be directly applied in the current round. This work denotes the probability of this case by $P^{(r)}_D(A)$; ii) the local update generated by an undrafted client in last round also has the chance to be used in the current round through the bypass scheme. The probability that this case occurs is denoted by $P^{(r)}_S(A)$. Apparently the two cases are exclusive because a client's model in the bypass will not be used if it gets selected. Therefore, $P^{(r)}(A)$ can be calculated by summing up $P^{(r)}_D(A)$ and $P^{(r)}_S(A)$. $P^{(r)}(B)$ is decomposed similarly.

Due to the space limitation, this work only presents the final expressions of $P^{(r)}(A)$ and $P^{(r)}(B)$ below in this section.

First, three cases need to be considered for client selection in SAFA given the fraction parameter C and crash ratio R:

case 1 $\iff C \geq 1 - R$

case 2 $\iff (1-C)(1-R) \leq C < 1-R$

case 3 $\iff C < (1-C)(1-R)$.

Literally, case 1 represents a deficit in client selection (i.e., too many crashes to fulfil the pick percentage C). Case 3 means that one can meet the selection ratio C by only selecting the arrived updates from clients not selected last round since they are prioritised by SAFA. Case 2 stands between cases 1 and 3. Namely, one can meet the selection ratio C by selecting the prioritised (i.e., last-round undrafted or crashed) clients first and then other clients who also committed their local updates in this round. Considering these cases, the following proposition is given:

Proposition 3.1. *The probabilities $P^{(r)}(A)$ and $P^{(r)}(B)$ can be formulated respectively by Eqs. (3.16) and (3.17) given $r > 1$:*

$$P^{(r)}(A) = \begin{cases} 1 - cr_A & \text{if case 1,} \\ 1 - cr_A & \text{if case 2,} \\ \sigma_A^{(r-1)} - cr_A^2 & \text{otherwise} \end{cases} \tag{3.16}$$

$$P^{(r)}(B) = \begin{cases} 1 - cr_B & \text{if case 1,} \\ \sigma_B^{(r-1)} - cr_B^2 & \text{if case 2,} \\ 1 - cr_B & \text{otherwise} \end{cases} \tag{3.17}$$

where $\sigma_A^{(k)} = 1 - P_D^{(k)}(A)$ and $\sigma_B^{(k)} = 1 - P_D^{(k)}(B)$.

Proof. By the definitions, $P^{(r)}(A)$ and $P^{(r)}(B)$ can be decomposed as the sum of two probabilities:

$$P^{(r)}(A) = P_D^{(r)}(A) + P_S^{(r)}(A) \tag{3.18}$$

$$P^{(r)}(B) = P_D^{(r)}(B) + P_S^{(r)}(B) \tag{3.19}$$

where $P_D^{(r)}(A)$ denotes the probability by which the update from client A goes directly into the cache, and $P_S^{(r)}(A)$ that client A's model in the bypass structure goes into the cache in round r. $P_D^{(r)}(B)$ and $P_S^{(r)}(B)$ are defined in a similar way. With the three cases considered, for client A:

$$P_D^{(r)}(A) = \begin{cases} 1 - cr_A & \text{if case 1,} \\ 1 - cr_A & \text{if case 2,} \\ (1 - cr_A)(1 - P_D^{(r-1)}(A)) & \text{otherwise} \end{cases} \tag{3.20}$$

$$P_S^{(r)}(A) = \begin{cases} 0 & \text{if case 1,} \\ 0 & \text{if case 2,} \\ cr_A(1 - P_D^{(r-1)}(A) - cr_A) & \text{otherwise} \end{cases} \tag{3.21}$$

The first two cases in Eqs. (3.20) and (3.21) indicate that client A, once finishing local training without crash, can always submit its update into the cache (for the upcoming aggregation). For case 3 (where $C < (1 - R - C)(1 - R)$), the chance for client A to be directly merged into cache equals to $(1 - cr_A)\sigma_A^{(r-1)}$ because two conditions need to be satisfied: being undrafted/crashed last round and being picked this round. The situation that client A's entry in the bypass takes effect in round r (i.e., $P_S^{(r)}(A)$ in case 3) only happens when the server ignores client A in both round $r - 1$ and r while A actually completed local training at round $r - 1$.

For client B one can derive that:

$$P_D^{(r)}(B) = \begin{cases} 1 - cr_B & \text{if case 1,} \\ (1 - cr_B)(1 - P_D^{(r-1)}(B)) & \text{if case 2,} \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

$$P_S^{(r)}(B) = \begin{cases} 0 & \text{if case 1,} \\ cr_B(1 - P_D^{(r-1)}(B) - cr_B) & \text{if case 2,} \\ 1 - cr_B & \text{otherwise} \end{cases} \tag{3.23}$$

The analysis for B is a bit more intuitive. In case 1(i.e., $C \geq 1 - R$), client B cannot have any bypass entry available because its update will always be merged into cache, and in case 2 client B's entry in bypass takes effect only when it crash this round and was undrafted last round. In case 3, client B never gets picked (thus no direct update to cache) because too many prioritized clients are expected in each round, leaving no chance for B as it is the slowest in local training. But it still has a chance to contribute via the bypass if it went undrafted in the previous round.

Considering the recurrence relation of $P_D^{(r)}(A)$ and $P_D^{(r)}(B)$ in Eqs. (3.20) and (3.22) in case 3 and 2 respectively, by resolving it one can derive the following expressions in terms of r:

$$P_D^{(r)}(A) = \frac{(cr_A - 1)^{r+1} + 1 - cr_A}{cr_A - 2}, \text{ for case 3} \tag{3.24}$$

$$P_D^{(r)}(B) = \frac{(cr_B - 1)^{r+1} + 1 - cr_B}{cr_B - 2}, \text{ for case 2} \tag{3.25}$$

Further, by defining $\sigma_A^{(k)} = 1 - P_D^{(k)}(A)$ and $\sigma_B^{(k)} = 1 - P_D^{(k)}(B)$, one can reformulate $P_D^{(r)}(A)$, $P_S^{(r)}(A)$, $P_D^{(r)}(B)$ and $P_S^{(r)}(B)$ as:

$$P_D^{(r)}(A) = \begin{cases} 1 - cr_A & \text{if case 1,} \\ 1 - cr_A & \text{if case 2,} \\ (1 - cr_A)\sigma_A^{(r-1)} & \text{otherwise} \end{cases} \tag{3.26}$$

$$P_S^{(r)}(A) = \begin{cases} 0 & \text{if case 1,} \\ 0 & \text{if case 2,} \\ cr_A(\sigma_A^{(r-1)} - cr_A) & \text{otherwise} \end{cases} \tag{3.27}$$

$$P_D^{(r)}(B) = \begin{cases} 1 - cr_B & \text{if case 1,} \\ (1 - cr_B)\sigma_B^{(r-1)} & \text{if case 2,} \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

$$P_S^{(r)}(B) = \begin{cases} 0 & \text{if case 1,} \\ cr_B(\sigma_B^{(r-1)} - cr_B) & \text{if case 2,} \\ 1 - cr_B & \text{otherwise.} \end{cases} \tag{3.29}$$

Combining all these results one can derive Eq. (3.16) and Eq. (3.17). □

With $P_D^{(r)}(A)$ and $P_D^{(r)}(B)$ (see Eqs. 3.20 and 3.22) formulated, one can derive $\sigma_A^{(k)}$ and $\sigma_B^{(k)}$:

$$\begin{cases} \sigma_A^{(k)} = \frac{2cr_A - (cr_A - 1)^{k+1} - 3}{cr_A - 2}, \\ \sigma_B^{(k)} = \frac{2cr_B - (cr_B - 1)^{k+1} - 3}{cr_B - 2}. \end{cases} \tag{3.30}$$

Therefore, combining Eqs. (3.16), (3.17) and the definition of bias, one can derive the bias introduced by SAFA in round r ($r > 1$) as follows:

$$bias_{SAFA}^{(r)} = \begin{cases} \frac{1-cr_A}{1-cr_B} & \text{if case 1,} \\ \frac{1-cr_A}{\sigma_B^{(r-1)}-cr_B^2} & \text{if case 2,} \\ \frac{\sigma_A^{(r-1)}-cr_A^2}{1-cr_B} & \text{otherwise.} \end{cases} \tag{3.31}$$

Fig. 3.5 visualises the bias of FedAvg and SAFA as a function of round index r. In case 1 where all local updates committed by the clients are aggregated, it introduces a fixed bias of $\frac{1-cr_A}{1-cr_B}$, which is the same as FedAvg. In case 2, client B, as the slowest one, will be picked (once it has committed) by the server as long as it was undrafted or crashed in the previous round, which effectively reduces the bias to a level below that of FedAvg. As for case 3, the quota (decided by C) will be fulfilled only with last-round undrafted or crashed clients. Assuming both A and B missed last round, client B is disadvantaged because the server is likely to end the round before B finishes training when the fraction C has been fulfilled by other faster, prioritised clients (including client A). In all these cases, the bias between A and B converges after a few rounds once FL starts.

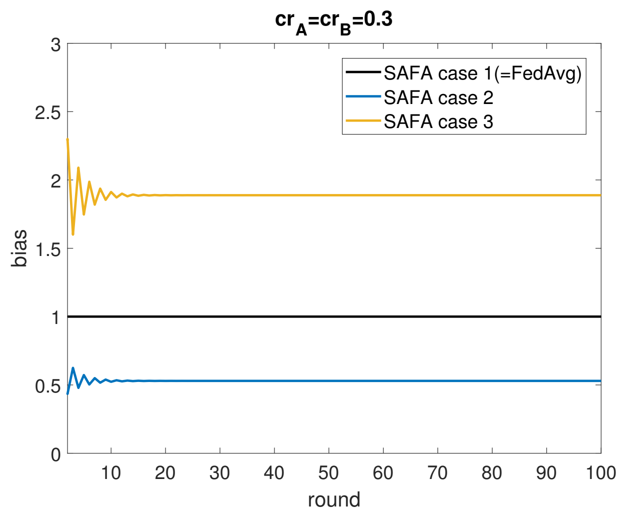


Figure 3.5: The bias incurred by FedAvg and SAFA (in the circumstances of three different cases) as a function of federated round index.

3.7 Experiments

3.7.1 Experimental Setup

The experiments were conducted in a simulation-based, discrete event-driven system built using Python 3.7 and PyTorch (Build: 1.7.0) on a dual-core (CPU model: Intel i5-8500) physical machine equipped with Nvidia GeForce GTX 1050Ti graphics card. The client-side local training logic was implemented under the PyTorch framework with CUDA (version: 11.1) support. Local training was run sequentially in the runtime but considered parallel by the server in the discrete-event simulation system. Fig. 3.6 visualises an FL cycle in the simulated system where experimental statistics are collected through event processing. End devices were initialised with heterogeneous performance specifications and heterogeneous local datasets. A star topology was established for this system. Using a local network setting similar to that in [139], the average throughputs of the clients and the server are 1.4 Mbps and 10 Gbps, respectively. The client–server communication was implemented via in-memory tensor exchange. The server maintains a simulated wall clock and uses the equations provided in the System Model (Section 3.3) to calculate all the time consumptions. The advantages of customised simulation over existing libraries/platforms are i) the flexibility in algorithm design and runtime optimisation, and ii) easy configuration of the environment, models and result analyser.

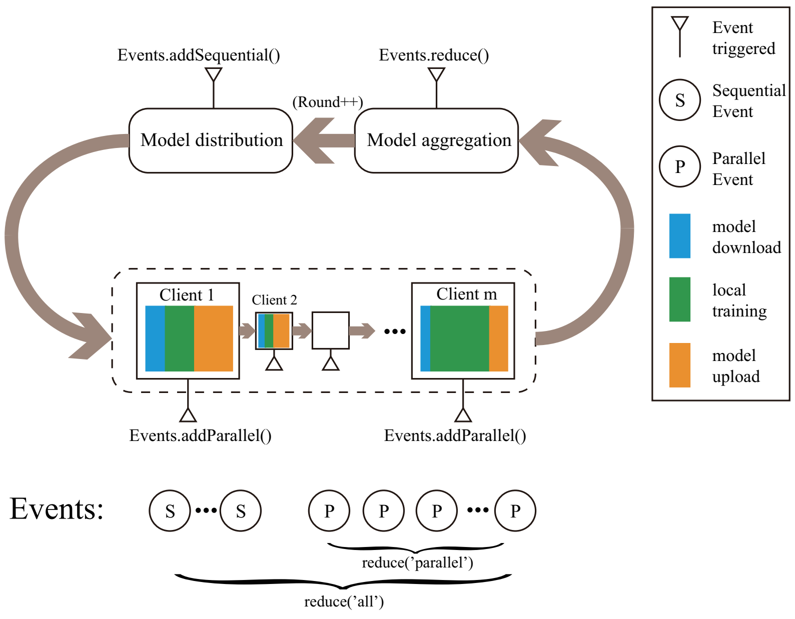


Figure 3.6: Visualisation of the FL cycle in the simulated system.

The effectiveness of SAFA was evaluated on three typical machine learning tasks. *Task 1* is to fit a regression model on the public Boston Housing dataset[1], which is available in public repositories. *Task 2* is to learn a handwritten digit image classification model implemented using a convolutional neural network (CNN), which is comprised of two 5×5 convolution layers (the first one with 20 channels and the second with 50 channels) with 2×2 max pooling, a fully-connected layer with ReLu as the activation function, and a final softmax output layer. This lightweight CNN is suitable for end devices with small memory size and is also adopted in the experiment by [47]. *Task 3* is to learn a classification model for detecting network intrusion given the TCP dump data. For this task, TCP-connection examples are extracted from the KDD Cup'99 dataset[2] and a Support Vector Machine (SVM) is used as the classification model.

Separate environments were set up for these three learning tasks to investigate the performance of the proposed algorithm in different FL settings. To simulate the unreliability of clients, this work sets a crash probability (cr) in each run of test and assume each client has the equal chance cr to drop out in any round of federated training. The round length limits T_{lim} were set differently for the tasks to ensure that probabilistically 97% of the clients (randomly initialised as per the distribution) will be able to finish the round if they do not drop out. The details of the experimental setup are listed in Table 3.2.

Table 3.2: Experimental setup

Parameter	symbol	Task 1	Task 2	Task 3
Dataset	D	Boston	MNIST	KDDCup99
Number of features	d	13	28×28	35
Model	w	Regression	CNN	SVM
Dataset size	n	506	70k	186k
Number of clients	N	5	100	500
Local Data sizes	-	$\mathcal{N}(71, 21^2)$	$\mathcal{N}(600, 180^2)$	$\mathcal{N}(260, 78)$
Clients performance	s_k	Randomly generated from Exp($\lambda = 1.0$)		
Avg. drop-out prob.	cr	$\{0.1, 0.3, 0.5, 0.7\}$		
Participation fraction	C	$\{0.1, 0.3, 0.5, 0.7, 1.0\}$		
Max number of rounds	R	100	50	100
Round length limit	T_{lim}	830s	5600s	1620s
Number of local epochs	E	3	5	5
Mini-batch size	B	5	40	100
Learning rate	lr	1e-4	1e-3	1e-2

To simulate data imbalance and the heterogeneity in end devices, this work assumes

[1]`https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`
[2]`https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

that the size of data partitions (i.e., local data size) follows the Gaussian distribution $\mathcal{N}(\mu, 0.3\mu)$ where $\mu = n/N$, and that clients' performance follows the exponential distribution with $\lambda = 1.0$. Here the performance of a client is defined as the number of batches it can process per second in training. End devices (i.e., clients) may be unreliable and crash occasionally with a probability of ρ_k. In the experiment, the clients are assumed to crash independently with the same probability in any federated round and set ρ_k to be cr, i.e. $\rho_k = cr, k = 1, 2, ..., N$. The size of each model is set to 10MB considering the compression ratio for neural networks [269].

For comparison, FedAvg [47], FedCS and a fully local training process were also implemented. FedCS [139] is a refined FL algorithm that estimates the speed that clients work and filters out some slow clients proactively (at the stage of client selection) to improve the overall efficiency of FL. The fully local algorithm never performs the global aggregation until the end of the final round.

3.7.2 Evaluation Results

This section presents the results of experiments and discusses the evaluated FL algorithms in terms of the quality of the obtained global model (shown in Figs. 3.7, 3.8 and 3.9, with more details in Tables A.1, A.3 and A.5 in Appendix A) as well as holistic metrics including round efficiency (summarised in Tables 3.4, 3.6 and 3.8), communication overheads (in Tables 3.5, 3.7 and 3.9) and local resource utilisation (Tables A.2, A.4 and A.6 in Appendix A).

For different machine learning models, this work defines their accuracy in different ways, as shown in Table 3.3. In the table, y and $\hat{y}$ denote the label and the output of the model, respectively. The function $\phi(\cdot)$ returns 1 if $\hat{y}$ matches y, otherwise it returns 0. The loss functions for the three tasks are Mean Square Error (MSE), Negative Log Likelihood (NLL) and Hinge loss.

Table 3.3: Evaluation metrics for the global model in the three tasks

ML task	Loss function	Accuracy formulation
Task 1: regression	MSE	$1 - \frac{1}{n}\sum_{i=1}^{n} \lvert \frac{y_i - \hat{y_i}}{\max(y_i, \hat{y_i})} \rvert$
Task 2: image classification	NLL	$\frac{1}{n}\sum_{i=1}^{n} \phi(y_i, \hat{y_i})$
Task 3: binary classification	Hinge	$\frac{1}{n}\sum_{i=1}^{n} \max(0, \mathrm{sign}(y_i \cdot \hat{y_i}))$

Task 1: Regression

This task aims to learn a regression model on a small group of clients to predict the median value of a house in the area of Boston Mass. Input features include 13 properties about the estate such as average number of rooms per dwelling and crime rate. In this experiment, FL was run with every candidate algorithm (i.e., SAFA, FedAvg, FedCS and fully local training). Their effectiveness is compared in terms of the achieved accuracy of the global model, round efficiency and communication overhead.

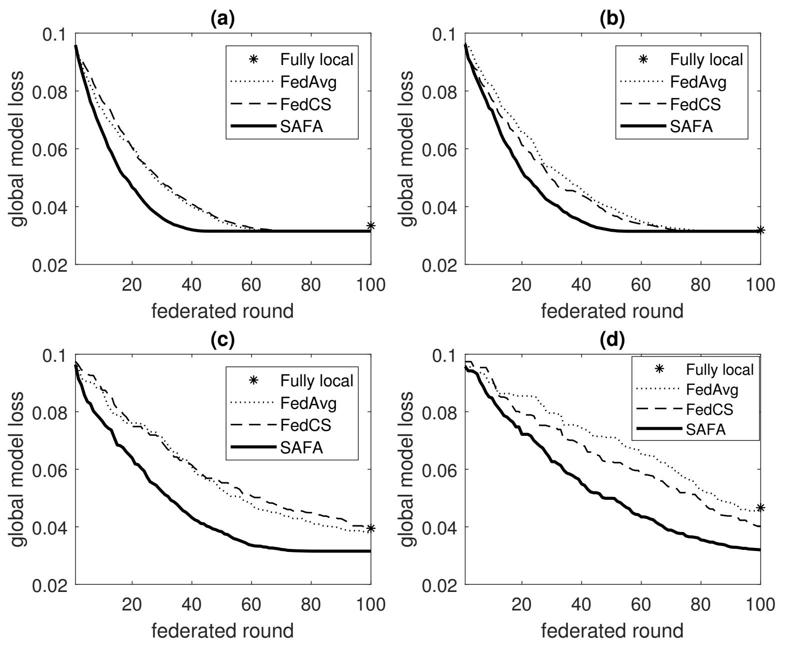


Figure 3.7: The loss trace of the global model as the FL process progresses on Task 1 where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5 and 0.7 for the four sub-figures (a)-(d), respectively

It can be seen from Fig. 3.7 and Table A.1 (in Appendix A) that SAFA significantly improves the convergence rate as well as the best accuracy achieved by the global regression model, especially under settings of unstable environments (i.e., $cr \geq 0.5$). This is mainly attributed to the proposed staleness-tolerant mechanism. Another advantage of the tolerance to stragglers is the preservation of local training results. The metric *Futility Percentage* is used to measure the percentage of local progress that is wasted due to the model synchronisation forced by the server (FedAvg and FedCS force the selected clients to overwrite its local model with the latest global model). Results of *SR* and futility percentage in Table A.2 show that the wasted

Table 3.4: Average length of a federated round in secs on Task 1 wherein each algorithm was tested with varying selection fraction under different environment settings. Round time limit is set to 830s considering the client performance and data distribution.

Avg. round length (Task 1: regression)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	316.22	489.37	586.90	731.12	808.59
0.3	429.63	652.39	641.40	736.53	832.02
0.5	372.43	495.37	475.14	621.91	676.41
0.7	354.34	405.86	593.10	728.25	661.67
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	207.50	487.47	564.20	656.49	786.96
0.3	336.97	519.58	651.23	401.95	832.02
0.5	186.51	221.46	467.98	621.91	676.41
0.7	195.09	398.81	584.68	393.09	661.67
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	149.69	389.44	540.41	606.48	734.40
0.3	202.44	430.68	583.22	371.77	699.23
0.5	169.33	215.66	408.85	510.85	508.23
0.7	161.81	293.09	402.18	411.06	379.29

Table 3.5: Average model distribution overhead (unit: seconds) on Task 1

Avg. T_{dist} (Task 1: regression)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.40	0.81	1.21	1.62	2.02
0.3	0.40	0.81	1.21	1.62	2.02
0.5	0.40	0.81	1.21	1.62	2.02
0.7	0.40	0.81	1.21	1.62	2.02
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.33	0.81	1.21	1.62	2.02
0.3	0.40	0.81	1.21	1.31	2.02
0.5	0.33	0.64	1.21	1.62	2.02
0.7	0.33	0.81	1.21	1.29	2.02
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	1.84	1.83	1.80	1.84	1.81
0.3	1.49	1.46	1.43	1.40	1.41
0.5	1.00	1.07	0.96	1.05	1.02
0.7	0.76	0.69	0.77	0.75	0.74

training progress is reduced by SAFA effectively.

As shown in Tables 3.4 and 3.5, there is not much difference in average round length and model distribution overhead due to the very limited number of devices used to run task 1. But one can still observe notable efficiency boost and convergence speedup by SAFA under the circumstance where the selection fraction C is very small. With C set to 0.1, SAFA halves the time required to finish a federated round compared to FedAvg.

Task 2: CNN

The MNIST dataset is divided into N partitions of which the sizes are random variables (following a Gaussian distribution). The CNN models with randomly initialised weights were created on 100 clients.

Table 3.6: Average length of a federated round in secs on Task 2 wherein each algorithm was tested with varying selection fraction under different environment settings. Round time limit is set to 5600s considering the client performance and data distribution.

Avg. round length (Task 2: CNN)					
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	3402.55	5557.25	5610.20	5614.28	5620.40
0.3	5410.97	5606.12	5610.20	5614.28	5620.40
0.5	5602.04	5606.12	5610.20	5614.28	5620.40
0.7	5602.04	5606.12	5610.20	5614.28	5620.40
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	1487.96	2133.02	3668.70	1871.65	1982.91
0.3	1261.59	1542.61	3132.86	2349.46	5395.54
0.5	1273.37	1642.59	3025.75	2876.63	3162.02
0.7	1253.74	1969.28	2180.46	4344.88	2530.01
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	198.28	315.33	3703.81	1708.93	1947.90
0.3	206.88	368.01	2691.25	1899.23	2149.23
0.5	203.48	800.64	2573.60	2727.25	2186.67
0.7	241.86	1893.14	1877.30	2619.79	2340.80

As a result, the Fully Local algorithm can finish with an accuracy around 90% on this classification task with the CNN model, while FedAvg, FedCS and SAFA can raise that to 96.0% $\sim$ 98.0% (Table A.3 in Appendix A). SAFA shows a significant advantage in round efficiency (see Table 3.6) — it is able to achieve up to 27× and 6× speed-up compared to FedAvg and FedCS in an unreliable environment where clients frequently opt/drop out and only a small fraction (i.e., $C = 0.1$ or 0.3) of them are allowed to participate in a round.

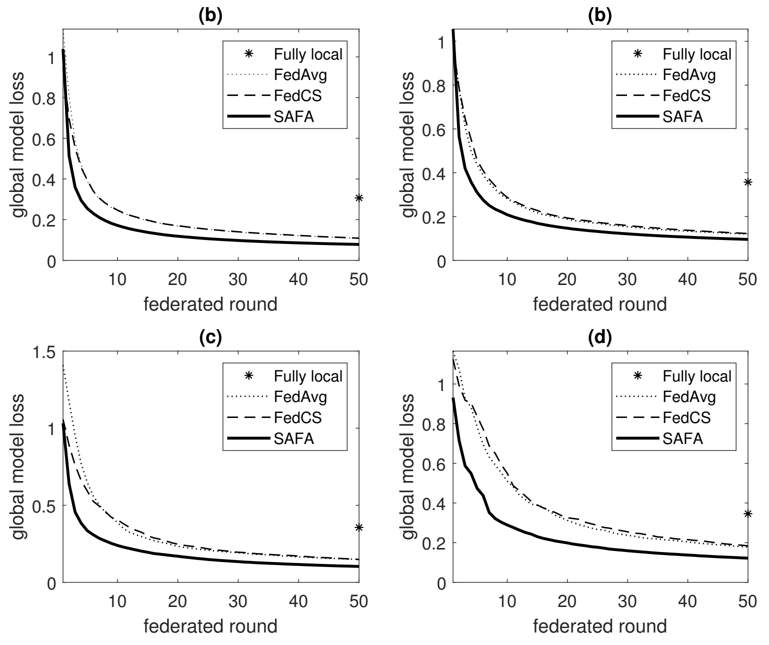


Figure 3.8: The loss trace of the global model as the FL process progresses on Task 2 where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5 and 0.7 for the four sub-figures (a)-(d), respectively.

Table 3.7: Average model distribution overhead (unit: seconds) on Task 2

Avg. T_{dist} (Task 2: CNN)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	2.04	6.12	10.20	14.28	20.40
0.3	2.04	6.12	10.20	14.28	20.40
0.5	2.04	6.12	10.20	14.28	20.40
0.7	2.04	6.12	10.20	14.28	20.40
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	2.04	6.12	10.20	14.14	20.40
0.3	2.02	6.05	10.20	14.13	20.40
0.5	2.04	6.06	10.20	14.28	20.20
0.7	2.04	6.12	10.11	14.28	20.20
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	18.27	18.45	18.26	18.47	18.38
0.3	14.45	14.65	14.48	14.54	14.69
0.5	10.89	10.51	10.70	10.84	10.58
0.7	7.17	7.23	7.55	7.21	7.41

The average T_{dist} for SAFA mainly depends on client crash probability (see Table 3.7), and it remains at a low level with $cr \geq 0.5$. In the case where devices are more reliable in local training (i.e., $cr < 0.5$), SAFA embraces a greater number of updates and results in a slightly higher cost (of tens of seconds) during the stage of model distribution, but the overhead is still acceptable considering the overall length of a federated round (which could last thousands of seconds, see Table 3.6).

Task 3: SVM

For this task a relatively large data set is used containing 186,480 TCP dump records including several types of network intrusions. The target is to learn a global SVM model to recognise malicious connections. The dataset is dispersed onto 500 clients to perform FL.

Table 3.8: Average length of a federated round in secs on Task 3 wherein each algorithm was tested with varying selection fraction under different environment settings. Round time limit is set to 1620s considering the client performance and data distribution.

Avg. round length (Task 3: SVM)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	1640.20	1680.60	1721.00	1761.40	1822.00
0.3	1640.20	1680.60	1721.00	1761.40	1822.00
0.5	1640.20	1680.60	1721.00	1761.40	1822.00
0.7	1640.20	1680.60	1721.00	1761.40	1822.00
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	788.75	1319.17	1607.42	1539.14	1802.09
0.3	685.26	1216.12	1521.82	1617.97	1775.50
0.5	714.73	1229.87	1371.03	1605.23	1821.60
0.7	754.52	1190.44	1526.23	1573.42	1731.65
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	310.70	353.98	1419.29	1514.38	1802.15
0.3	274.03	330.32	1499.79	1559.50	1762.51
0.5	242.93	398.27	1317.91	1476.14	1724.52
0.7	212.52	1187.96	1313.99	1223.72	1690.61

Table A.5 (in Appendix A) shows that FedAvg, FedCS and SAFA can produce very accurate global models (with the classification accuracy of over 99%) after convergence. SAFA could incur higher overhead in model distribution (as the *SR* is larger for SAFA, see Tables 3.9 and A.6 in some cases). Nevertheless, SAFA still significantly outperforms FedAvg and FedCS by 7.7× and 3.7×, respectively, in average round length (see Table 3.8). Its advantage decreases as more clients are set to engage in training but it is still the most efficient algorithm. In contrast to FedAvg

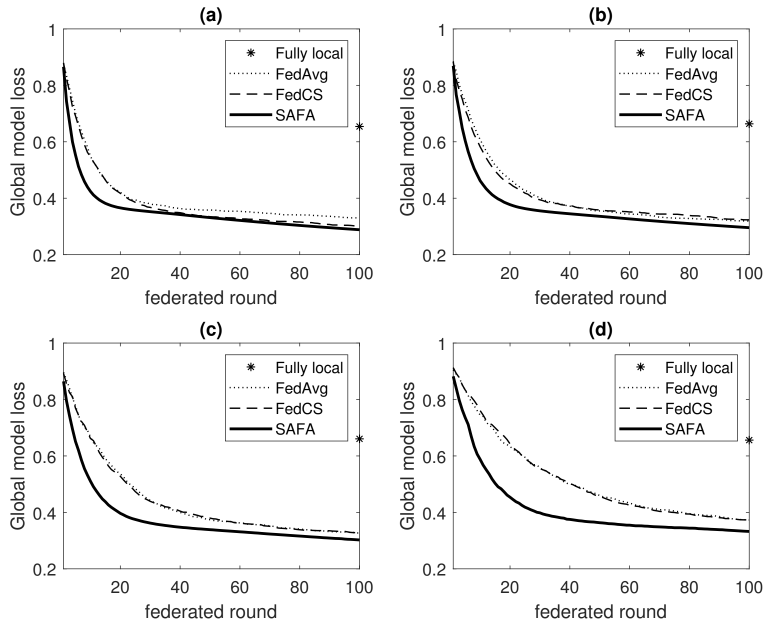


Figure 3.9: The loss trace of the global model as the FL process progresses on Task 3, where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5 and 0.7 for the four sub-figures (a)-(d) respectively

Table 3.9: Average model distribution overhead (in seconds) on Task 3

Avg. T_{dist} (Task 3: SVM)					
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	20.20	60.60	101.00	141.40	202.00
0.3	20.20	60.60	101.00	141.40	202.00
0.5	20.20	60.60	101.00	141.40	202.00
0.7	20.20	60.60	101.00	141.40	202.00
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	20.20	60.48	100.78	140.79	201.60
0.3	20.11	60.60	100.81	141.09	201.60
0.5	20.13	60.60	100.85	140.84	201.60
0.7	20.20	60.60	100.61	141.14	201.19
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	181.95	182.32	181.49	181.84	182.15
0.3	142.89	141.91	141.95	142.50	142.81
0.5	104.38	104.56	105.34	104.59	104.52
0.7	70.62	70.63	70.55	70.05	70.61

and FedCS, SAFA capitalises the contribution from straggling clients effectively, leading to a very small futility percentage (below 4%, see Table A.6 in Appendix A) on this task, which means that the majority of local training progress can make contributions to the convergence of the final global model, even in a very unreliable environment.

3.7.3 Discussion

The experimental results with several tasks including regression and classification demonstrate the effectiveness of applying the proposed semi-asynchronous algorithm to FL over unreliable clients. The improvement achieved by SAFA lies in three factors: i) faster convergence of the global model and a higher accuracy achieved, ii) significant reduction in average round length, and iii) increased utilisation of local training progress made by the stragglers. A few interesting phenomena were also observed in the experiments. First and foremost, this work finds that increasing the client fraction C may not significantly improve the quality of the global model. For example, a reasonably high accuracy is obtained by setting C to 0.3 or 0.5 (instead of 1.0) in task 2 in the case of a low crash probability. This in some ways implies that involving more clients each round could have very limited benefit. In addition, it is noticed that fully local training without round-wise aggregation is in some cases able to produce a reasonably good model, e.g., in the cases of Task 1 with $C = 0.3$ and Task 3 with $C = 0.1$ and $cr = 0.7$. Also, sometimes the synchronous FL algorithm FedAvg can produce a global model slightly better than the proposed solution in the case of $C = 1.0$, i.e., trying to involve all clients in every round. This advantage is probably brought by the feature that pure synchronisation can avoid the negative effect from stale models, which amplifies as a larger fraction of clients get involved. However, it is practically unrealistic to set a big C for FL because communication could be expensive while the enhancement of the resulting accuracy is very limited (see Tables A.1, A.3 and A.5 in Appendix A).

3.8 Conclusion

It is very challenging to guarantee the efficiency of FL considering the unreliable nature of end devices while the cost of device–server communication cannot be neglected. With the aim of improving the efficiency of federated learning over unreliable end devices, a semi-asynchronous algorithm is proposed which incorporates a novel client selection algorithm decoupling the central server and the selected clients for a reduction of average round time as well as a lag-tolerant mechanism in model distribution for tackling the tradeoff between faster convergence and lower communication overhead. This work also analyses the upper bound of the bias introduced by using SAFA in FL. The results of experimental evaluation on three typical machine learning tasks show that the proposed algorithm effectively enhances the round efficiency of federated optimisation process, improves the quality of the global model and reduces local resource wastage at a relatively low cost of communication.

Considering the subtle correlation between local models and the global model, a future plan is to look into the balance between generating the best local models for end devices and obtaining an optimal global model in the central server. Besides, it is also necessary to investigate how to further improve federated learning using model parallelism and compression.

Chapter 4

Efficient Multi-layer FL in MEC Systems

The prevalence of Internet of Things (IoT) and Edge Intelligence [270, 271] stimulates the efforts of pushing the computation to the edge of the network (closer to where the source of data resides) for faster response and better service quality [272]. With these two streams of research endeavour, it has been a major trend to empower the edge devices with AI applications. Mobile Edge Computing (MEC) [273, 274], which consists of cloud, edge nodes and end devices, is an emerging technology that can serve as the fundamental architecture for sinking AI to the edge [271] — it is forecast that the global edge computing market will reach 61.14 billion dollars by 2028 [275].

However, there are still many obstacles when it comes to practical AI scenarios where the participants of the model training process are end devices such as cell phones, smart sensors and wearable electronics. Centralised training may not be feasible in many sensitive scenarios (e.g., clinical diagnosis [276], intelligent surveillance [190] and typing assistants [15]), whilst distributed training has to be adapted to realistic MEC systems that are usually comprised of low-spec, unreliable end devices, geographically distributed edge nodes with moderate performance and relatively low-speed, noisy communication channels.

4.1 Motivation

4.1.1 Scalability

Mobile Edge Computing provides a natural production environment for federated learning based applications. A number of studies have provided their solutions to such adaptation. For example, Wang et al. [55] take into account the resource budgets for edge nodes and proposed a pace control algorithm that adaptively adjusts the aggregation interval of FL. They adopt a system architecture in which the data reside on edge nodes, which essentially makes the system still a two-layer FL without utilising the computing power of end devices. In the wireless edge environment, the joint optimisation problem of model convergence and communication cost has also been extensively studied [120, 135, 156]. However, these approaches are confined to the canonical two-layer FL framework where the system scale is limited to the coverage of an edge node such as a base station (BS) [94, 149]. From a systematic perspective, most of the existing solutions are running 'local-area FL' despite the fact that the whole edge layer is actually interconnected via the core network [178]. 'Local-area FL' is not the desiderata for large-scale applications but just a partial solution that turns 'data islands' into 'region islands' (Fig. 4.1). In other words, the lack of scalability is hindering the utility of FL in MEC circumstances.

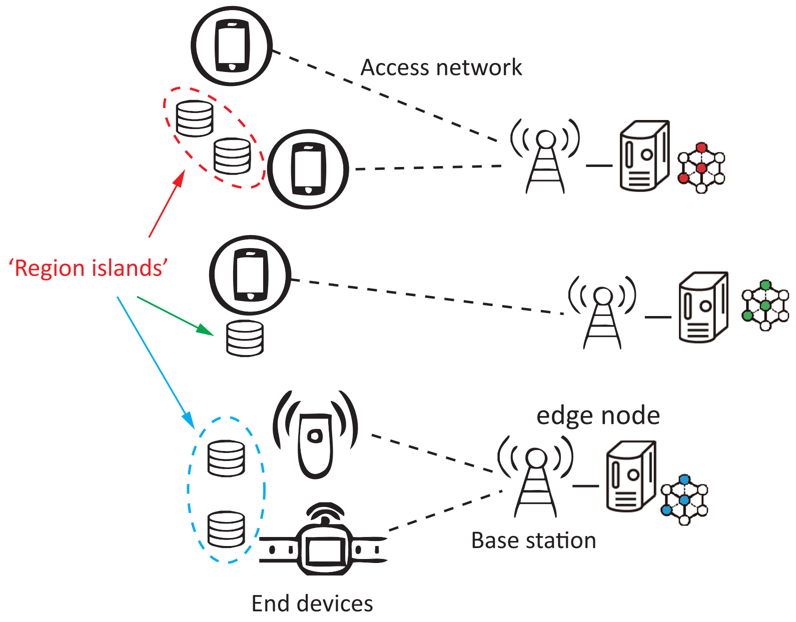


Figure 4.1: An illustration of 'region islands' caused by 'local-area FL'.

4.1.2 Communication Bottleneck

We are witnessing a sharp growth of computational power for end devices (also termed User Equipments in the literature). An increasing number of mobile phones are now equipped with high-frequency CPUs and even neural (co-)processors. Most of these devices are more than capable of running FL applications as clients. On the contrary, communication resources at the edge hardly suffice as the upgrade of infrastructures is usually slower than the surge of demands. As a result, communication has become the bottleneck at the edge [277]. A widely adopted approach to addressing the communication problem is model (or model update) compression [80, 86, 278]. But compression often comes at the cost of performance loss and may cause convergence problems when combined with artificial noise for privacy protection.

Taking advantage of the MEC system is another option. The basis idea is to leverage the scale of system to 'cover' the impact of unreliable local communication. Liu et al. [279] implemented a hierarchical FL algorithm that utilises the cloud server and the edge nodes to perform two levels of model aggregation. The algorithm is a straightforward extension of FedAvg, allowing multiple rounds in the edge layer before a global aggregation by the cloud. A main problem of the work is that each pair of interactions (i.e., client–edge and edge–cloud) is tightly coupled. As a result, the communication bottleneck at the edge is still unsolved (see Fig. 4.2). In addition, in HierFAVG, edge nodes have to perform multiple rounds of edge-level aggregation before sending models to the cloud. This significantly postpones the global exchange of model information and consequently slows down the convergence.

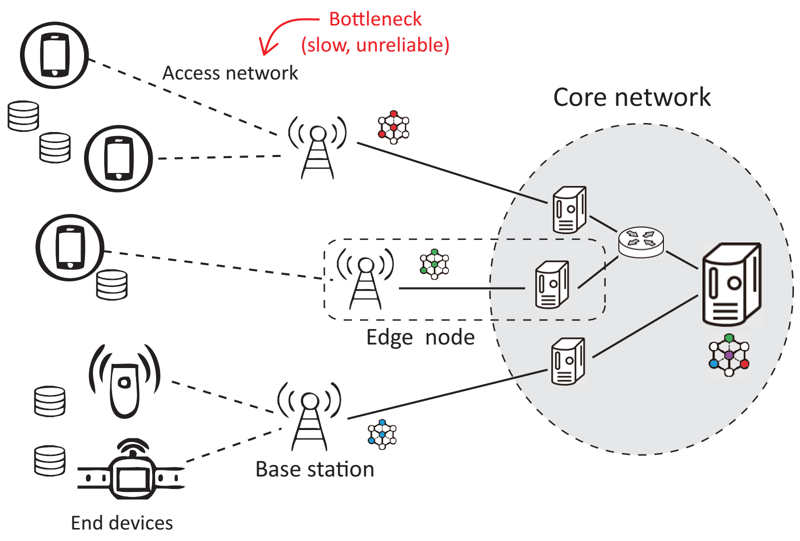


Figure 4.2: The access network at the edge is usually the bottleneck of FL.

Considering the drawbacks of the existing solutions in driving FL in the MEC systems, the aim of this work is to develop a more efficient FL algorithm that enables fast, robust model training by leveraging the pervasive data at the edge while addressing the communication bottleneck.

4.2 Hierarchical FL: Benefits and Challenges

Mobile Edge Computing, which incorporates the cloud, edge nodes and end devices, has shown great potential in bringing data processing closer to the data sources. Meanwhile, federated learning has emerged as a promising privacy-preserving approach to facilitating AI applications for user devices. Yet the adaptation of FL to the three-layer MEC system still needs much exploration. This section analyses the potential benefits and obstacles.

4.2.1 Benefits

Motivated by the potential of hierarchical FL, this work takes the approach of integrating the FL framework with the multi-layer architecture of MEC systems. The expected benefits are:

- **Scalability**: the reach of FL can be extended to the entire edge layer without being restricted by the connectivity of any edge nodes.
- **Model generalisation**: the collaborative training can be performed over an increased magnitude of devices across multiple cellular regions to achieve better generalisation of the global model.
- **Controllable cost**: scaling up the FL framework only incurs a marginal cost of communication system-wise.

4.2.2 Challenges

It remains a big challenge to optimise the efficiency and effectiveness of FL when it is integrated with the MEC architecture. The difficulties lie in the following aspects:

- **Adaptation**: the standard FL is a two-layer framework with a canonical client–server architecture, but the abstraction of a MEC system consists of three layers while a practical one can be even more complex.

- **Heterogeneity**: extending FL across multiple communication regions may increase the heterogeneity of devices and make it more challenging for the collaborative training in FL.
- **Resource restrictions**: wireless communication resources at the edge are hardly sufficient and thus the number of active devices within each region are usually restricted.

In view of the benefits and challenges, the research to be presented in this chapter focuses on designing a multi-layer FL algorithm for the MEC architecture with flexible communication and participation mechanisms to enable scalable, system-wise federated learning while mitigating the unreliability and heterogeneity of end devices.

4.3 Contributions

The main contributions of this work are outlined as follows:

- A novel algorithm (HybridFL) to drive the FL process in the three-layer architecture of MEC. HybridFL facilitates efficient model exchanges via the combination of quota-triggered regional aggregation (via the edge layer) and immediate cloud aggregation.
- A mechanism for mitigating the impact of client drop-out by introducing a regional slack factor for each edge node (i.e. region) into the client selection step via a probabilistic estimation method under strong privacy-preserving conditions that cause clients' reliability to be unknown.
- Extensive experiments with machine learning tasks (using two public data sets) for performance evaluation of the proposed HybridFL algorithm. Experimental results demonstrate significant improvement in average round duration, global model's convergence speed and accuracy and the energy consumption of end devices.

4.4 Client Reliability Unknown Federated Learning in MEC

In this section, hierarchical FL in the three-layer MEC system is investigated and a novel algorithm (HybridFL) is presented for privacy-preserving, efficient federated learning under the MEC architecture. Fig. 4.3 shows an overview of the proposed algorithm. This work proposes to leverage the capability of edge nodes to boost the efficiency of communication and adapt FL to the three-layer hierarchy of MEC by making the FL process a hybrid of client–edge and edge–cloud collaborations. Both the unreliability and heterogeneity of end devices are taken into account under strong privacy conditions. A novel pace control mechanism is adopted as a hybrid of synchronous (edge–cloud) and asynchronous (client–edge) communication.

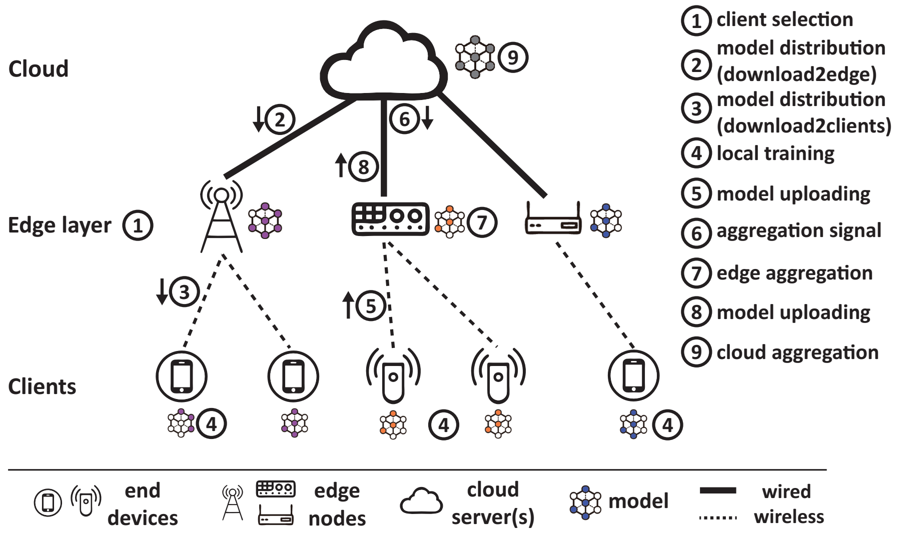


Figure 4.3: A schematic overview of the HybridFL algorithm designed for the MEC architecture.

4.4.1 System Overview

The proposed three-layer FL framework is depicted in Fig. 4.3 where end devices (e.g., mobile phones and IoT equipments) are the clients to perform local training, edge nodes (e.g., base stations and edge servers) provide wireless access to different regions of clients, and the cloud acts as a central coordinator that orchestrates the whole FL process. The two levels of model aggregation are controlled by the 'aggregation signal' sent from the cloud based on the information of local models collected by the edge nodes. In the system, clients are connected to the edge nodes via relatively low-speed (typically wireless), shared (therefore noisy) channels, whilst the edge–cloud connection is more stable (typically through Ethernet) and the bandwidth is typically sufficient. In this work, the collection of clients connected to an edge node is referred to as a *region*. D_r denotes the set of data in region r (note that the data cannot leave their end devices), i.e., $D_r = \{D_{r,k} | \forall k \in \text{region}\, r\}$. Without loss of generality, it is assumed that a client can only connect to a single edge node and that clients are independent of each other regarding their behaviours. Table 4.1 lists the notations frequently used in this chapter. Other notations (e.g., the number of clients N, selection fraction C, etc.) are defined in consistency with those in previous chapters.

Table 4.1: List of symbols

Symbol	Description
D	the complete dataset
$D_{r,k}$	the data partition on client k in region r
D_r	the (logical) set of data in region r
N_r	the number of clients connected to edge node r
M	the number of edge nodes (regions)
V	the set of edge nodes
U_r	the set of clients in region r ($\lvert U_r\rvert = N_r$)
w	parameters of the global model
w_r	parameters of the model on edge node r
$w_{r,k}$	parameters of the local model on client k
C_r	the proportion of clients selected in region r
$S_r^{(t)}$	the set of selected clients within region r in round t
$X^{(t)}$	the set of all clients (across all regions) that do not drop out in round t
$X_r^{(t)}$	the set of clients in $X^{(t)}$ belonging to region r
$\Psi^{(t)}$	the set of clients that submit their models in time in round t
$\Psi_r^{(t)}$	the set of clients in $\Psi^{(t)}$ belonging to region r

Formally, the proposed solution is to solve the following global optimisation problem at any given round t:

$$\arg\min_{w} F(w,t) = \sum_{k \in U} \gamma(k, r(k), t) F_k(w) \tag{4.1}$$

where w denotes the parameters (e.g., weights for neural nets) of the global model to be optimised, $r(k)$ is the region index for client k, and $F_k(w)$ is local objective for the training on client k. The coefficient $\gamma(k, r(k), t)$ is the weight of client k's model. Without ambiguity, the abbreviation $\gamma(k, r, t)$ is used to denote $\gamma(k, r(k), t)$ for brevity. The value of $\gamma(k, r, t)$ depends on the volume of client k's local data, the joint volume of data on devices within region $r(k)$ and effective data coverage (EDC) for round t. The formal definitions of $\gamma(k, r(k), t)$ and EDC will be presented later.

The local objective $F_k(w)$ is the average loss on client k's local dataset:

$$F_k(w) = \frac{1}{|D_{r(k),k}|} \sum_{(X_i,Y_i)\in D_{r(k),k}} f(w; X_i, Y_i) \tag{4.2}$$

where $f(\cdot)$ is the loss function and $D_{r(k),k}$ is the data possessed by client k with $r(k)$ being its region.

There are eight steps in each round of training using the HybridFL algorithm to drive FL in the MEC architecture (see Fig. 4.3). These eight steps comprise three basic stages: *model distribution*, *local training* and *model aggregation*. The stage of model distribution starts with client selection (step 1, Fig. 4.3), after which the (latest) global model is distributed over the edge nodes (step 2) and then across all the clients (step 3). The second stage, local training, is performed on the clients (end devices) and also includes the steps of model downloading and uploading via the client–edge connections. Aggregation is the final stage of a round where the local models (selected and uploaded) are merged into a global model. The proposed algorithm adopts a hybrid pace control mechanism that allows flexible control over the edge-level (i.e, regional) model aggregation, which is signified by the cloud.

In the first stage of any FL round, client selection is often performed to ensure that only a reasonable proportion of clients are engaged in training this round. For example, the number of selected clients is determined by the proportion C in [12, 47]. As pointed out by Kairouz et al. [49], it is necessary to restrict the participating population to a small fraction for two reasons. First, it has been shown that involving an excessive number of clients can hardly benefit the convergence and quality of the global model [47]. Second, recruiting excessive devices is neither cost-efficient in communication nor realistic for the end device owners.

Nevertheless, a severe shortage of participants in FL also leads to an inferior global model, of which the unreliability of end devices is the main cause. These devices can opt out of any round of local training or drop out occasionally due to various reasons such as low battery, device failure or network disconnection. Let $X_r^{(t)}$ denote the set of clients who are in region r and do not drop out of round t. Then, since the client may drop-out, manually or unexpectedly, one can expect $|X_r^{(t)}| \le C \cdot N_r$ and $|X^{(t)}| = \sum_{r=1}^{M} |X_r^{(t)}| \le C \cdot N$, where C is the desired proportion of clients with successful model submission (C is preset by the cloud server).

In order to mitigate the shortage of participants caused by drop-out, this work introduces $C_r^{(t)}$ as the region-wise selection proportion into the client selection step (i.e., step 1 in Fig. 4.3) of the HybridFL training process at the start of each round. More specifically, an edge node r will determine $C_r^{(t)}$ and select a fraction of $C_r^{(t)} \cdot N_r$ (instead of $C \cdot N_r$) clients randomly before signifying these clients to begin local training in round t. An ideal value of $C_r^{(t)}$ should satisfy that: i) the resulting $|S_r^{(t)}|$ should be large enough so that the stragglers and dropouts have the minimal impact on round efficiency, and ii) $|S_r^{(t)}|$ should not be too large, otherwise local training on some devices may be futile because the cloud only accepts a maximum of $C \cdot N$ clients each round. The main challenge here is that *it is not permitted, for privacy reasons, for an edge node to probe the state of its clients* (including their IDs, aliveness, training progress, and the number of model updates made by a particular client), which causes the client's reliability (i.e., the probability that it drops out in a round) to be *unknown* to the edge and the cloud. In view of this, a probabilistic approach is developed in this work to determine $C_r^{(t)}$ for each region.

The following sections detail the design of HybridFL, in particular on how this work introduces regional slack factors into the model distribution stage and how the proposed algorithm performs the aggregations at the edge- and cloud-level.

4.4.2 Regional Participation Control

By specifying the regional selection proportion $C_r^{(t)}$ in round t, this work aims to involve a fraction of $C_r^{(t)} \cdot N_r$ clients in region r and expect that $C \cdot N_r$ of them do not drop/opt out, provided that all these clients behave independently and may be unreliable. Formally, the target of the proposed region-wise selection method can be formulated as:

$$\mathbb{E}[|X_r^{(t)}|; C_r^{(t)*}, N_r] = C \cdot N_r \tag{4.3}$$

where $\mathbb{E}[|X_r^{(t)}|; C_r^{(t)*}, N_r]$ is the expectation of the number of clients in region r that do not drop out in round t given that an optimal proportion of clients $C_r^{(t)} = C_r^{(t)*}$ are selected from N_r (i.e., total number of clients in region r) clients to perform local training.

Given any selection proportion $C_r^{(t)}$, the expectation at the left-hand side of Eq. (4.3) is equivalent to:

$$\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] = \sum_{k=0}^{C_r^{(t)} N_r} k \sum_{b \in \mathrm{comb}(S_r^{(t)}, k)} P(b) \tag{4.4}$$

where $S_r^{(t)}$ is the set of clients selected in region r, $comb(U, k)$ is the set of all combinations for selecting k elements from the set of U, and $P(b)$ is the probability that the combination b of end devices happen to be those who do not drop out in round t. Given a combination $b \in comb(S_r^{(t)}, k)$ and let $P_{r,i}^{(t)}$ denote the probability that device i of region r does not drop/opt out in round t, $P(b)$ can be calculated by Eq. (4.5):

$$P(b) = \prod_{i \in b} P_{r,i}^{(t)} \cdot \prod_{i \notin b} (1 - P_{r,i}^{(t)}) \tag{4.5}$$

Therefore, to obtain the optimal client selection proportion $C_r^{(t)*}$ one must solve it from Eq. (4.6) combining Eq. (4.5):

$$\sum_{k=0}^{C_r^{(t)*} N_r} k \sum_{b \in \mathrm{comb}(S_r^{(t)}, k)} P(b) = C \cdot N_r \tag{4.6}$$

However, $C_r^{(t)*}$ cannot be solved from Eq. (4.6) without a priori knowledge on the probability $P_{r,i}^{(t)}$ (i.e., reliability) of every individual client. In this work a FL scenario with strong privacy-preserving condition is considered, under which it is prohibited to acquire the clients' identifiers and their states [49], i.e., $P_{r,i}^{(t)}$ is unknown. In view of this, this work presents a novel approach to the problem and eventually works out the optimal value of $C_r^{(t)}$ for each region r each round t.

Assume that $\theta_r^{(t)}$ is such a probability that after each individual $P_{r,i}^{(t)}$ in Eq. (4.5) is replaced with $\theta_r^{(t)}$, the resulting expectation of $|X_r^{(t)}|$ remains unchanged. One can always find such $\theta_r^{(t)}$ because after the replacement, $|X_r^{(t)}|$ of region r follows the Binomial distribution $\mathcal{B}(C_r^{(t)} N_r, \theta_r^{(t)})$ and the expectation of $|X_r^{(t)}|$ (i.e., $\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] \in [0, C_r^{(t)} N_r]$) is a surjective function of $\theta_r^{(t)} \in [0, 1]$. Now the right-hand side of Eq. (4.4) can be rewritten as:

$$\begin{aligned} \mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] &= \sum_{k=0}^{C_r^{(t)} N_r} k \cdot P(|X_r^{(t)}| = k) \\ &= C_r^{(t)} N_r \theta_r^{(t)} \end{aligned} \tag{4.7}$$

where the second equality in Eq. (4.7) holds because $|X_r^{(t)}| \sim \mathcal{B}(C_r^{(t)} N_r, \theta_r^{(t)})$. Note that in this work, clients are assumed independent of each other regarding their behaviours.

Combining Eq. (4.7) and the selection target Eq. (4.3):

$$C_r^{(t)} = \frac{C}{\theta_r^{(t)}} \text{ if } C_r^{(t)} = C_r^{(t)*} \tag{4.8}$$

where C is the desired global proportion of clients (specified by the cloud) with successful model submissions in round t over the entire MEC system; $\theta_r^{(t)}$ defined in Eq. (4.7) rescales the selection proportion in a region to compensate the client drop-out in that region. Therefore in this work $\theta_r^{(t)}$ is termed the *regional slack factor* (for region r).

Note that $\theta_r^{(t)}$ in Eq. (4.8) cannot be decided arbitrarily, otherwise the optimality of $C_r^{(t)}$ is not guaranteed. This is because there is only one optimal value for $C_r^{(t)}$ given any distribution of client reliability and the target formulated in Eq. (4.3). In other words, if one determines $C_r^{(t)}$ via Eq. (4.8) provided an under-estimated $\theta_r^{(t)}$, the selection proportion $C_r^{(t)}$ will be too big (i.e., $C_r^{(t)} > C_r^{(t)*}$) for region r and consequently, the target expectation of $|X_r^{(t)}|$ will be higher than the desired level, i.e., $\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] > C \cdot N_r$. It is similar for the situation of over-estimated $\theta_r^{(t)}$.

According to Eq. (4.8), one can determine how many clients are needed in each region for an upcoming FL round after $\theta_r^{(t)}$ is resolved. In this work, a novel method is used to estimate $\theta_r^{(t)}$ based on the historical records of model submissions (since the course of FL is organised in rounds), i.e., how many models are collected by each region in previous rounds. Note that edge nodes can only count the models they collected but do not know which client submitted the model.

HybridFL adopts a quota-triggered aggregation mechanism in which the cloud ends a round once $N \cdot C$ client models have been submitted globally across the MEC system. As a result, the following holds:

$$\sum_{r \in V} |\Psi_r^{(t)}| = \min(N \cdot C, \sum_{r \in V} |X_r^{(t)}|) \tag{4.9}$$

where $\Psi_r^{(t)}$ is the set of clients that submit their models in time in round t (before the cloud ends a round after collecting $N \cdot C$ models globally) and V is the set of edge nodes. Note that $\Psi_r^{(t)} \subseteq X_r^{(t)}$ because when the cloud ends a round, some clients may be still working and have not finished local training. Details of how to determine the aggregation timing will be introduced later. Formally, a factor $q_r^{(t)*}$ is used to characterise the relation between $|\Psi_r^{(t)}|$ and $|X_r^{(t)}|$:

$$|\Psi_r^{(t)}| = |X_r^{(t)}| \cdot q_r^{(t)*} \tag{4.10}$$

where $q_r^{(t)*}$ denotes the percentage of clients in $X_r^{(t)}$ that submit local models in time (these clients make up $\Psi_r^{(t)}$). Note that $|\Psi_r^{(t)}|$ is observable as the number of local models collected by edge node r in round t. However, $X_r^{(t)}$ is unknown since this work considers a strong privacy scenario where the edge nodes are not allowed to probe the state of clients. One can only observe how many clients submitted the updated models (i.e., $|\Psi_r^{(t)}|$) but cannot know who have dropped out and who are still working. Therefore, one can transform Eq. (4.10) into Eq. (4.11) given $\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] \neq 0$, and then define $q_r^{(t)}$ in Eq. (4.12).

$$|\Psi_r^{(t)}| = \mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] \cdot \frac{|X_r^{(t)}| \cdot q_r^{(t)*}}{\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r]} \tag{4.11}$$

$$q_r^{(t)} \triangleq \frac{|X_r^{(t)}| \cdot q_r^{(t)*}}{\mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r]} \tag{4.12}$$

From Eqs. (4.11), (4.12) and (4.7), one can derive that:

$$\begin{aligned} |\Psi_r^{(t)}| &= \mathbb{E}[|X_r^{(t)}|; C_r^{(t)}, N_r] \cdot q_r^{(t)} \\ &= C_r^{(t)} N_r \theta_r^{(t)} \cdot q_r^{(t)} \end{aligned} \tag{4.13}$$

Note that the value of $\theta_r^{(t)}$ needs to be estimated before round t starts so that the selection proportion $C_r^{(t)}$ can be determined (as every round begins with the client selection step). However, Eq. (4.13) cannot be used directly to obtain $\theta_r^{(t)}$ because $\Psi_r^{(t)}$ and $q_r^{(t)}$ are unknown before round t is completed (Note that $\Psi_r^{(t)}$ is observable at the end of round t so $q_r^{(t)}$ can be calculated with $|\Psi_r^{(t)}|$ at round ends by Eq. (4.14) combining Eqs. (4.12), (4.10) and (4.3) with the assumption that $C_r^{(t)}$ is the optimal).

$$q_r^{(t)} = \frac{|\Psi_r^{(t)}|}{C \cdot N_r} \tag{4.14}$$

Therefore this work develops the following practical approach to work out $\theta_r^{(t)}$ by exploiting the historical records of the variables observable to edge nodes. More specifically, edge node r has stored $S_r^{(1)}, S_r^{(2)}, \ldots, S_r^{(t-1)}$, $q_r^{(1)}, q_r^{(2)}, \ldots, q_r^{(t-1)}$ and $C_r^{(1)}, C_r^{(2)}, \ldots, C_r^{(t-1)}$ at the start of round t. Also, according to the definition of $\theta_r^{(t)}$, it represents a region-wise property. Let us assume $\theta_r^{(t)}$ does not change significantly over the course of the FL training. Thus, a constant $\hat{\theta}_r^{(1..t)}$ is used as the approximation of $\theta_r^{(t)}$ within the time window spanning from round 1 to round t:

$$\theta_r^{(i)} \approx \hat{\theta}_r^{(1..t)}, \forall i \in \{1, 2, \ldots, t\} \tag{4.15}$$

Replacing $\theta_r^{(i)}$ with $\hat{\theta}_r^{(1..t)}$ in Eq. (4.13) and for round i, $\forall i < t$,

$$\frac{|S_r^{(i)}|}{N_r} \approx C_r^{(i)} q_r^{(i)} \hat{\theta}_r^{(1..t)}, \forall i \in \{1, 2, \ldots, t-1\} \tag{4.16}$$

Therefore, (4.16) is equivalent to a series of observations (the number of which is $t-1$) sampled from a function in the form of '$y = ax$' (i.e., $|S_r^{(i)}|/N_r$ and $C_r^{(i)} q_r^{(i)}$ being the y and x of sample i, respectively, and $\hat{\theta}_r^{(1..t)}$ is the coefficient). This work uses Least Square Estimation (LSE) to fit the value of $\hat{\theta}_r^{(1..t)}$ based on (4.16), which produces:

$$\hat{\theta}_r^{(1..t)} \overset{LSE}{=} \frac{1}{N_r} \frac{\sum_{i=1}^{t-1} C_r^{(i)} q_r^{(i)} |S_r^{(i)}|}{\sum_{i=1}^{t-1} \left(C_r^{(i)} q_r^{(i)}\right)^2}, \ t > 1. \tag{4.17}$$

where $S_r^{(i)}$, $C_r^{(i)}$ and $q_r^{(i)}, i = 1, 2, \ldots, t-1$ are retrieved from the logs on edge node r. At the start of round t, one can compute $\hat{\theta}_r^{(1..t)}$ and use it as an estimate of $\theta_r^{(t)}$, and then determine region r's client selection proportion $C_r^{(t)}$ (defined in Eq. (4.8)) using Eq. (4.18):

$$C_r^{(t)} = C \cdot N_r \frac{\sum_{i=1}^{t-1} \left(C_r^{(i)} q_r^{(i)}\right)^2}{\sum_{i=1}^{t-1} C_r^{(i)} q_r^{(i)} |S_r^{(i)}|}, \ t > 1. \tag{4.18}$$

For t=1 (the 1st round of FL), $\theta_r^{(t)}$ is initialised as a default value (e.g., $\theta_r^{(1)}$=0.5). $C_r^{(1)}$ is initialised to $C/\theta_r^{(1)}$ accordingly. To investigate the effectiveness of the proposed method in terms of achieving the selection target Eq. (4.3), 20 clients are simulated in two regions and ran 100 rounds (5 local epochs per round, sufficient for convergence) of federated learning using HybridFL as the control algorithm and $\theta_r^{(1)}$ is initialised to 0.5. The reliability of clients in training follows the Gaussian distribution $\mathcal{N}(\mu, 0.15^2)$ where $\mu = \mathbb{E}[P_{r,i}^{(t)}]$ is set to 0.43 and 0.57 for Region 1 and Region 2, respectively.

Clients also differ in performance which follows $\mathcal{N}(0.5, 0.1^2)$ (cycles/s); The global selection fraction C is set to 0.3.

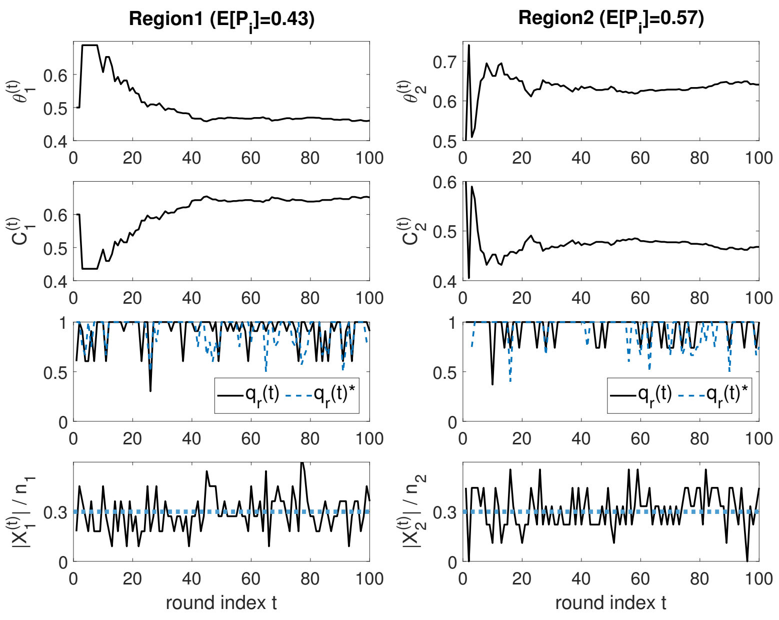


Figure 4.4: The traces of $\theta_r^{(t)}$, $C_r^{(t)}$, $q_r^{(t)}$ and $|X_r^{(t)}|/N_r$ in a simulation where there are 11 and 9 clients in Region 1 and Region 2, respectively.

From the traces of $\theta_r^{(t)}$, $C_r^{(t)}$, $q_r^{(t)}$ and $|X_r^{(t)}|/N_r$ in Fig. 4.4, one can observe that the proposed probabilistic estimation drives $\theta_r^{(t)}$ and $C_r^{(t)}$ (the first two rows in the figure) to the convergence at about 40 rounds of FL. Note that $\theta_1^{(t)}$ and $\theta_2^{(t)}$ converge to 0.46 and 0.63, which, by the definition, are not necessarily equal to $\mathbb{E}[P_{r,i}^{(t)}]$ (recall that $P_{r,i}^{(t)}$ is the reliability of client i in region r) which is set to 0.43 and 0.57 for Region 1 and Region 2 in this example, respectively. Besides, $q_r^{(t)}$ is defined without using any knowledge about $X_r^{(t)}$, but still produces a close approximation to its true value $q_r^{(t)*} = |\Psi_r^{(t)}|/|X_r^{(t)}|$ (the 3rd row in Fig. 4.4). Consequently, the client participating ratio in a region, quantified by $|X_r^{(t)}|/N_r$, is maintained around $C = 0.3$ (shown in the last row of Fig. 4.4; the blue dash line represents $C = 0.3$) after the convergence of $\theta_r^{(t)}$ and $C_r^{(t)}$.

This toy case demonstrates that the proposed method for estimating $\theta_r^{(t)}$ (which determines $C_r^{(t)}$) is both theoretically and practically feasible for finding the optimal value of the regional selection proportion that leads to the very expectation of $|X_r^{(t)}|$ desired by the cloud (see the target Eq. 4.3).

4.4.3 Two-level Model Aggregation

For coherence with the previous chapters, the notation $w^{(t)}(z)$ denotes the model in round t at time step z. Note that the time step z keeps counting since the very beginning of the first round.

In the proposed algorithm (HybridFL), model aggregation is a multi-step stage, involving both edge- and cloud-level aggregation (see steps 6, 7, 8 and 9 in fig. 4.3). In HybridFL, once the updated models submitted by the clients across the MEC system amount to $N \cdot C$ (i.e., $|\Psi^{(t)}| = \sum_{r \in V} |\Psi_r^{(t)}|$ reaches $N \cdot C$), it triggers the cloud to send the 'aggregation signal' to the edge nodes (see step 6, Fig. 4.3). *The edge nodes will then stop waiting for more local models.* This quota-triggered regional aggregation effectively mitigates the impact of the clients which have dropped out or are straggling. Consequently the round length is expected to be shortened (the experimental results presented later support this expectation).

The proposed method adopts an *immediate cloud aggregation* strategy, which allows the cloud-level model aggregation to be conducted right after the edge-level aggregation is completed. The rationale behind this strategy is that the cloud-edge network connection is typically reliable and of low latency. Therefore, it facilitates the global information exchange and the convergence of the global model by aggregating the regional models at the cloud level as early as possible after the regional aggregations are completed at the edge nodes. Fig. 4.5 demonstrates how rounds are orchestrated in HybridFL.

The cloud keeps monitoring the total number of clients that have submitted their models each round by listening to the reports of the current value of $|\Psi_r^{(t)}|$ from the edge nodes. Once the total number of client model submissions reaches the quota $N \cdot C$, the cloud will signify the edge nodes to perform regional aggregation, the result of which can be formulated as:

$$w_r^{(t)} = \sum_{k \in U_r} \frac{|D_{r,k}|}{|D_r|} w_{r,k}^{(t)} \tag{4.19}$$

where $w_{r,k}^{(t)}$ is the model on client k in region r and $w_r^{(t)}$ denotes the resulting regional model for edge node r in round t. Note that the aggregation involves all client models in the region, not limited to those who finished local training successfully (see fig. 4.5). To alleviate model staleness, a cache solution is used in which the local models without successful update in the current round are replaced with the existing regional model

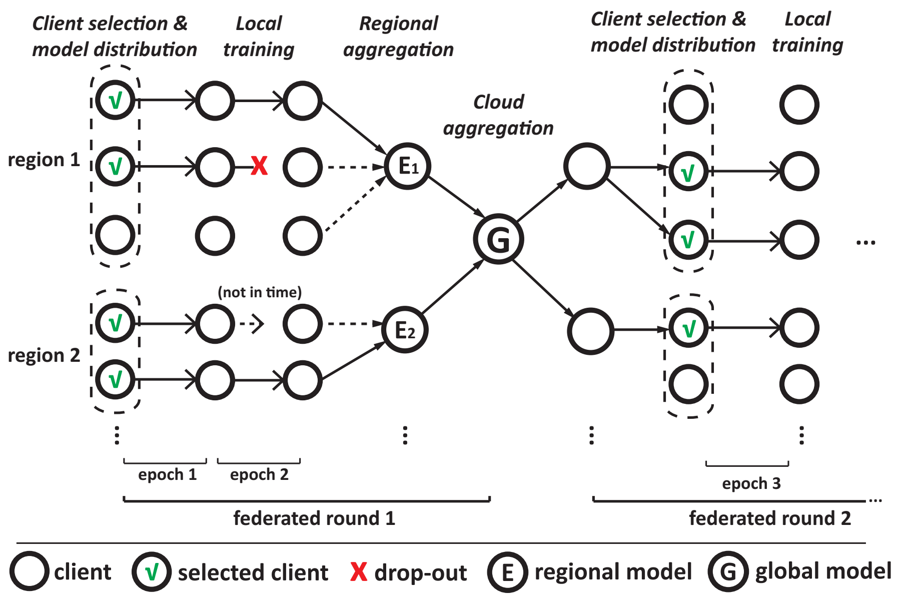


Figure 4.5: The workflow of the proposed HybridFL algorithm wherein the cloud requires two local model submissions (C=0.4 and n=5 in this example) each round to trigger the model aggregation.

obtained in last round before the aggregation is conducted, i.e., $w_{r,k}^{(t)} = w_r^{(t-1)}$ if $k \notin \Psi_r^{(t)}$.

The cloud aggregation will be performed immediately after the regional aggregation to produce the cloud model. Instead of using a constant weight for each regional model as in the literature [279], the proposed approach adopts a data-oriented weight averaging strategy by introducing the *Effective Data Coverage* (EDC) for each region in every round. EDC quantifies the actual size of data covered in round t's training based on $\Psi_r^{(t)}$. EDC for region r in round t (denoted by $EDC_r^{(t)}$) is formulated as:

$$EDC_r^{(t)} = \sum_{k \in \Psi_r^{(t)}} |D_{r,k}| \tag{4.20}$$

where $\Psi_r^{(t)}$ is the set of clients who submitted their models successfully to its regional edge node. Accordingly, this work further defines EDC for the whole MEC system (denoted by $EDC^{(t)}$) as:

$$EDC^{(t)} = \sum_{r \in V} EDC_r^{(t)}. \tag{4.21}$$

In the model aggregation step at the cloud level, the proposed aggregation method weights each regional model $w_r^{(t)}$ based on EDC to characterise its round-wise contribution in producing the global model $w^{(t)}$:

$$w^{(t)} = \sum_{r \in V} \frac{EDC_r^{(t)}}{EDC^{(t)}} w_r^{(t)}. \tag{4.22}$$

Algorithm 3 presents the pseudo-code of the entire process of FL using the proposed algorithm.

Algorithm 3: the HybridFL algorithm

```
Input  : maximum number of rounds $t_{max}$, local epochs per round $\tau$, desired
         proportion $C$, response time limit $T_{lim}$
Output : finalised global model $w$
// Cloud process: running on the central server
Initialise global model $w$
$quota \leftarrow C \cdot n$
for round $t \leftarrow 1$ to $t_{max}$ do
    Distribute $w$ to all the edge nodes
    for each edge node $r$ in $V$ in parallel do
        Compute $C_r^{(t)}$ according to Eq. (4.18)
        edgeUpdate($r, C_r^{(t)}, \tau$)
    end
    Keep monitoring update count by edge nodes
    if $|\Psi^{(t)}| \geq quota$ or $T_{lim}$ is reached then
        // trigger regional aggregation
        Send aggregation signal to all edge nodes
        edgeAggregation($r$)
    end
    // cloud aggregation
    Update $w$ according to Eq. (4.22)
end
return $w$
// Edge process: running on edge node $r$
edgeUpdate($r, C_r, \tau$):
    $q \leftarrow C_r \cdot N_r$
    $Q \leftarrow q$ randomly selected clients in region $r$
    for each client $k$ in $Q$ in parallel do
        clientUpdate($k, r, \tau$)
        Keep reporting update count to the cloud
    end
return
edgeAggregation($r$):
    Compute $w_r$ according to Eq. (4.19)
return
// Client process: running on client $k$
clientUpdate($k, r, \tau$):
    for step $e \leftarrow 1$ to $\tau$ do
        Update $w_{r,k}$ using Gradient Descent method
    end
return
```

4.4.4 Convergence Analysis

The convergence of the global model by federated learning has been proven for both the two-layer architecture [55] and the three-layer edge computing systems [279]. However, since modification is made to the aggregation rules, it is necessary to provide the convergence analysis with the focus on showing the difference compared to the proof provided in the existing work.

Since the regional aggregation in HybridFL is followed instantly by the cloud (global) aggregation, one can mathematically re-formulate the global model $w^{(t)}$ by combining Eq. (4.19) and Eq. (4.22), which yields Eq. (4.23):

$$\begin{aligned} w^{(t)} &= \sum_{r \in V} \frac{EDC_r^{(t)}}{EDC^{(t)}} \sum_{k \in U_r} \frac{|D_{r,k}|}{|D_r|} w_{r,k}^{(t)} \\ &= \sum_{k \in U} \underbrace{\frac{EDC_{r(k)}(t)}{EDC^{(t)}} \frac{|D_{r(k),k}|}{|D^{r(k)}|}}_{\gamma(k,r(k),t)} w_{r(k),k}^{(t)} \end{aligned} \tag{4.23}$$

where $r(k)$ stands for the corresponding edge node connected to client k and the symbol $\gamma(k, r(k), t)$ represents the weight of client k's model during the aggregation. Without ambiguity, in the following content, the abbreviation $\gamma(k, r, t)$ is used to denote $\gamma(k, r(k), t)$ for brevity.

In Eq. (4.23), the first equality represents the two-level aggregation, namely, the second '$\sum$' represents the edge-level aggregation while the first '$\sum$' represents the cloud-level aggregation. The second equality in Eq. (4.23) transforms the two '$\sum$' into one. This suggests that the entire aggregation in the proposed three-layer MEC system is equivalent to the two-layer FL process as shown in [47] with the only difference lying in weights.

The following content analyses the convergence of the proposed algorithm by quantifying the upper bound of $F(w^{(t)}, t) - F(w^*, t)$, where w^* denotes the optimal model parameters for the optimisation target (4.1). Due to space limit, the proof is presented based on the analysis provided by Wang et al. [55] and extend their *Theorems 1 and 2* for the case of the proposed algorithm, which yields the **Theorem 4.1** and **Theorem 4.2** in this work, respectively. The following assumption is made to facilitate the analysis:

Assumption 4.1 (Loss function)**.** *$F_k(w, t)$ is convex, ρ-Lipschitz and β-smooth.*

For the loss functions that do not satisfy the assumption above, Wang et al. [55] still validated the effectiveness of FL in such cases. The assumption implies that $F(w, t)$ *is convex, ρ-Lipschitz and β-smooth* with regards to w for any round t, which can be proven using the triangle inequality based on Eq. (4.1). This work also defines δ_k as the upper bound of the divergence between the gradients of $F_k(w)$ and $F(w, t)$, and $\bar{\delta}$ as the upper limit of $\delta_k, \forall k \in U$:

$$\|\nabla F_k(w) - \nabla F(w, t)\| \leq \delta_k \leq \bar{\delta}. \tag{4.24}$$

The following analysis is provided at the level of local update steps. Let z denote the index of local update steps and the relation between a round index t and the update time steps in the round follows that $z \in ((t-1)\tau, t\tau]$. To facilitate the analysis, let $w(z)$ be a virtual global model as the result of aggregating all $w_{r,k}(z)$ after each update. It is not to be confused with $w^{(t)}$ because $w^{(t)}$ is only visible after the aggregation at the end of a round. Besides, this work also considers an auxiliary model $v^{(t)}(z)$ learnt using *centralised* gradient descent initialised as $w^{(t-1)}$ in the context of round t for optimising the same target $F(w, t)$. Given z as an update step in round t, $w(z)$ is defined as:

$$w(z) = \sum_{k \in U} \gamma(k, r, t) w_{r,k}(z) \tag{4.25}$$

where $w_{r,k}(z)$ is updated from $w_{r,k}(z-1)$:

$$w_{r,k}(z) = w_{r,k}(z-1) - \eta \nabla F_k(w_{r,k}(z-1)) \tag{4.26}$$

For $v^{(t)}(z)$ with $z \in ((t-1)\tau, t\tau]$, it follows that

$$v^{(t)}(z) = v^{(t)}(z-1) - \eta \nabla F(v^{(t)}(z-1), t) \tag{4.27}$$

The proposed Theorem 4.1 is presented as follows:

Theorem 4.1 (Loss divergence bound)**.** *for any step z in round t,*

$$F(w(z), t) - F(v^{(t)}(z), t) \leq \rho \bar{h}(z - (t-1)\tau) \tag{4.28}$$

where

$$\bar{h}(x) \triangleq \frac{\bar{\delta}}{\beta}((\eta\beta + 1)^x - 1) - \eta\bar{\delta}x \tag{4.29}$$

Proof. This work bases the proof of Theorem 4.1 on [Lemma 2, [55]]. First let $\delta(t)$ denote the weighted average of δ_k by $\gamma(k, r, t)$, which will be used later in the proof:

$$\begin{aligned} \delta(t) &\triangleq \sum_{k \in U} \gamma(k, r, t)\delta_k \\ &\leq \sum_{k \in U} \gamma(k, r, t)\bar{\delta} \\ &= \bar{\delta} \\ &\quad (\text{because} \sum_{k \in U} \gamma(k, r, t) = 1) \end{aligned} \tag{4.30}$$

where $\gamma(k, r, t)$ is the abbreviation of $\gamma(k, r(k), t)$ defined in Eq. (4.23) and δ_k is defined in (4.24).

Combining Eq. (4.25) and Eq. (4.26),

$$w(z) - w(z-1) = -\eta \sum_{k \in U} \gamma(k, r, t) \nabla F_k(w_{r,k}(z-1)) \tag{4.31}$$

From Eq. (4.27) and Eq. (4.31) given any $z \in ((t-1)\tau, t\tau]$, one can derive that:

$$\begin{aligned}
&\|w(z) - v^{(t)}(z)\| \\
&= \|w(z-1) - \eta \sum_{k \in U} \gamma(k,r,t) \nabla F_k(w_{r,k}(z-1)) \\
&\quad - v^{(t)}(z-1) + \eta \nabla F(v^{(t)}(z-1), t)\| \\
&= \|w(z-1) - \eta \sum_{k \in U} \gamma(k,r,t) \nabla F_k(w_{r,k}(z-1)) \\
&\quad - v^{(t)}(z-1) + \eta \sum_{k \in U} \gamma(k,r,t) \nabla F_k(v^{(t)}(z-1))\| \\
&= \|w(z-1) - v^{(t)}(z-1) \\
&\quad - \eta \sum_{k \in U} \gamma(k,r,t) \big(\nabla F_k(w_{r,k}(z-1)) - \nabla F_k(v^{(t)}(z-1))\big)\| \\
&\le \|w(z-1) - v^{(t)}(z-1)\| \\
&\quad + \eta \sum_{k \in U} \gamma(k,r,t) \|\nabla F_k(w_{r,k}(z-1)) - \nabla F_k(v^{(t)}(z-1))\| \\
&\qquad \text{(from triangle inequality).}
\end{aligned} \tag{4.32}$$

Using the property of the local objective $F_k(\cdot)$, it follows that

$$\begin{aligned}
&\|w(z) - v^{(t)}(z)\| \\
&\le \|w(z-1) - v^{(t)}(z-1)\| \\
&\quad + \eta\beta \sum_{k \in U} \gamma(k,r,t) \|w_{r,k}(z-1) - v^{(t)}(z-1)\| \\
&\qquad \text{(because } F_k(\cdot) \text{ is } \beta\text{-smooth)} \\
&\le \|w(z-1) - v^{(t)}(z-1)\| \\
&\quad + \eta\beta \sum_{k \in U} \gamma(k,r,t) \frac{\delta_k}{\beta} \big((\eta\beta+1)^{z-1-(t-1)\tau} - 1\big) \\
&\qquad \text{(from [Lemma 2, [55])} \\
&= \|w(z-1) - v^{(t)}(z-1)\| \\
&\quad + \eta\delta(t) \big((\eta\beta+1)^{z-1-(t-1)\tau} - 1\big) \\
&\le \|w(z-1) - v^{(t)}(z-1)\| \\
&\quad + \eta\overline{\delta} \big((\eta\beta+1)^{z-1-(t-1)\tau} - 1\big) \\
&\qquad \text{(from the definition of } \delta(t) \text{ in (4.30))}
\end{aligned} \tag{4.33}$$

Equivalently,

$$\begin{aligned}
&\|w(z) - v^{(t)}(z)\| - \|w(z-1) - v^{(t)}(z-1)\| \\
&\leq \eta\bar{\delta}\big((\eta\beta+1)^{z-1-(t-1)\tau} - 1\big) \qquad (4.34)
\end{aligned}$$

Since $w(z) = v^{(t)}(z)$ when $z = (t-1)\tau$ according to the definition of the auxiliary model $v^{(t)}(z)$, it is true that $\|w(z) - v^{(t)}(z)\| = 0$ at $z = (t-1)\tau$. By summing up (4.34) over $z \in ((t-1)\tau, t\tau]$ (i.e., steps in round t), one can derive that:

$$\begin{aligned}
&\|w(z) - v^{(t)}(z)\| \\
&= \sum_{i=(t-1)\tau+1}^{z} \|w(i) - v^{(t)}(i)\| - \|w(i-1) - v^{(t)}(i-1)\| \\
&\leq \eta\bar{\delta} \sum_{i=(t-1)\tau+1}^{z} \big((\eta\beta+1)^{i-1-(t-1)\tau} - 1\big) \\
&= \eta\bar{\delta} \sum_{j=1}^{z-(t-1)\tau} \big((\eta\beta+1)^{j-1} - 1\big) \\
&\quad (\text{let } j = i-(t-1)\tau) \\
&= \eta\bar{\delta}\frac{(1-(\eta\beta+1)^{z-(t-1)\tau})}{-\eta\beta} - \eta\bar{\delta}(z-(t-1)\tau) \\
&= \frac{\bar{\delta}}{\beta}\big((\eta\beta+1)^{z-(t-1)\tau} - 1\big) - \eta\bar{\delta}(z-(t-1)\tau) \\
&= \bar{h}(z-(t-1)\tau) \qquad (4.35)
\end{aligned}$$

Recall that the target loss function $F(w,t)$ is ρ-Lipschitz (with regard to w) as a corollary from Assumption 4.1. Using the result above one can further derive that:

$$\begin{aligned}
F(w(z),t) - F(v^{(t)}(z),t) &\leq \|F(w(z)) - F(v^{(t)}(z))\| \\
&\leq \rho\|w(z) - v^{(t)}(z)\| \\
&\leq \rho\bar{h}(z-(t-1)\tau) \qquad (4.36)
\end{aligned}$$

□

Theorem 4.1 gives the theoretical difference in loss between the global model $w(z)$ (by aggregating local models) and the baseline $v^{(t)}(z)$ (learnt on centralised data) during the training process in round t. Note that $F(w(z), t) - F(v^{(t)}(z), t) \leq \rho\bar{h}(\tau)$ at $z = t\tau$ and $\bar{h}(1) = 0$. This means that $w(z)$ is equivalent to $v^{(t)}(z)$ if the aggregation interval $\tau = 1$. Based on Theorem 4.1 and recalling that $w^{(t)} = w(t{\cdot}\tau)$, the convergence upper bound of $w^{(t)}$ in Theorem 4.2 is given by:

Theorem 4.2 (Convergence upper bound)**.** *After t rounds with τ update steps in each round, the convergence of the global model is guaranteed by:*

$$F(w^{(t)}, t) - F(w^*, t) \leq \frac{1}{t\tau\big(\omega\eta(1 - \frac{\beta\eta}{2}) - \frac{\rho\bar{h}(\tau)}{\tau\epsilon^2}\big)} \tag{4.37}$$

when the conditions below are satisfied:
1) $\eta \leq \frac{1}{\beta}$
2) $\omega\eta(1 - \frac{\beta\eta}{2}) - \frac{\rho\bar{h}(\tau)}{\tau\epsilon^2} > 0$
3) $F(v^{(t)}(z), t) - F(w^*, t) \geq \epsilon$, $\forall z \in ((t-1)\tau, t\tau]$
4) $F(w^{(t)}, t) - F(w^*, t) \geq \epsilon$
where $\epsilon > 0$, $\omega \triangleq \min_t \frac{1}{\|v^{(t)}((t-1)\tau) - w^*\|^2}$, *and* $\bar{h}(\cdot)$ *is defined in Eq. (4.29).*

Condition 1 places a limit on the learning rate η whilst condition 2 guarantees that the upper bound is a positive value. Conditions 3 and 4 limit the lower bound of the gap to a positive value ϵ because $w^{(t)}$ is an approximation of w^* given that the proposed algorithm performs the aggregation after every τ local epochs and $\tau > 1$. Theorem 4.2 can be proved based on the conclusion of Theorem 4.1 combined with the [Lemmas 1, 3 and 4, [55]], and the steps of proof are the same as that provided in [55].

Theorem 4.2 implies that the gap between $w^{(t)}$ and w^* (the optimum) in terms of optimising the target loss function narrows as the FL process proceeds, i.e., t increases.

4.5 Experiments

The effectiveness of the proposed HybridFL was evaluated in terms of model convergence speed, round efficiency and the global model's accuracy. Experiment results also include the energy consumption of end devices, which is often considered as an important metric in practice.

4.5.1 Experimental Setup

For evaluation at scale, a simulated MEC system was built for federated learning using the same Python-based programming environment introduced in Section 3.7.1, Chapter 3. The MEC system was established with three types of parties (i.e., the cloud, edge nodes and end devices) that comprise the three-layer architecture. On-device training in the FL process was implemented using the PyTorch framework. Each group of end devices (clients) are managed by and connected to an edge node via wireless channels, which forms a region, whilst the edge nodes and the cloud are connected through high-speed Ethernet. Actual model exchange was implemented as in-memory tensor exchange with a server-side wall clock for simulating time consumptions. All the clients and their local network are implemented as being unreliable in the simulated MEC system. The drop-out probability of client k is set as dr_k, which follows a Gaussian distribution (see Table 4.2) with its mean value set to $\mathbb{E}[dr]$.

HybridFL was evaluated on two machine learning tasks: *Aerofoil* (Task 1) and *MNIST* (Task 2). Different MEC environments were configured to test the performance of HybridFL with different scale of end devices and edge nodes with different data distribution. The size of data partitions in each end device in Task 1 follows the Gaussian distribution while in Task 2, it is set to be non-IID by assigning the samples of class y_i, with a probability of 0.75, to the clients with indices $k \equiv y_i \pmod{10}$.

The evaluation also includes two existing algorithms in recent literature for comparison: FedAvg [47] and HierFAVG [279]. FedAvg is the vanilla FL algorithm proposed by Google for the two-layer client/server architecture. HierFAVG is a three-layer FL algorithm for Edge Computing systems and adopts a similar training architecture as the proposed algorithm by introducing the edge layer that performs the edge-level model aggregation before the global aggregation conducted by the cloud. HierFAVG has no adaptive control over the flow of models. Both the edge and the cloud have to await the responses from all the selected clients. The FL training process driven by these algorithms and HybridFL are compared under the same settings. The parameters in the experimental setting are listed in Table 4.2.

In this work, a cycle from the stage of model distribution, local training to model aggregation is called a *federated round.* Note that the global aggregation is performed every federated round by FedAvg and the proposed HybridFL, but HierFAVG performs it after several times of edge-level aggregation (i.e., it runs multiple federated

Table 4.2: Experimental setup for federated learning and the MEC system. The units of performance, bandwidth and throughput in the table are GHz, MHz, and Mbps, respectively.

Setting	Symbol	Task 1	Task 2
Dataset	D	Aerofoil	MNIST
Number of features	d	5	28×28
Model	w	FFN	LeNet-5
Dataset size	$\|D\|$	1503	70k
Number of clients	N	15	500
Number of edge nodes	M	3	10
Data distribution	-	$\mathcal{N}(100, 30^2)$	non-IID, 0.75
Client performance	s_k	$\mathcal{N}(0.5, 0.1^2)$	$\mathcal{N}(1.0, 0.3^2)$
Client bandwidth	bw_k	$\mathcal{N}(0.5, 0.1^2)$	$\mathcal{N}(1.0, 0.3^2)$
Signal-noise ratio	SNR	1e2	1e2
Drop-out probability	dr_k	$\mathcal{N}(\mathbb{E}[dr], 0.05^2)$	$\mathcal{N}(\mathbb{E}[dr], 0.05^2)$
Region population	N_r	$\mathcal{N}(5.0, 1.5^2)$	$\mathcal{N}(50, 15^2)$
Cloud–edge throughput	BR	1e3	1e3
Max number of rounds	t_{max}	600	400
Bits per sample	BPS	6*8*8	28*28*1*8
Cycles per bit	CPB	300	400
Number of local epochs	τ	5	5
Loss function	f	MSE Loss	NLL Loss
Learning rate	η	1e-4	1e-3

rounds before a cloud aggregation). The cloud-level aggregation interval (κ_2 in [279]) for HierFAVG is set to 10, which is shown to be an optimal setting in their work.

For fair comparison, all the algorithms were run for the same number of (federated) rounds, denoted by t_{max}, which also means the same number of total updates because each client runs the same number of local updates, denoted by τ, before the edge nodes conduct an edge-level aggregation.

The length of a federated round, denoted by T_{round}, can be formulated as:

$$T_{round} = T_{c2e2c} + \min\left\{T_{lim}, \max_{k \in S'}\{T_k^{comm} + T_k^{train}\}\right\} \tag{4.38}$$

where T_{lim} is the preset limit of response time, which is configured as the time required by an extremely straggling client to finish its local training and communication with an average partition size. Given the performance (denoted by s_k) and bandwidth (denoted by bw_k) of the clients follow the normal distribution with the mean and standard deviation being μ and σ, respectively, the performance and bandwidth of such an extremely straggling client is set to be $\mu - 3\sigma$.

Note that S' represents the selected fraction of clients for FedAvg and HierFAVG, but for HybridFL $S' \equiv \Psi^{(t)}$ because of the proposed quota-triggered aggregation mechanism. T_{c2e2c} is the cloud–edge communication time, which is calculated by

(4.39). T_k^{comm} and T_k^{train} are the communication time and local training time of client k, which are calculated by (4.40) and (4.41), respectively.

$$T_{c2e2c} = 3 \times \frac{msize \cdot M}{BR} \tag{4.39}$$

where BR is the bit rate of the cloud–edge connection. The multiplier '3' exists because the model upload typically spends twice as much time as the model download, given that uplink bandwidth is typically 50% of the total. The size of the model ($msize$) is set to 5 MB and 10 MB for Tasks 1 and 2, respectively. For FedAvg, $T_{c2e2c} \equiv 0$ because it does not involve the edge layer. For the edge–client wireless network, the effective bit rate is calculated by applying the Shannon theorem to the corresponding bandwidth bw_k,

$$T_k^{comm} = 3 \times T_k^{download} = 3 \times \frac{msize}{bw_k \cdot \log(1 + SNR)} \tag{4.40}$$

$$T_k^{train} = \frac{|D_{r,k}| \cdot \tau \cdot BPS \cdot CPB}{s_k} \tag{4.41}$$

The numerator in (4.41) quantifies the total number of CPU cycles needed for training the local partition $D_{r,k}$.

Based on (4.40) and (4.41) this work further models the energy consumed by end device for local training:

$$\begin{aligned} E_k &= E_k^{comm} + E_k^{train} \\ &= P_{trans} \cdot T_k^{comm} + P_{comp}^{base} s_k^3 \cdot T_k^{train} \end{aligned} \tag{4.42}$$

where P_{trans} is the power consumption of transmitter and $P_{comp}^{base} s_k^3$ represents the power for on-device computation based on the frequency power model [280]. P_{trans} and P_{comp}^{base} are set to 0.5 and 0.7 Watt respectively based on the benchmarking results reported in [281].

4.5.2 Evaluation Results

The FL process was run in two ways: i) stop the process at a preset maximum round t_{max}, and ii) stop when a preset accuracy is achieved for the global model. In

Table 4.3 and Table 4.4, the results for task 1 and task 2 are presented respectively, in terms of best model accuracy achieved, average round length (obtained when stopping at t_{max}), the number of rounds needed and the total time duration (for achieving the desired model accuracy). This work also investigates the model convergence by comparing the accuracy traces (Figs. 4.6 and 4.8) for FedAvg, HierFAVG and HybridFL. Figs. 4.7 and 4.9 show the average energy consumption by end devices.

Task 1: Aerofoil

Aerofoil is a numerical regression task. FL was performed to learn a global Feed-Forward Network (FFN) model from a small group of clients that possess private aerofoil self-noise data[1]. Clients hold different partitions of the data without overlapping and cannot share the data with each other. This task simulates an industrial scenario where the production data are proprietary. The size of local partitions follow the Gaussian distribution specified in Table 4.2.

Table 4.3: Experimental results with Task 1: Aerofoil under different environmental settings of $\mathbb{E}[dr]$ and client selection proportion C.

		Stop @t_{max}					
		Best Accuracy			Round length (sec)		
C		0.1	0.3	0.5	0.1	0.3	0.5
$\mathbb{E}[dr] = 0.1$	FedAvg	0.727	0.727	0.727	52.42	73.12	80.73
	HierFAVG	0.727	0.726	0.728	51.56	71.90	81.08
	HybridFL	**0.729**	0.727	0.728	**37.80**	**63.80**	**58.15**
$\mathbb{E}[dr] = 0.3$	FedAvg	0.728	0.727	0.727	64.21	83.64	87.90
	HierFAVG	0.727	0.728	0.727	66.74	83.24	88.14
	HybridFL	**0.729**	0.728	**0.728**	**38.94**	**64.83**	**69.84**
$\mathbb{E}[dr] = 0.6$	FedAvg	0.711	0.727	0.728	83.54	89.78	90.39
	HierFAVG	0.714	0.727	0.728	81.43	89.91	90.44
	HybridFL	**0.727**	**0.728**	0.728	**65.38**	**73.23**	**84.96**

		Stop @Acc=0.70					
		Rounds needed			Total time (sec)		
C		0.1	0.3	0.5	0.1	0.3	0.5
$\mathbb{E}[dr] = 0.1$	FedAvg	238	95	56	12515.1	5585.2	4149.7
	HierFAVG	250	80	50	14608.6	4765.8	4285.8
	HybridFL	**113**	**75**	**49**	**4254.6**	**3341.4**	**3143.0**
$\mathbb{E}[dr] = 0.3$	FedAvg	376	125	74	25442.2	10674.6	6588.8
	HierFAVG	340	130	60	22237.5	12025.9	6132.9
	HybridFL	**141**	**77**	**51**	**7010.7**	**4994.9**	**3711.6**
$\mathbb{E}[dr] = 0.6$	FedAvg	598	233	144	50122.4	21141.3	13108.4
	HierFAVG	590	230	140	48922.2	21623.3	13565.6
	HybridFL	**160**	**66**	**62**	**10584.1**	**4780.4**	**5488.3**

[1]Airfoil Self-Noise Data Set, UCI. `https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise`

600 FL rounds were run to compare the best accuracy achieved and the average length of a round. The results are shown in the 'Stop @t_{max}' column of Table 4.3. We can see that the proposed algorithm effectively shortens the average round length by 6% to 42% with slight improvements on the global model's accuracy in most cases. In Fig. 4.6, we plot the trace of model accuracy over the FL process under the settings of $C \in \{0.1, 0.3, 0.5\}$ and $\mathbb{E}[dr] \in \{0.3, 0.6\}$ ($\mathbb{E}[dr] = 0.1$ is not shown in the plots due to space limit). From the figure one can observe a solid improvement in model convergence by HybridFL, especially under unstable MEC circumstances where end devices drop out frequently. In the setting of $\mathbb{E}[dr]$=0.6 and the selection proportion C=0.1 (Fig. 4.6(b)), the global model can hardly converge in 600 rounds using FedAvg or HierFAVG, but reached its optimum in 200 rounds under the control of the proposed HybridFL algorithm.

The algorithms were also evaluated by specifying a target model accuracy as the stop criterion to observe the number of rounds needed for convergence and the total time duration. The results are shown in the 'Stop @Acc' column of Table 4.3. HybridFL requires much fewer rounds and less time to achieve the accuracy target in Task 1, which yields up to 4× speedup compared to FedAvg and HierFAVG. In the setting where the clients are mostly unreliable (i.e., $\mathbb{E}[dr]$=0.6), HybridFL can still achieve very fast convergence, requiring only about 1/3 of the rounds needed by HierFAVG. Another benefit of fast convergence is energy conservation. Fig. 4.7 shows the energy consumption of end devices. It can be seen that the proposed algorithm is most energy consumption friendly to end devices. HybridFL reduces the average energy usage of end devices by roughly 50% for Task 1 in the case of $\mathbb{E}[dr] = 0.6$ and $C = 0.1$.

Task 2: MNIST

For this task the aim is to simulate a scenario in which the image samples are distributed over a relatively large fleet of end devices and they are not shared among the devices or allowed to be uploaded to the servers. This is a realistic scenario for mobile applications that are restricted by the privacy terms. In this experiment, 500 clients and 10 edge nodes were set up for running this task. Besides, to emulate the statistical heterogeneity of data (i.e., the bias in the data distribution over devices), samples are assigned to clients by matching data labels with clients' indices — sample (X_i, Y_i) has a 75% chance to reside on (one of) the clients whose IDs are congruent to

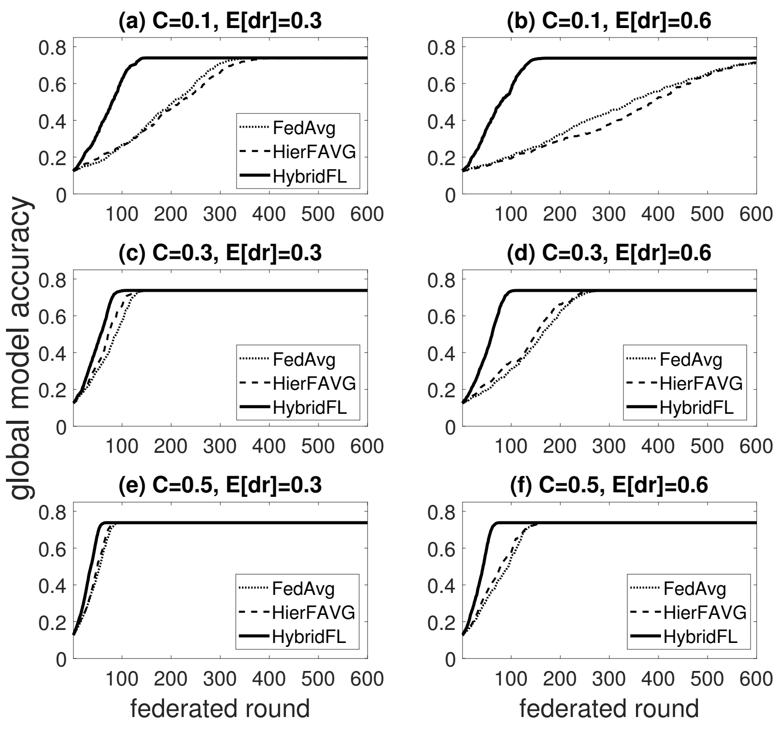


Figure 4.6: The trace of model accuracy during the FL process in Task 1: Aerofoil with C =0.1, 0.3 and 0.5. The cloud always keeps the best global model throughout the process. The drop-out probability of end devices follows the Gaussian distribution $\mathcal{N}(\mathbb{E}[dr], 0.05^2)$.

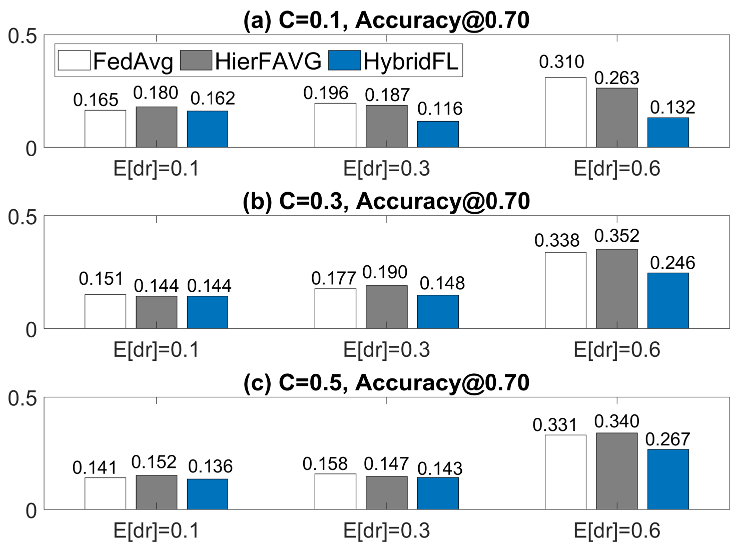


Figure 4.7: Comparing the energy consumption (Watt hours) of end devices when running Task 1 (Aerofoil) among FedAvg, HierFAVG and HybridFL. The values obtained are the average over all the end devices in the MEC system.

y_i modulo 10 (the MNIST data set has 10 classes). This way the data distribution on each device is far from being IID.

Table 4.4: Experimental results with Task 2: MNIST under different environmental settings of $\mathbb{E}[dr]$ and client selection proportion C.

		Stop @t_{max}					
		Best Accuracy			Round length (sec)		
C		0.1	0.3	0.5	0.1	0.3	0.5
$\mathbb{E}[dr] = 0.1$	FedAvg	0.936	0.958	0.962	377.28	378.02	378.02
	HierFAVG	0.936	0.958	0.964	377.65	378.26	378.26
	HybridFL	**0.940**	**0.959**	**0.965**	**63.59**	**96.55**	**140.51**
$\mathbb{E}[dr] = 0.3$	FedAvg	0.925	0.951	0.962	378.02	378.02	378.02
	HierFAVG	0.926	0.954	0.962	378.10	378.26	378.26
	HybridFL	**0.940**	**0.959**	**0.966**	**109.72**	**135.96**	**113.20**
$\mathbb{E}[dr] = 0.6$	FedAvg	0.901	0.933	0.950	378.02	378.02	**378.02**
	HierFAVG	0.905	0.941	0.952	378.10	378.26	378.26
	HybridFL	**0.937**	**0.960**	**0.963**	**37.59**	**126.15**	380.42

		Stop @Acc=0.90					
		Rounds needed			Total time (sec)		
C		0.1	0.3	0.5	0.1	0.3	0.5
$\mathbb{E}[dr] = 0.1$	FedAvg	200	66	32	75166.3	25327.3	12474.6
	HierFAVG	150	40	30	60519.8	18911.3	15128.8
	HybridFL	**124**	**41**	**26**	**7856.9**	**3762.9**	**3730.6**
$\mathbb{E}[dr] = 0.3$	FedAvg	230	77	37	87322.3	29485.5	14364.7
	HierFAVG	200	60	30	79432.7	26476.5	15128.8
	HybridFL	**123**	**41**	**25**	**15978.4**	**7148.9**	**4867.4**
$\mathbb{E}[dr] = 0.6$	FedAvg	376	146	65	142513.1	55568.8	24949.2
	HierFAVG	350	100	60	136171.6	41606.9	26476.5
	HybridFL	**118**	**41**	**31**	**11743.1**	**7334.8**	**12171.8**

The lightweight convolutional neural net LeNet-5 (consisting of two convolutional layers with max pooling and three fully connected layers) is used as the model for this image classification task. Again the FL process was run for a fixed number of rounds first to observe the best accuracy and round length. The results are shown in the 'Stop @t_{max}' column in Table 4.4, from which one can see that HybridFL outperformed FedAvg and HierFAVG in terms of the accuracy of the global model in all cases, especially when the participating devices are generally unreliable ($\mathbb{E}[dr] = 0.6$). Fig. 4.8 tracks the accuracy of the global model in the training process under the settings of C= 0.1, 0.3 and 0.5 and $\mathbb{E}[dr]$= 0.3 and 0.6 ($\mathbb{E}[dr] = 0.1$ is not shown in the plots due to space limit). It can be observed that the convergence of the global model is improved when using HybridFL as the controlling algorithm. These results suggest that compared with FedAvg and HierFAVG, HybridFL can achieve the best global model in the fewest number of federated rounds for Task 2.

The performance of HybridFL is also compared with the baseline algorithms by

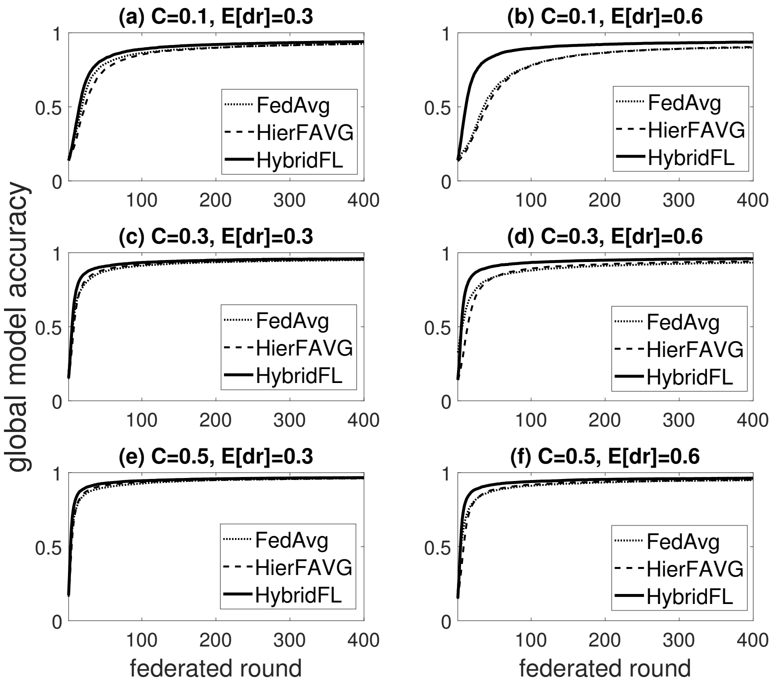


Figure 4.8: The trace of accuracy during the FL process in Task 2: MNIST with $C = 0.1, 0.3$ and 0.5. The cloud always keeps the best global model throughout the process. The drop-out probability of end devices follows the Gaussian distribution $\mathcal{N}(\mathbb{E}[dr], 0.05^2)$.

specifying acc= 0.9 as the convergence target for the global model. The results are listed in the lower part of Table 4.4. One can observe from the table that HybridFL significantly reduces the number of rounds and total time needed to achieve the accuracy target, compared with other two algorithms. For example, HybridFL achieves a roughly 12× speedup in the case where $\mathbb{E}[dr]$=0.6 and C=0.1, which represents a situation where the clients may drop out frequently and the participating fraction is restricted.

Some interesting results were observed in the experiments. In both tasks 1 and 2 with $\mathbb{E}[dr] = 0.6$, the proposed algorithm requires fewer rounds to converge given $C = 0.5$ than that with $C = 0.3$, but the total time consumption for $C = 0.5$ is longer (see Tables 4.3 and 4.4). This is because the extremely high drop-out probability (0.6 on average in the cases) of clients makes it almost impossible to engage 50% of them (given $C = 0.5$) in training, even with the modulation of the regional slack factors. This is the case $|\Psi^{(t)}| < N \cdot C$. In such a case, the edge nodes and the cloud have to wait until the preset round length limit is reached, and thus the round length is prolonged. To some extent, this observation explains why it is suggested in literature

[49] that the selection proportion C should not be set too large.

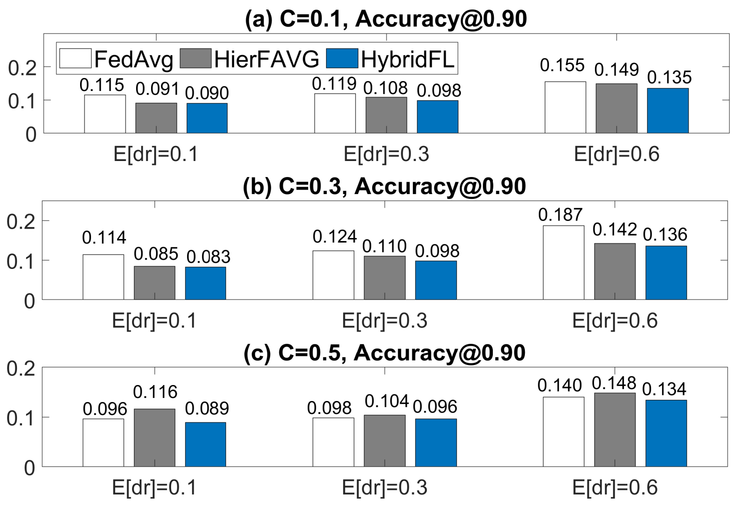


Figure 4.9: Comparing the average on-device energy (Watt hours) consumed for Task 2: MNIST in local training when using FedAvg, HierFAVG and HybridFL as the control algorithms. The values obtained are the average over all the end devices in the MEC system.

Device energy usage can be a key factor that affects the willingness of device owners to participate in the FL training. Fig. 4.9 shows the average energy consumption of an end device as a participant in the FL process to achieve the preset accuracy target 0.9 for Task 2. The advantage of HybridFL in energy saving in Task 2 is not as prominent as in Task 1. Yet HybridFL still managed to retain the on-device energy usage at the lowest level. This is because it enables much faster convergence (therefore less total training time for devices) than the baseline algorithms. In practice, the energy-saving feature of HybridFL can help attract more end devices in each round.

The evaluation of the proposed algorithm (HybridFL) with the two machine learning tasks under different environment settings demonstrates its effectiveness in terms of boosting the efficiency of FL, improving the global model's quality and saving on-device energy consumption in a three-layer MEC system. The reasons behind these improvements are three-fold. First, the quota-triggered regional aggregation in HybridFL effectively prevents the situation where some regions with extremely unreliable clients slow down the entire FL process. Second, the proposed algorithm enables each edge node to rescale its regional quota based on its slack factor to improve the robustness of FL against client drop-out. Third, the cloud (i.e., global) aggregation is designed to be performed immediately after regional aggregation so that the global exchange of the model is made as early as possible.

4.6 Conclusion

Thanks to the ever-increasing capacity of compute, storage and bandwidth at the edge of network, it has been a prominent trend that more and more end devices are enhanced by the power of artificial intelligence. Meanwhile, the rising concerns about data privacy are changing the way we develop machine learning techniques and also reveal the great potential of using Federated Learning as a promising privacy-preserving solution. This chapter presents a novel scheme that adapts FL to the mobile edge computing systems, aiming to improve both efficacy and efficiency. A three-layer FL algorithm called HybridFL is devised to enable two levels of model aggregation to boost efficiency and mitigate the impact by the unreliable nature of end devices through managing client selection in a region-wise manner, which results in a reasonable number of local updates as desired by the cloud. Extensive experiments demonstrate that HybridFL significantly improves FL in the MEC system by shortening the average length of a round, speeding up the convergence of the global model, promoting the model accuracy and reducing device-side energy consumption.

In the future, the research is planned be extended to more complex system architectures which have different hierarchies and to diverse FL participants which have different roles. Also, it is worth investigating how to improve the effectiveness of local training on each device without breaching the privacy constraints.

Chapter 5

Optimising Horizontal FL with Learnt Representations

A large proportion of data used for machine learning are often generated outside the data centres by distributed resources. Federated learning has shown great potential as a privacy-preserving solution to learning from decentralised data but its performance is largely affected by the distribution of data (both locally and globally) as well as data quality.

Much effort has been paid in optimising FL, especially in terms of its efficiency and efficacy. The existing solutions span a variety of perspectives including communication [123, 245, 247], update rules [85, 160], flexible aggregation [1, 55], local training optimisation [88, 230] and client selection [88, 159, 162]. However, little attention has been paid to the problems caused by the low-quality data on end devices and the challenge of how to minimise their impact on FL's efficiency and efficacy. Goetz et al. [159] proposed a loss-oriented client selection strategy, in which they prioritise the clients with high loss feedbacks. This strategy has the tendency to select the clients whose local models yield the higher errors, but it is also likely to favour the clients with noisy and irrelevant data (i.e., low quality data). Tuor et al. [230] pointed out that the clients of FL are likely to contain a lot of useless data and only a subset of them is valuable for the training task. They further propose a data selection method based on the loss distribution in order to filter out noisy and irrelevant data on local devices. However, their solution requires a baseline loss distribution produced by a pre-trained model, which could be difficult to set up without prior knowledge.

FL features strong privacy protection and allows for learning from decentralised data without relying on any distributional assumption. However, being completely data-agnostic can bring about problems.

5.1 Motivation

5.1.1 FL's Susceptibility to Data Quality

Generally, in each round of FL the server exchanges the model with only a fraction of clients selected for this round of training (involving too many clients leads to diminishing gains towards the quality of the global model [85]). The standard FL implementation (i.e., FedAvg [47]) selects clients randomly, which implies that every client (and its local data) is considered equally important as they have the same chance of being selected. However, a common problem of decentralised data sources is the **discrepancy in both data quality and data distribution** — a large proportion of on-device data (e.g., user-generated texts [15] and photos) can be biased and noisy. In some scenarios, local data may contain irrelevant or even adversarial samples [282][283]. The injection of low-quality or adversarial data is often used to launch model poisoning attacks against FL [155, 215, 283].

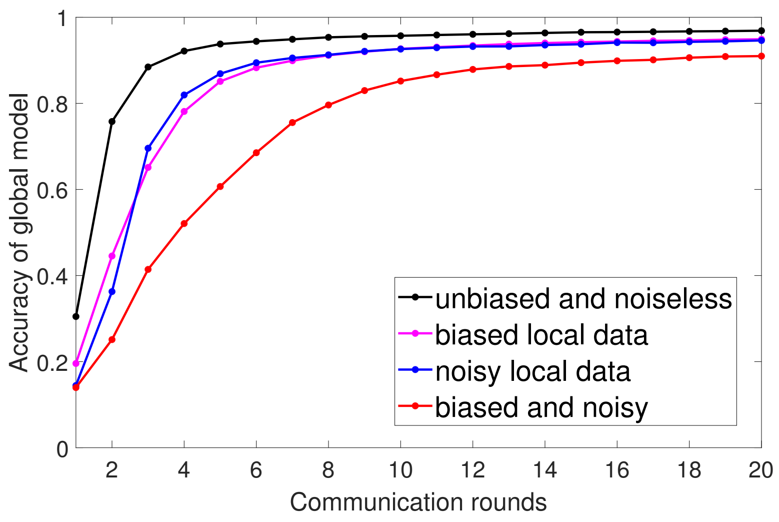


Figure 5.1: A demonstration of the global model's convergence under different data conditions. Detailed setup is given in Section 5.3.1.

For an intuitive demonstration, part of the *Preliminary Experiment* results is presented here as shown in Fig. 5.1 (with details provided in the corresponding Section 5.3.1). The toy experiment is devised to reveal the impact of involving clients with low-quality data by running FL over 100 clients with the aim of learning a CNN model on MNIST using the standard FedAvg algorithm. From the traces one can see that training over clients with problematic or strongly biased data can compromise the efficiency and effectiveness of FL, resulting in an inferior global model that takes more rounds to converge.

Traditional solutions to the problem include local data augmentation [284] and re-sampling [285]. But these methods are designed to correct the population distribution. Applying them to local datasets may introduce extra noise [286] and increase the risk of information leakage [287]. Another naive solution is to directly exclude those clients with 'low quality' data, which, however, is often impractical because i) the quality of the data depends on the learning task and is difficult to gauge, ii) some noisy or biased data could be useful to the training at early stages [288], and iii) sometimes low-quality data are very common across the clients.

5.1.2 Learnt representations can reflect data distribution and quality

Data representation learning is recognised as a cornerstone on which the success of modern machine learning (especially deep learning) is built [220]. Representation learning stems from feature learning methods (e.g., PCA) as a common approach to extracting useful information and capturing the intrinsic structures behind the data [289]. The development of deep models (neural networks particularly) further reveals the importance of data representations. Studies have shown that the ability to learn high-quality representations is critical to the model performance in many domains including CV and NLP [221, 290]. The value of representations also lies in the fact that they characterise the domain and learning task using the model as a medium [291, 292]. In the context of FL, the similarity of representations are used for refining the model update rules [105, 236], but the distributional difference of representations from heterogeneous data is not yet explored. With this motivation, the work to be presented in this chapter studies the distribution of data presentation and seeks to use representations as the 'lens' to reflect the training value of local data.

Explained from a technical perspective, representations are the result of one or multiple non-linear transformations of the raw data [220]. For example, deep Neural Networks (DNNs) hierarchically construct the representations in a layer-wise manner, as illustrated in Fig. 5.2.

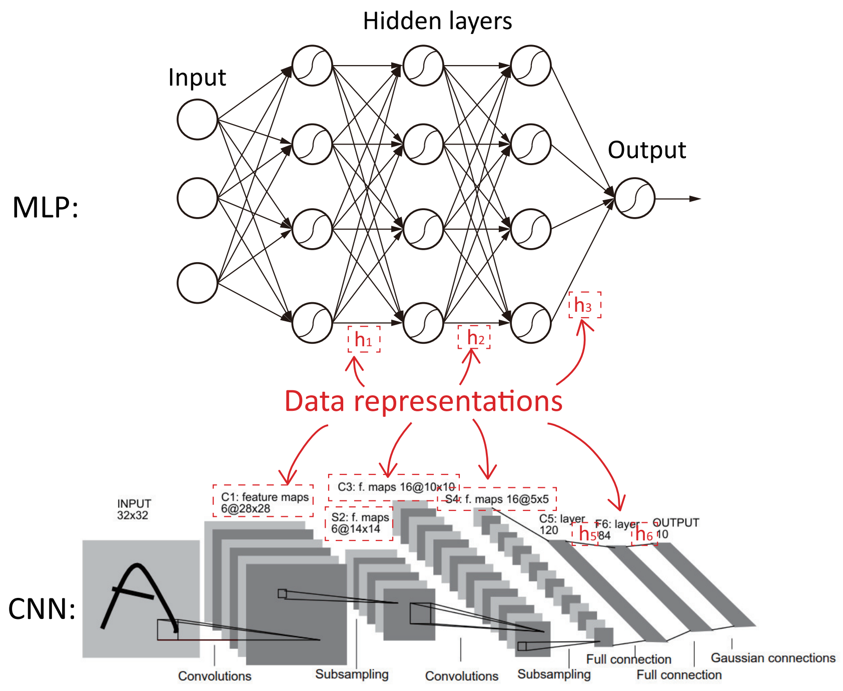


Figure 5.2: An illustration of data representations from a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN). The CNN architecture is from [21].

Good representations disentangle the underlying explanatory factors of variation behind the data and provide useful information for the downstream machine learning tasks. The utility of learnt representations has led to a diversity of ML paradigms such as transfer learning, knowledge distillation and split learning. Although a few studies explore the possibilities of using representations to optimise the efficiency and efficacy of federated learning, most of these approaches use presentations to connect split models [293] or are designed for Vertical FL [236] or Federated Transfer Learning scenarios [240, 241]. Li et al. [105] proposed a model-contrastive FL framework MOON using the distances between different models' representations to rectify local training. MOON is 'heavy' because three types of models need to be stored and inference on each client. More importantly, the method cannot address the poisoning of low-quality data on some clients.

5.2 Contributions

Motivated by the power of representation learning and the importance of data quality to FL, the aim of this work is to devise an optimised FL algorithm with robustness to low-quality local data from a distributional perspective.

- This work first provides theoretical proofs for the observation that data representations from neural networks tend to follow Gaussian distribution, based on which a representation profiling and matching scheme is proposed to enable fast, low-cost comparison between different representation profiles.
- A novel FL algorithm FedProf is proposed to achieve selective FL through adaptively adjusting clients' participation probability based on representation profile dissimilarities.
- Results of extensive experiments show that FedProf can reduce the number of communication rounds by up to 77%, shortens the overall training time (up to 4.5× speedup) while increasing the accuracy of the global model by up to 2.5%.

5.3 Preliminary Experiments

This section presents the results of preliminary experiments comprised of two parts. Part I is a toy experiment of FL with disparate conditions of training data for demonstrating the impact of low-quality data on FL's performance. Part II demonstrates the representations learnt in different machine learning tasks from a distributional perspective.

5.3.1 Part I: The Impact of Low-quality Data

A main motivation of this work is the susceptibility of FL to the quality of local training data. This part of experiment reveals this fact using a demonstration of the global model's convergence under different data conditions. The resulting plot is shown in Fig. 5.1. The data points are obtained by running the FL process (using the standard FedAvg algorithm [47]) with 100 clients to learn a CNN model on the MNIST dataset, which is partitioned and allocated to clients in four different ways where the data are 1) *original* (black line): noiseless and evenly distributed over clients, 2) *biased* (magenta line): locally class-imbalanced, 3) *noisy* (blue line): blended with noise, or 4) *biased and noisy* (red line). The noise (if applied) covers 65% of the clients; the dominant class accounts for >50% of the samples for biased local data. The fraction of selected clients is 0.3 for each round.

From the traces one can see that training over clients with problematic or strongly biased data can compromise the efficiency and effectiveness of FL, resulting in an inferior global model that takes more rounds to converge. In other words, local training data's quality significantly influences the convergence of the global model and its final accuracy.

5.3.2 Part II: Observations on Data Representations

The research to be presented is inspired by an intriguing finding as follows. During the training process of a CNN, the representations are recorded during the forward propagation and it turns out that the outputs of each neuron (before activation) in the dense layers appear to follow Gaussian distributions. Fig. 5.3 shows the outputs of several neurons randomly selected from the FC-1 (i.e., the first fully-connected layer) of a CNN model when being trained on MNIST. The figure displays the outputs of five randomly selected neurons (namely, #1, #15, #21, #75 and #127 out of 128 neurons in FC-1 for this example) at the epochs 1, 6 and 10. One can see the distribution of the neuron output matches the Gaussian distribution well since the first epoch of training.

Fig. 5.4 shows the data representations from the two hidden layers of a Feed-Forward Network (FFN) model after training (till convergence) on the Boston Housing dataset[1], with training set and test set representations displayed in different colors. One can again observe clear Gaussian-like distributional patterns of the neurons' output (i.e., individual components of the representation). The figure also exhibits the distributional distances between the representations of the training set (in blue) and the test set (in orange).

Further experiments show that the learnt representations from non-linear operators in a deep neural network also tend to follow the Gaussian distribution. As a demonstration a ResNet-18 model is trained on the image classification dataset CIFAR-100. Fig. 5.5 shows the distribution of fused representations (in a channel-wise manner) extracted from a plain convolution layer and a residual block within the network.

These observations motivate this research to study the distributional property of data representations and use it as a means to differentiate clients' value.

[1]`https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`

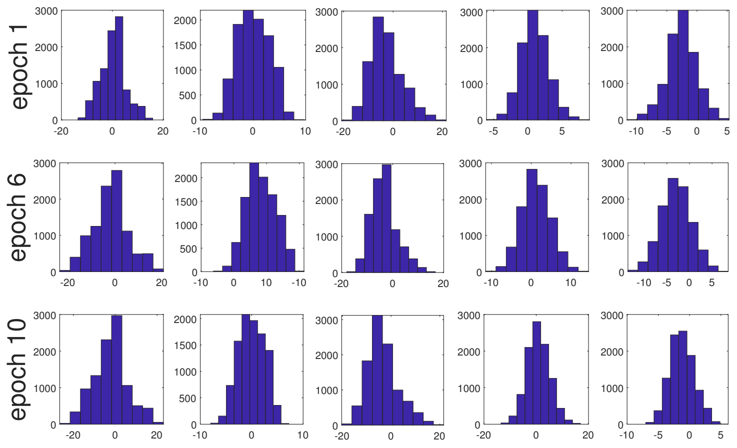


Figure 5.3: An example of training LeNet-5 on MNIST till convergence. After each epoch the model is evaluated and the pre-activation outputs of each neuron in the model's FC-1 layer are collected.

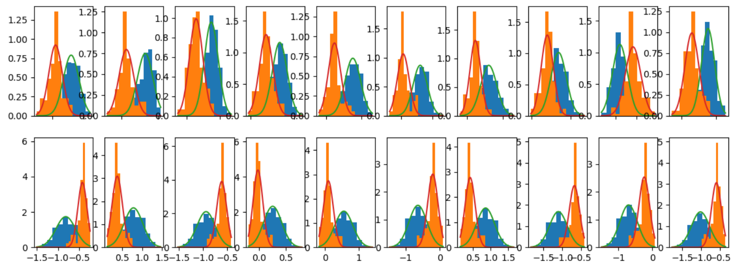


Figure 5.4: An example of training a two-layer FNN on the Boston Housing dataset till convergence. Depicted here are representations (blue for the training set, orange for the test set) extracted from the model's hidden layers in the form of density histograms.

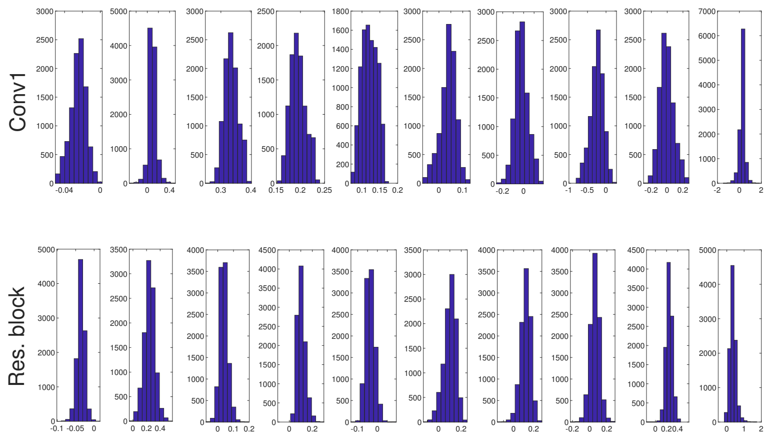


Figure 5.5: Fused representations from a standard convolution layer (1st row) and a residual block (2nd row) of a ResNet-18 model trained for 100 epochs on CIFAR-100.

5.4 Representation Profiling and Matching

5.4.1 Gaussian Distribution of Representations

In this chapter, a typical cross-device FL setting [49] is considered, where multiple end devices collaboratively perform local training on their own datasets $D_i, i = 1, 2, ..., N$. The server owns a validation dataset D^* for model evaluation. Every dataset is only accessible to its owner. It is usually unrealistic to profile the raw data. On the one hand, the raw feature space such as images can be very high-dimensional and the distribution of samples in each dimension could follow any random pattern, which makes it difficult to profile. On the other hand, the resulting profiles of local data need to be sent to the server for comparison, which presents the risk of leaking user information. Therefore, this work proposes a novel method which leverages the global model as a medium to profile the *latent representations of data*.

This work adopts a representation profiling method on the basis of latent representations of data from the global model. The reasons why it is proposed to exploit the global model as the medium for data representation profiling are multi-fold. First, introducing a separate data encoder incurs the extra cost in communication and training. Second, once properly initialised, the hidden layer(s) of the global model can act as a natural encoder that maps the raw data to a set of feature embeddings (i.e., representations) which become more sensitive to the input data of different qualities as the global model improves during the FL process. Besides, since the global model is synchronised with the clients every training round, one can guarantee that an identical encoder is used for embedding both local data on the clients and the benchmark data on the server.

The observation presented in the Preliminary Experiments (Part II) empirically shows clear distributional patterns for the representations learnt by a neural network. In this section, theoretical proofs are first provided to support the observation that learnt representations from neural network models tend to follow Gaussian distribution. Then a novel scheme is presented for profiling data representations and and a measure called profile dissimilarity is defined for fast and secure representation comparison.

The following definition is made to facilitate the analysis.

Definition 5.1 (The Lyapunov's condition). *A set of random variables* $\{Z_1, Z_2, \ldots, Z_v\}$

satisfy the Lyapunov's condition if there exists a δ such that

$$\lim_{v\to\infty} \frac{1}{s^{2+\delta}} \sum_{k=1}^{v} \mathbb{E}\left[|Z_k - \mu_k|^{2+\delta}\right] = 0, \tag{5.1}$$

where $\mu_k = \mathbb{E}[Z_k]$, $\sigma_k^2 = \mathbb{E}[(Z_k - \mu_k)^2]$ and $s = \sqrt{\sum_{k=1}^{v} \sigma_k^2}$.

The Lyapunov's condition can be intuitively explained as a limit on the overall variation (with $|Z_k - \mu_k|^{2+\delta}$ being the $(2+\delta)$-th moment of Z_k) of a set of random variables.

Regarding the distributional property of data representations from linear operators, Proposition 5.1 is formally stated as follows.

Proposition 5.1. *The representations from linear operators (e.g., a pre-activation dense layer or a plain convolutional layer) in a neural network tend to follow the Gaussian distribution if the layer's weighted inputs satisfy the Lyapunov's condition.*

Without loss of generality, this work provides the proof of Proposition 5.1 for the pre-activation representations from dense (fully-connected) layers and standard convolutional layers, respectively. The results can be easily extended to other linear neural operators.

Dense layers

Proof. Let $\Omega = \{neu_1, neu_2, ..., neu_q\}$ denote a dense layer (with q neurons) of any neural network model and H_k denote the pre-activation output of neu_k in Ω. The theoretical proof is first provided to support the observation that H_k tends to follow the Gaussian distribution.

Let $\chi = \mathbb{R}^v$ denote the input feature space (with v features) and assume the feature X_i (which is a random variable) follows a certain distribution $\zeta_i(\mu_i, \sigma_i^2)$ (not necessarily Gaussian) with finite mean $\mu_i = \mathbb{E}[X_i]$ and variance $\sigma_i^2 = \mathbb{E}[X_i - \mu_i]$. For each neuron neu_k, let $\boldsymbol{w}_k = [w_{k,1}\, w_{k,2} \dots w_{k,v}]$ denote the neuron's weight vector, b_k denote the bias, and $Z_{k,i} = X_i w_{k,i}$ denote the i-th weighted input. Let H_k denote the output of neu_k. During the forward propagation, it follows that:

$$\begin{aligned} H_k &= X\boldsymbol{w}_k^T + b_k \\ &= \sum_{i=1}^{v} X_i w_{k,i} + b_k \\ &= \sum_{i=1}^{v} Z_{k,i} + b_k. \end{aligned} \tag{5.2}$$

Apparently $Z_{k,i}$ is a random variable because $Z_{k,i} = X_i w_{k,i}$ (where the weights $w_{k,i}$ are constants during a forward pass), thus H_k is also a random variable according to Eq. (5.2).

In an ideal situation, the inputs variables $X_1, X_2, \dots, X_v$ may follow a multivariate Gaussian distribution, in which case Proposition 5.1 automatically holds due to the property of multivariate normal distribution that every linear combination of the components of the random vector $(X_1, X_2, \dots, X_v)^T$ follows a Gaussian distribution [294]. In other words, $H_k = X_1 w_{k,1} + X_2 w_{k,2} + \ldots + X_v w_{k,v} + b_k$ is a normally distributed variable since $w_{k,i}$ and b_k $(k = 1, 2, \dots, v)$ are constants in the forward propagation. A special case for this condition is that $X_1, X_2, \dots, X_v$ are independent on each other and X_i follows a Gaussian distribution $\mathcal{N}(\mu_i, \sigma_i^2)$ for all $i = 1, 2, \dots, v$. In this case, by the definition of $Z_{k,i}$, it follows that

$$Z_{k,i} = X_i w_{k,i} \sim \mathcal{N}\big(w_{k,i}\mu_i, (w_{k,i}\sigma_i)^2\big), \tag{5.3}$$

where $Z_1, Z_2, \dots, Z_v$ are independent on each other. Combining Eqs. (5.2) and (5.3), one can derive that

$$H_k \sim \mathcal{N}\big(\sum_{i=1}^{v} w_{k,i}\mu_i + b_k, \sum_{i=1}^{v}(w_{k,i}\sigma_i)^2\big), \tag{5.4}$$

For more general cases where $X_1, X_2, \dots, X_v$ are not necessarily normally distributed, it is assumed that the weighted inputs $Z_{k,i}$ of the dense layer satisfy the Lyapunov's condition (see *definition* 5.1). As a result, the following holds according to the Central Limit Theorem (CLT) [295] considering that X_i follows $\zeta_i(\mu_i, \sigma_i^2)$:

$$\frac{1}{s_k} \sum_{i=1}^{v} \big(Z_{k,i} - w_{k,i}\mu_i\big) \xrightarrow{d} \mathcal{N}(0, 1) \tag{5.5}$$

where $s_k = \sqrt{\sum_{i=1}^{v} \big(w_{k,i}\sigma_i\big)^2}$ and $\mathcal{N}(0, 1)$ denotes the standard normal distribution.

Equivalently, for every neu_k it follows that

$$\sum_{i=1}^{v} Z_{k,i} \xrightarrow{d} \mathcal{N}(\sum_{i=1}^{v} w_{k,i}\mu_i, s_k^2) \tag{5.6}$$

Combining Eqs. (5.2) and (5.6) one can derive that

$$H_k \xrightarrow{d} \mathcal{N}(\sum_{i=1}^{v} w_{k,i}\mu_i + b_k, s_k^2), \tag{5.7}$$

which means that $H_k (k = 1, 2, \ldots, v)$ tend to follow Gaussian distributions and proves Proposition 5.1 for fully-connected layers.

□

Convolutional layers

Proof. Standard convolution in CNNs is also a linear transformation of the input feature space and its main difference from dense layers rests on the restricted size of receptive field. Without loss of generality, this work analyses the representation (output) of a single kernel. To facilitate the analysis for convolutional layers, let C denote the number of input channels and K denote the kernel size. For ease of presentation, a receptive field mapping function $\Theta(k, i, j)$ is defined for mapping the positions (k for channel index, i and j for indices on the same channel) of elements in the feature map (i.e., the representations) to the input features. For the k-th kernel, let W_k denote its weight tensor (with $W_{k,c}$ being the weight matrix for channel c) and b_k its bias.

Given the corresponding input patch $X_{\Theta(k,i,j)}$, The element $H_{k,i,j}$ of the representations from a convolutional layer can be formulated as:

$$H_{k,i,j} = \sum_{c=1}^{C} \sum_{i'=1}^{K} \sum_{j'=1}^{K} \left(X_{\Theta(k,i,j)} \circ W_{k,c} \right)_{i',j'} + b_k, \tag{5.8}$$

where $\circ$ denotes Hadamard product. The three summations reduce the results of element-wise product between the input patch and the k-th kernel to the correspond representation element $H_{k,i,j}$ in the feature map. For ease of presentation, here the notation $Z^{(k)}_{c,i',j'}$ is used to replace $\left(X_{\Theta(k,i,j)} \circ W_{k,c} \right)_{i',j'}$ and let $\zeta(\mu_{c,i',j'}, \sigma^2_{c,i',j'})$ be the distribution that $Z^{(k)}_{c,i',j'}$ follows. Note that ζ can be any distribution since this work makes no distributional assumption on $Z^{(k)}_{c,i',j'}$.

With the notations, Eq. (5.8) can be rewritten in a similar form to Eq. (5.2):

$$H_{k,i,j} = \sum_{c=1}^{C} \sum_{i'=1}^{K} \sum_{j'=1}^{K} Z^{(k)}_{c,i',j'} + b_k. \tag{5.9}$$

Then the condition is used that the random variables $Z^{(k)}_{c,i',j'}$ satisfy the Lyapunov's condition, i.e., there exists a δ such that

$$\lim_{C\times K^2 \to \infty} \frac{1}{s^{2+\delta}} \sum_{c=1}^{C} \sum_{i'=1}^{K} \sum_{j'=1}^{K} \mathbb{E}\left[|Z^{(k)}_{c,i',j'} - \mu_{c,i',j'}|^{2+\delta}\right] = 0, \tag{5.10}$$

where $s = \sqrt{\sum_{c=1}^{C} \sum_{i'=1}^{K} \sum_{j'=1}^{K} \sigma^2_{c,i',j'}}$.

Then according to the Lyapunov CLT, the following holds:

$$H_{k,i,j} \xrightarrow{d} \mathcal{N}(\sum_{c,i',j' \in \Theta(k,i,j)} \mu_{c,i',j'} + b_k, \sum_{c,i',j' \in \Theta(k,i,j)} \sigma^2_{c,i',j'}), \tag{5.11}$$

which proves Proposition 5.1 for standard convolution layers.

□

Regarding the distributional property of data representations from non-linear operators, Proposition 5.2 is formally stated as follows.

Proposition 5.2. *The fused representations*[2] *from non-linear operators (e.g., a hidden layer of LSTM or a residual block of ResNet) in a neural network tend to follow the Gaussian distribution if the layer's output elements satisfy the Lyapunov's condition.*

Without loss of generality, this work proves Proposition 5.2 for the fused representations from the LSTM layer and the residual block of ResNet models, respectively. The results can be easily extended to other non-linear neural operators.

LSTM

Proof. Long Short-Term Memory (LSTM) models are popular for extracting useful representations from sequence data for tasks such as speech recognition and language modelling. Each LSTM layer contains multiple neural units. For the k-th unit, it takes as input the current input feature vector $X_t = (X_{t,1}, X_{t,2}, \ldots)$, hidden state vector H_{t-1} and its cell state $c_{t-1,k}$. The outputs of the unit are its new hidden state $h_{t,k}$ and cell state $c_{t,k}$. This work studies the distribution of $h_{t,k}$. Multiple gates are

[2] *Fused* representations refer to the sum of elements in the original representations produced by a single layer (channel-wise for a residual block).

adopted in an LSTM unit: $i_{t,k}$, $f_{t,k}$, $g_{t,k}$ and $o_{t,k}$ respectively denote the input gate, forget gate, cell gate and output gate of the LSTM unit k at time step t. The update rules of these gates and the cell state are:

$$\begin{aligned}
i_{t,k} &= \text{sigmoid}(W_{(i)k}[H_{t-1}, X_t] + b_{(i)k}),\\
f_{t,k} &= \text{sigmoid}(W_{(f)k}[H_{t-1}, X_t] + b_{(f)k}),\\
g_{t,k} &= \tanh(W_{(g)k}[H_{t-1}, X_t] + b_{(g)k}),\\
o_{t,k} &= \text{sigmoid}(W_{(o)k}[H_{t-1}, X_t] + b_{(o)k}),\\
c_{t,k} &= f_{t,k} \cdot c_{t-1,k} + i_{t,k} \cdot g_{t,k},
\end{aligned} \tag{5.12}$$

where the $W_{(i)k}$, $W_{(f)k}$, $W_{(g)k}$ and $W_{(o)k}$ are the weight parameters and $b_{(i)k}$, $b_{(f)k}$, $b_{(g)k}$ and $b_{(o)k}$ are the bias parameters for the gates.

The output of the LSTM unit $h_{t,k}$ is calculated as follows:

$$h_{t,k} = o_{t,k} \cdot \tanh(c_{t,k}). \tag{5.13}$$

Using the final hidden states $h_{T,k}$ (with T being the length of the sequence) as the elements of the layer-wise representation, one can apply the following layer-wise fusion to further produce H over all the $h_{T,k}$ in a single LSTM layer:

$$H = \sum_{k=1}^{d} h_{T,k}, \tag{5.14}$$

where d is the dimension of the LSTM layer. Again, let $\zeta(\mu_k, \sigma_k^2)$ denote the distribution of $h_{T,k}$ (where the notation T is dropped here since it is typically a fixed parameter). With $\{h_{T,k} | k = 1, 2, \ldots, d\}$ satisfying the Lyapunov's condition and by Central Limit Theorem, H tends to follow the Gaussian distribution:

$$H \xrightarrow{d} \mathcal{N}(\sum_{i=1}^{d} \mu_k, \sum_{i=1}^{d} \sigma_k^2), \tag{5.15}$$

which proves the Proposition 5.2 for layer-wise fused representations from LSTM. □

Residual blocks

Proof. Residual blocks are the basic units in the Residual neural network (ResNet) architecture [296]. A typical residual block contains two convolutional layers with batch normalisation (BN) and uses the ReLU activation function. The input of the whole block is added to the output of the second convolution (after BN) through a skip connection before the final activation. Since the convolution operators are the same as it is formulated before, here the notation $\Psi(X)$ is used to denote the sequential operations of convolution on X followed by BN, i.e., $\Psi(X) \triangleq BN(Conv(X))$. Again, one can reuse the receptive field mapping $\Theta(k, i, j)$ as defined for plain convolution operations to position the inputs of the residual block corresponding to the element $Z_{k,i,j}$ in the output representation of the whole residual block.

Let X denote the input of the residual block and $Z_{k,i,j}$ denote an element in the output tensor of the whole residual block. Then it follows that

$$\begin{aligned} Z_{k,i,j} &= f\Big(X_{k,i,j} + BN\big(Conv\big(f\big(BN(Conv(X_{\Theta(k,i,j)}))\big)\big)\big)\Big) \\ &= f\Big(X_{k,i,j} + \Psi\big(f(\Psi(X_{\Theta(k,i,j)}))\big)\Big), \end{aligned} \tag{5.16}$$

where f is the activation function (ReLU).

One can perform channel-wise fusion on the representation from the residual block to produce H_k for the k-th channel:

$$H_k = \sum_{i=1}^{d_H} \sum_{j=1}^{d_W} Z_{k,i,j}, \tag{5.17}$$

where d_H and d_W are the dimensions of the feature map and k is the channel index.

Let $\zeta(\mu_{k,i,j}, \sigma^2_{k,i,j})$ denote the distribution that $Z_{k,i,j}$ follows. Then one can apply the Lyapunov's condition to the representation elements layer-wise, i.e.,

$$\lim_{d_W \times d_H \to \infty} \frac{1}{s_k^{2+\delta}} \sum_{i=1}^{d_H} \sum_{j=1}^{d_W} \mathbb{E}\left[|Z_{k,i,j} - \mu_{k,i,j}|^{2+\delta}\right] = 0, \tag{5.18}$$

where $s_k = \sqrt{\sum_{i=1}^{d_H} \sum_{j=1}^{d_W} \sigma^2_{k,i,j}}$.

With the above condition satisfied, by CLT H_k (the fused representation on channel k) tends to follow the Gaussian distribution:

$$H_k \xrightarrow{d} \mathcal{N}(\sum_{i=1}^{d_H} \sum_{j=1}^{d_W} \mu_{k,i,j}, \sum_{i=1}^{d_H} \sum_{j=1}^{d_W} \sigma^2_{k,i,j}), \tag{5.19}$$

which proves the Proposition 5.2 for channel-wise fused representations from any residual block. $\square$

In practice, the Lyapunov's condition is typically met for both Propositions when the model is properly initialised and batch normalisation is applied.

Next section discusses the proposed representation profiling and matching protocol.

5.4.2 Distributional Profiling and Matching

Based on the Gaussian pattern of representations, the proposed approach compresses the data representations statistically into a compact form called *representation profiles*. The profile produced by the global model w on a dataset D, denoted by $RP(w, D)$, has the following format:

$$RP(w, D) = \{\mathcal{N}(\mu_i, \sigma_i^2)\}_{i=1}^{q}, \tag{5.20}$$

where q is the profile length determined by the dimensionality of the representations. For example, q is equal to the number of kernels for channel-wise fused representations from a convolutional layer. The tuple (μ_i, σ_i^2) contains the mean and the variance of the i-th representation element.

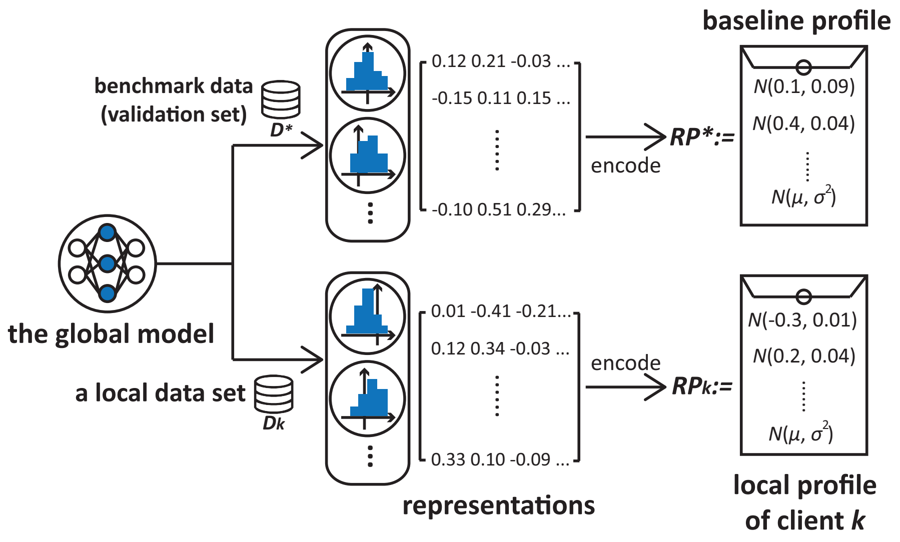


Figure 5.6: The proposed representation profiling scheme: the representations of the data are encoded as a sequence of (μ_i, σ_i^2) pairs, where μ_i and σ_i^2 are the mean and the variance, respectively, of an element in the representation vector.

Fig. 5.6 illustrates the workflow of the proposed data representation profiling scheme. The profiles are generated through the model evaluation, the cost of which is marginal compared to that of training the model. The profiles generated for the data on the clients need to be transmitted to the server for the comparison with the baseline profile obtained from the benchmark data on the server. The transmission cost is negligible since the size of a profile (as a sequence of (μ, σ^2) pairs) is very small compared to that of the model (note that the model is updated by a client in every round and needs to be uploaded to the server). Let RP_k denote the local profile from client k, and RP^* denote the baseline profile on the server. The dissimilarity between RP_k and RP^*, denoted by $div(RP_k, RP^*)$, can be determined by Eq. (5.21):

$$div(RP_k, RP^*) = \frac{1}{q}\sum_{i=1}^{q} \mathrm{KL}(\mathcal{N}_i^{(k)} || \mathcal{N}_i^*), \tag{5.21}$$

where KL($\cdot$) denotes the Kullback–Leibler (KL) divergence. An advantage of the proposed profiling scheme is that a *much simplified KL divergence formula* can be adopted because the latent representations of FC-1 enjoy the following property (see Appendix B, [297] for details):

$$\mathrm{KL}(\mathcal{N}_i^{(k)} || \mathcal{N}_i^*) = \log \frac{\sigma_i^*}{\sigma_i^{(k)}} + \frac{(\sigma_i^{(k)})^2 + (\mu_i^{(k)} - \mu_i^*)^2}{2(\sigma_i^*)^2}, \tag{5.22}$$

With Eq. (5.22), the KL divergence between two Gaussian distributions can be computed without calculating any integral, which largely reduces the computation cost. Besides, the computation of profile dissimilarity can be performed under the Homomorphic Encryption to achieve minimum knowledge disclosure [53] (see Appendix B for details). The data representation profiles are associated to the global model and thus change over rounds. Thus the proposed method tags every profile (including both local and baseline profiles) with the version number of the global model so that profile comparisons are version-aligned.

5.5 FedProf: Selective FL with Representation Profiling

In this chapter, a typical cross-device FL setting [49] is considered, where multiple end devices perform local training on their data and the server owns a set of validation data for model evaluation. Local data are bound to and only accessible to the corresponding clients. The validation data are used for the round-wise evaluation of the global model and are only accessible to the server.

This research aims to optimise the global model's convergence in FL over a large group of clients that own the data of diverse quality. Given the client set $U(|U| = N)$, let D_k denote the local dataset on client k and D^* the validation set on the server, The optimisation problem is formulated by (5.23) where the coefficient ρ_k differentiates the importance of the local objective functions $F_k(w)$. The global objective implies a non-uniform mixture of local data distributions and is in a sense similar to the agnostic learning scenario [209].

$$\arg\min_{w} F(w) = \sum_{k \in U} \rho_k F_k(w), \tag{5.23}$$

where w is the parameter vector of the global model[3]. The coefficients $\{\rho_k\}_{k \in U}$ add up to 1. $F_k(w)$ is client k's local objective function of training based on the loss function $\ell(\cdot)$:

$$F_k(w) = \frac{1}{|D_k|} \sum_{(x_i, y_i) \in D_k} \ell(x_i, y_i; w), \tag{5.24}$$

The key of the proposed strategy is to score each client by their training value in each round and adjust the clients' chance of participation accordingly. Given the selection fraction C, one can use a weighted random selection method[4] to choose a subset S of $N \cdot C$ participants from the client set U. Let λ_k denote client k's weight in the selection, the value of λ_k is determined by the dissimilarity between RP_k and RP^*:

$$\lambda_k = \exp\big(- \alpha_k \cdot div(RP_k, RP^*)\big), \tag{5.25}$$

where RP_k and RP^* are generated by an identical global model; α_k is the preference factor deciding how biased the strategy needs to be towards client k given its profile dissimilarity. With $\alpha_k = 0$ for all $k = 1, 2, \dots N$, our strategy degenerates to random selection. Each $div(RP_k, RP^*)$ is normalised by the sum $\Lambda = \sum_{k \in U} \lambda_k$ for converting to probability.

The scores connect the representation profiling and matching scheme to the design of the selective client participation strategy adopted in the proposed FL training algorithm FedProf, which is outlined in Algorithm 4.

Note that only the selected clients are required to update and upload local profiles in each round. Hence, an initialisation step is added before the start of the first FL round in order to obtain the initial profiles from the clients and initialise $\{\lambda_k\}_{k \in U}$

[3]The global model is a hypothesis $h_w \in \mathcal{H} : \chi \to \mathcal{Y}$. This work refers to it as w for brevity.

[4]For example, the `random.choices(U, weights, NC)` function provided in the Python Standard Library

Algorithm 4: the FedProf algorithm

```
Input  : maximum number of rounds $T_{max}$, local iterations per round $E$,
         client set $U$, client fraction $C$, validation set $D^*$;
Output : the global model $w$
// Server process: running on the server
1  Initialise global model $w$ using a seed
2  $v \leftarrow 0$  // version of the latest global model
3  Broadcast the seed to all clients for identical model initialisation
4  Collect initial profiles $\{RP_k\}_{k \in U}$ from all the clients
5  $v_k \leftarrow 0, \forall k \in U$
6  Generate initial baseline profile $RP^*(0)$ on $D^*$
7  $K \leftarrow |U| \cdot C$
   for round $T \leftarrow 1$ to $T_{max}$ do
8      Calculate $div(RP_k(v_k), RP^*(v_k))$ for each client $k$
9      Update client scores $\{\lambda_k\}_{k \in U}$ and compute $\Lambda = \sum_{k \in U} \lambda_k$
10     $S \leftarrow$ Choose $K$ clients by probability distribution $\{\frac{\lambda_k}{\Lambda}\}_{k \in U}$
11     Distribute $w$ to the clients in $S$
       for client $k$ in $S$ in parallel do
12         $v_k \leftarrow v, \forall k \in S$
13         $RP_k(v_k) \leftarrow$ updateProfile($k, w, v$)
14         $w_k \leftarrow$ localTraining($k, w, E$)
       end
15     Collect local profiles from the clients in $S$
16     Update $w$ via model aggregation
17     $v \leftarrow T$
18     Evaluate $w$ and generate $RP^*(v)$
   end
19 return $w$
// Client process: running on client $k$
   updateProfile($k, w, v$):
20     Generate $RP_k$ on $D_k$ with the global $w$ received
21     Label profile $RP_k$ with version number $v$
22     Return $RP_k$
   return
   localTraining($k, w, E$):
23     $w_k \leftarrow w$
       for step $e \leftarrow 1$ to $E$ do
24         Update $w_k$ using gradient-based method
       end
25     Return $w_k$
   return
```

properly. To do so, the server broadcasts a seed to all the clients for generating an identical initial global model across all clients and generating their first local profiles. The server needs to take four steps regarding the client selection:

1) Calculate $\{\lambda_k\}_{k\in U}$ according to Eq. (5.25).

2) Select $n \cdot C$ clients using weighted random selection with λ_k being client k's weight.

3) Collect and update local profiles from the selected clients.

4) Update the baseline profile on the server (after global aggregation).

The global model w is updated by aggregating local models at the end of each round (line 16). The proposed algorithm supports both full aggregation [47][55] and partial aggregation [85] methods that are adopted in the literature.

The convergence rate of FL algorithms with opportunistic client selection (sampling) has been extensively studied in the literature [85, 122]. Inspired by [55, 85], Theorem 5.1 is presented to guarantee the convergence of the proposed algorithm. Similar to [84, 85, 298], it is assumed that $\{F_k\}_{k=1}^N$ are L-smooth and μ-strongly convex and that in expectation, the variance of local stochastic gradients are bounded by ϵ^2 and their squared norms are bounded by G^2.

In the following analysis, the notation $w^{(r)}(t)$ is used to denote the model at time step t in round r. Also, it is defined that $w^{(r)} = w(rE)$ where E is the number of update steps per round (i.e., aggregation interval). For ease of presentation, the selected set S is defined for every time step t and $S(t) = S^{(r)}$ holds for all $t \in \{(r-1)E, (r-1)E+1, \ldots, rE\}$ if t is a time step within round r.

Theorem 5.1. *Using partial aggregation and our selection strategy that satisfies* $\alpha_k = -\frac{ln(\Lambda\rho_k)}{div(RP_k,RP^*)}$, *the global model* $w(t)$ *converges in expectation given an aggregation interval* $E \geq 1$ *and a decreasing step size (learning rate)* $\eta_t = \frac{2}{\mu(t+\gamma)}$.

$$\mathbb{E}\big[F(w(t))\big] - F^* \leq \frac{L}{(\gamma+t)}\Big(\frac{2(\mathcal{B}+\mathcal{C})}{\mu^2} + \frac{\gamma+1}{2}\Delta_1\Big), \tag{5.26}$$

where $t \in T_A = \{nE|n = 1,2,\ldots\}$, $\gamma = \max\{\frac{8L}{\mu}, E\} - 1$, $\mathcal{B} = \sum_{k=1}^N \rho_k^2\epsilon_k^2 + 6L\Gamma + 8(E-1)^2G^2$, $\mathcal{C} = \frac{4}{K}E^2G^2$, $\Gamma = F^* - \sum_{k=1}^N \rho_k F_k^*$, $\Delta_1 = \mathbb{E}\|\bar{w}(1) - w^*\|^2$, $\Lambda = \sum_{k=1}^N \lambda_k$, *and* $K = |S(t)| = N \cdot C$.

The proof of Theorem 5.1 is provided in Appendix C.

5.6 Experiments

Extensive experiments were conducted to evaluate fedProf under various FL settings, which cover multiple scales of client population and different machine learning tasks. Apart from the original FL algorithm FedAvg [47], several state-of-the-art FL algorithms were reproduced for comparison. For a fair comparison, the algorithms are grouped by the aggregation method (i.e., full aggregation and partial aggregation) and configured with the optimal hyper-parameter values following the settings in their papers (if any). Table 5.1 summarises these FL algorithms in terms of their aggregation methods and client selection strategies (the algorithms may differ in the specific rules for updating the global model during the aggregation; see the referenced papers for details). Note that the proposed algorithm can adapt to both aggregation methods.

Table 5.1: The implemented FL algorithms for comparison

Algorithm	Aggregation method	Rule of selection
FedAvg [47]	full	random
CFCFM [1]	full	by the order of submission
FedAvg-RP	partial(Scheme II, [85])	random
FedProx [88]	partial	weighted random by data ratio
FedAdam [160]	partial with momentum	random
AFL [159]	partial with momentum	by the loss on local data
fedProf (ours)	full/partial	weighted random by score

This work focuses on optimising the efficiency of FL in terms of the convergence speed and the final quality of the global model. Therefore, the experiment recorded the best accuracy of the global model that each algorithm can achieve given a maximum number of rounds, and also compare the algorithms in terms of the number of communication rounds, the running time and the energy consumption needed to reach a target accuracy. For brevity, if not specified the amount of improvement is reported over FedAvg and FedAvg-RP as the baselines for the full and partial aggregation groups, respectively.

5.6.1 Experimental Setup

The algorithms were evaluated in a discrete event-driven, simulation-based experimental environment where the FL system was built based on Python and the PyTorch framework (Build 1.7.0) same as that introduced in Section 3.7.1, Chapter 3. The relevant system configurations for this work are detailed in Table 5.2. The client–server communication was simulated via in-memory tensor exchange wherein the server kept an event-driven timer to track the time consumptions. The system calculates device energy consumption using power models. Details regarding the time and energy costs of local training and communication will be formulated later in this section.

A small-scale task (called S-Task) was first set up to learn a multi-layer feed-forward network (FFN) model from the sensor data, which are distributed over a relatively small number of sources, for predicting the carbon monoxide (CO) and nitrogen oxides (NOx) emissions. Next, a large-scale task (called L-Task) was set up to train a global model over a large population of user devices (e.g., smart phones) for the recognition of hand-written digits. In both tasks, each user device possesses the private data and data sharing is not allowed between any parties. The experiment uses the *GasTurbine*[5] and *EMNIST digits*[6] datasets for the S-Task and L-Task, respectively. The data are non-IID across the end devices.[7] For both tasks, a diversity of noises are introduced into the local datasets on end devices to simulate the discrepancy in data quality.

Detailed experimental settings are listed in Table 5.2. In the S-Task, the total population is 50 and the data collected by a proportion of the sensors (i.e., end devices of this task) are of low-quality: 10% of the sensors have no valid data and 40% of them produce noisy data. In the L-Task, a relatively large population (1,000 end devices) was set and the data (from *EMNIST digits*) spread across the devices with strong class imbalance — on each device, the dominant class accounts for roughly 60% of the samples. Besides, many local datasets are of low-quality: the images on 15% of the clients are irrelevant (valueless for the training of this task), 20% are (Gaussian) blurred, and 25% are affected by the salt-and-pepper noise (random black and white dots on the image, density=0.3). For the S-Task, the maximum number of rounds t_{max} is set to 100 for both aggregation modes, whilst for the L-Task, it is set to 300 and 50 for the full aggregation and partial aggregation, respectively. The preference factors for the proposed algorithm are all set to 10. Considering the population of clients, the setting of selection fraction C is based on the suggested scale of training participants in [49].

To simulate a realistic FL system that consists of disparate end devices, the clients are heterogeneous in terms of both performance and communication bandwidth (see

[5]`https://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set`

[6]`https://www.nist.gov/itl/products-and-services/emnist-dataset`

[7]In the S-Task, local datasets are of different sizes that follow a Gaussian distribution. In the L-Task, local data are largely imbalanced wherein on each client, roughly 60% of its samples have the same class label.

Table 5.2). A validation set is kept by the server (as the benchmark data) and is used for model evaluation.

Table 5.2: Experimental setup.

Setting	Symbol	S-Task	L-Task
Model	w	MLP	CNN
Dataset	D	GasTurbine	EMNIST digits
Total size	$\|D\|$	36.7k	280k
Validation set size	$\|D^*\|$	11.0k	40k
Client population	N	50	1000
Data distribution	-	$\mathcal{N}(514, 154^2)$	non-IID, dc0.6
Noise applied	-	fake, gaussian	fake, blur, s&p
Client spec. (GHz)	s_k	$\mathcal{N}(0.5, 0.1^2)$	$\mathcal{N}(1.0, 0.1^2)$
Comm. bandwidth (MHz)	bw_k	$\mathcal{N}(0.5, 0.1^2)$	$\mathcal{N}(1.0, 0.1^2)$
Signal-noise ratio	SNR	1e2	1e2
Bits per sample	BPS	11*8*8	28*28*1*8
Cycles per bit	CPB	300	400
# of local epochs	M	2	5
Loss function	f	MSE Loss	NLL Loss
Learning rate	η	1e-2	1e-2
lr decay	-	0.99	0.99

In each FL round, the server selects a fraction (i.e., C) of clients, distributes the global model to these clients and waits for them to finish the local training and upload the models. Given a selected set of clients S, the time duration of an FL round can be formulated as:

$$T_{round} = \max_{k \in S}\{T_k^{comm} + T_k^{train} + T_k^{RP}\} \tag{5.27}$$

where T_k^{comm} and T_k^{train} are the device–server communication time and local training time, respectively. T_k^{RP} is the time for generating and uploading local profiles (other algorithms do not have the term T_k^{RP}).

This work assumes wireless communication between the clients and the server. T_k^{comm} can be modelled by Eq. (5.28) according to [98], where bw_k is the downlink bandwidth of device k (in MHz); *SNR* is the Signal-to-Noise Ratio of the communication channel, which is set to be constant as the end devices, in general, are coordinated by the base stations for balanced *SNR* with fairness-based policies; $msize$ is the size of the model; the model upload time is twice as much as that for model download since the uplink bandwidth is set as 50% of the downlink bandwidth.

$$\begin{aligned} T_k^{comm} &= T_k^{upload} + T_k^{download} \\ &= 2 \times T_k^{download} + T_k^{download} \\ &= 3 \times \frac{msize}{bw_k \cdot \log(1 + SNR)}, \end{aligned} \tag{5.28}$$

The T_k^{train} in Eq. (5.27) can be modeled by Eq. (5.29), where s_k is the device performance (in GHz) and the numerator computes the total number of processor cycles required for processing M epochs of local training on D_k.

$$T_k^{train} = \frac{M \cdot |D_k| \cdot BPS \cdot CPB}{s_k}, \tag{5.29}$$

T_k^{RP} consists of two parts: T_k^{RPgen} for local model evaluation (to generate the profiles of D_k) and T_k^{RPup} for uploading the profile. T_k^{RP} can be modeled as:

$$\begin{aligned} T_k^{RP} &= T_k^{RPgen} + T_k^{RPup} \\ &= \frac{1}{M} T_k^{train} + \frac{RPsize}{\frac{1}{2} bw_k \cdot \log(1 + SNR)}, \end{aligned} \tag{5.30}$$

where T_k^{RPgen} is estimated as the time cost of one epoch of local training; T_k^{RPup} is computed in a similar way to the calculation of T_k^{comm} in Eq. (5.28) (where the uplink bandwidth is set as one half of the total bw_k); $RPsize$ is the size of a profile, which is equal to $4 \times 2 \times q = 8 \times q$ (four bytes for each floating point number) according to the definition of profile in (5.20).

This work models the energy consumption of the end devices in FL by mainly considering the energy consumption of the transmitters for communication (Eq. 5.31) and on-device computation for local training (Eq. 5.32). For the proposed algorithm, there is an extra energy cost for generating and uploading profiles (Eq. 5.33).

$$E_k^{comm} = P_{trans} \cdot T_k^{comm} \tag{5.31}$$

$$E_k^{train} = P_f s_k^3 \cdot T_k^{train} \tag{5.32}$$

$$E_k^{RP} = P_{trans} \cdot T_k^{RPup} + P_f s_k^3 \cdot T_k^{RPgen}, \tag{5.33}$$

where $P_f s_k^3$ is a simplified computation power consumption model [299] and P_f is the power of a baseline processor. P_{trans} is the transmitter's power. The experiment sets P_{trans} and P_f to 0.5 and 0.7 Watts respectively based on the benchmarking data provided in [281]. Thus the energy consumed by client k to participate in one round of FL is formulated as (E_k^{RP} only applies to the proposed algorithm):

$$E_k = E_k^{comm} + E_k^{train} + E_k^{RP}. \tag{5.34}$$

5.6.2 Evaluation Results

The performance of fedProf algorithm was evaluated in terms of the effectiveness and efficiency in establishing a global model for the two tasks. Results for FedAvg and five other state-of-the-art algorithms are incorporated for comparison (see Table 5.1). The effectiveness of a algorithm is measured by the best model accuracy that the algorithm can achieve, whilst the efficiency is evaluated by setting an accuracy goal for the convergence of the global model and is assessed from multiple perspectives including the number of rounds needed, the amount of time required and the device energy consumed for achieving the target accuracy. Tables 5.3 and 5.4 summarise the results. Figs. 5.7, 5.8, 5.9 and 5.10 plot the traces of the accuracy in the round-wise evaluation of the global model.

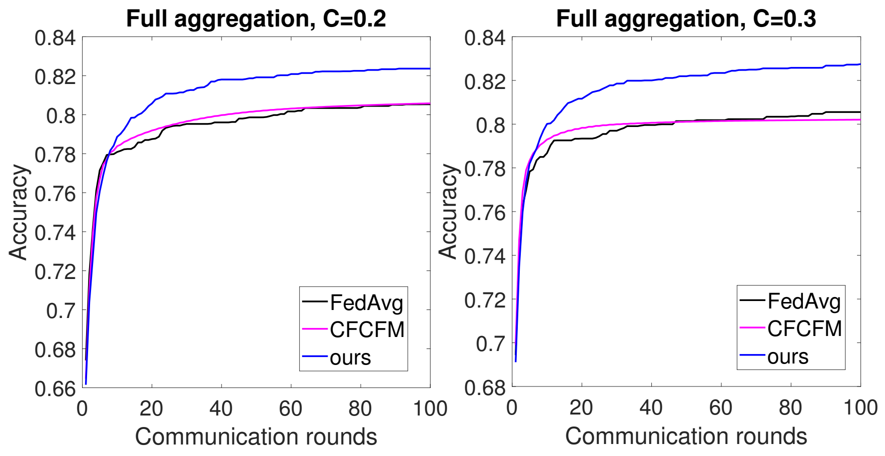


Figure 5.7: The traces of evaluation accuracy of the global model through 100 rounds using the full-aggregation algorithms in the S-Task.

1) Convergence in different aggregation modes: First evaluated is algorithm performance by comparing the convergence of the global model using different aggregation methods. From Figs. 5.7, 5.8, 5.9 and 5.10, one can observe that partial aggregation facilitates faster convergence of the global model than full aggregation, which

Table 5.3: The results of running the S-Task; the FL process for the S-Task is run in two ways: i) running for t_{max} rounds (plotting the best accuracy achieved), and ii) running the process until the global model reaches the targeted accuracy (0.8 for the S-Task). The tested algorithms are grouped by aggregation method.

		Full aggregation		
		C=0.2		
	Best accuracy	For accuracy@0.8		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg	0.805	56	2869.66	2.87
CFCFM	0.806	39	1230.81	1.61
Ours	**0.824**	**16**	**803.74**	**0.80**
		C=0.3		
	Best accuracy	For accuracy@0.8		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg	0.806	52	3160.01	4.12
CFCFM	0.802	42	1495.34	2.91
Ours	**0.827**	**12**	**701.18**	**0.94**
		Partial aggregation		
		C=0.2		
	Best accuracy	For accuracy@0.8		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg-RP	0.819	13	735.48	0.71
FedProx	0.821	16	899.93	0.79
FedAdam	0.818	8	438.20	0.42
AFL	0.816	6	313.81	0.30
Ours	**0.844**	**5**	**283.76**	**0.27**
		C=0.3		
	Best accuracy	For accuracy@0.8		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg-RP	0.817	8	466.90	0.64
FedProx	0.810	16	841.65	1.08
FedAdam	0.819	12	667.00	0.94
AFL	0.813	6	298.81	0.42
Ours	**0.841**	**4**	**235.22**	**0.35**

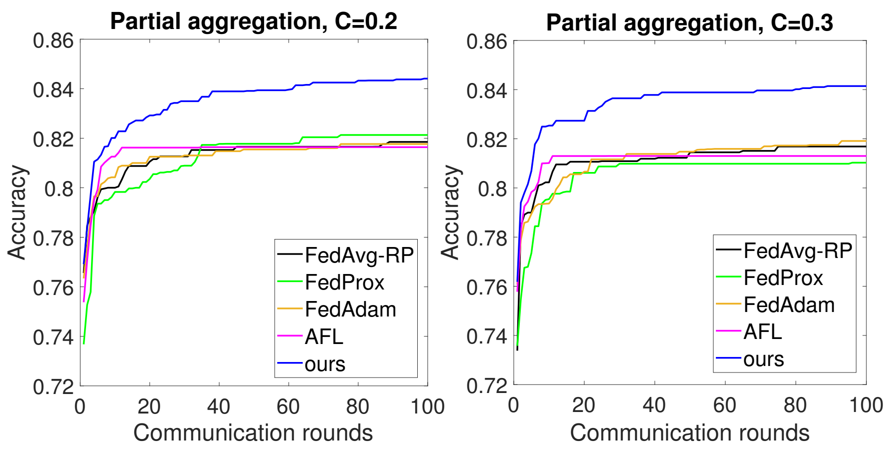


Figure 5.8: The traces of evaluation accuracy of the global model through 100 rounds using the partial-aggregation algorithms in the S-Task.

Table 5.4: The results of the L-Task; the FL process for the L-Task is run in two ways: i) running for t_{max} rounds (plotting the best accuracy achieved), and ii) running the process until the global model reaches the targeted accuracy (0.9 for the L-Task). The algorithms are grouped by aggregation method.

Full aggregation C=0.05				
	Best accuracy	For accuracy@0.9		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg	0.906	213	10407.30	60.01
CFCFM	0.923	251	8645.17	61.37
Ours	**0.926**	**98**	**4846.15**	**28.22**
Full aggregation C=0.1				
	Best accuracy	For accuracy@0.9		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg	0.929	76	3894.26	43.16
CFCFM	0.932	75	2675.56	37.62
Ours	**0.945**	**45**	**2295.03**	**25.98**
Partial aggregation C=0.05				
	Best accuracy	For accuracy@0.9		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg-RP	0.937	12	572.90	3.29
FedProx	0.936	13	640.01	3.58
FedAdam	0.940	12	599.96	3.47
AFL	0.952	10	479.73	2.73
Ours	**0.962**	**8**	**383.94**	**2.26**
Partial aggregation C=0.1				
	Best accuracy	For accuracy@0.9		
		Rounds needed	Total time (s)	Energy (Wh)
FedAvg-RP	0.938	12	603.94	6.57
FedProx	0.942	11	559.16	5.91
FedAdam	0.939	12	608.85	6.76
AFL	0.944	9	476.62	5.26
Ours	**0.962**	**8**	**413.16**	**4.64**

is consistent with the observations by [85]. The difference in convergence speed is especially obvious in the L-Task where partial aggregation requires significantly fewer communication rounds to reach 90% accuracy. FedProf algorithm yields the fastest convergence in both groups of comparison for both tasks because limiting the contribution from clients with low-quality data benefits the global model for both aggregation methods.

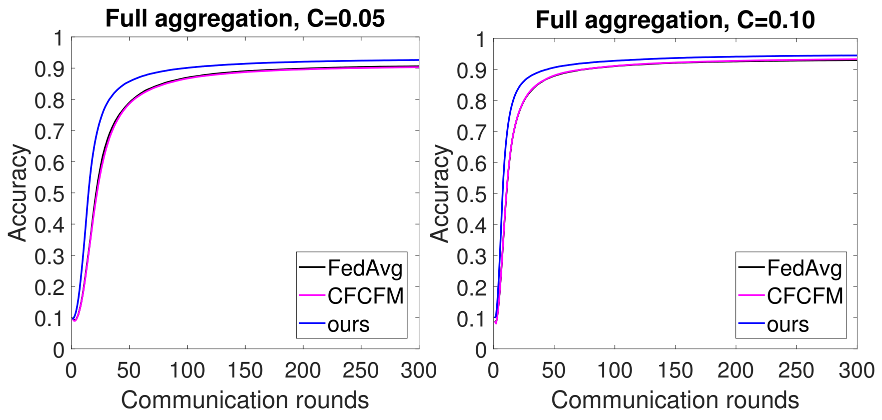


Figure 5.9: The traces of evaluation accuracy of the global model through 300 rounds using the full-aggregation algorithms in the L-Task.

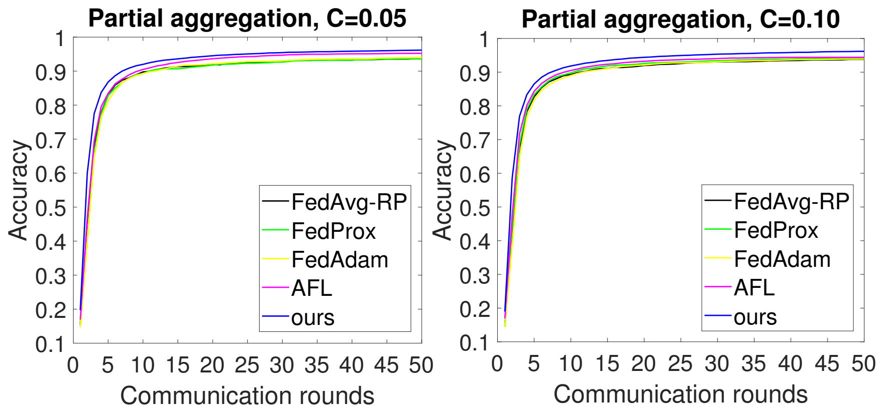


Figure 5.10: The traces of evaluation accuracy of the global model through 50 rounds using the partial aggregation algorithms in the L-Task.

2) Best accuracy of the global model: Through the federated learning process, the global model is evaluated each round on the server. The best global model obtained thus far is kept on the server. In case where the global model obtained in the current round is inferior to the best one, the inferior model is discarded. The second column of Tables 5.3 and 5.4 compares the best accuracy the global model can reach given a sufficient number of rounds in training. FedProf algorithm achieves up to 2.5% accuracy improvement when compared against the baselines (FedAvg and FedAvg-

RP). The AFL algorithm uses a loss-oriented client selection strategy, which shows the closest performance to the proposed algorithm in the L-Task but the worst accuracy in the S-Task.

3) Total communication rounds for convergence: The number of communication rounds required for convergence is a key indicator to the efficiency of FL. The presented experiment recorded this metric by setting an accuracy objective (0.8 and 0.9 for the S-Task and L-Task, respectively) for the algorithms in the training process to compare their efficiency, as summarised in Tables 5.3 and 5.4. In the S-Task, the proposed algorithm takes less than half the communication rounds required by other algorithms in most cases. In the L-Task, the global model converges slowly using full aggregation with C=0.05; The proposed algorithm reaches 90% accuracy within 100 rounds whilst FedAvg and CFCFM need more than 200. In this case, fedProf also achieves approximately 7% higher accuracy at round 50 compared to FedAvg and CFCFM. Partial aggregation algorithms turn out to be much more efficient in terms of convergence, which also implies significant less communication costs for the global model to converge. FedAvg-RP needs at least 8 rounds to reach the accuracy target for the S-Task and 12 rounds for the L-Task, whilst the proposed algorithm reduces the numbers to 5 and 8, respectively.

4) Total time needed for convergence: The overall time consumption is tightly coupled with the number of communication rounds for convergence, but it also reflects the actual time cost for each round of training (which includes local training and model transmission). The fourth column of Tables 5.3 and 5.4 gives the total time consumed by each algorithm to achieve the preset accuracy of the global model. One can see that algorithms requiring more rounds to converge typically take longer to reach the accuracy target except the case of CFCFM, which priorities the clients that work faster. Using FedAvg as the baseline, CFCFM accelerates the training process by 2.1× whist the proposed algorithm provides a 4.5× speedup in terms of achieving the target accuracy (S-Task, C=0.3). fedProf also has a clear advantage over FedAvg-RP, FedProx and FedAdam in the partial aggregation group where it shows a 2.6× speedup over FedAvg-RP in the S-Task with C=0.2.

5) Device-side energy consumption: A main concern for the end devices, as the participants of FL, is their power usage in both local training and communication. The analysis is presented from a holistic perspective by evaluating the total energy consumption (in Watt hours) across all of the clients. The results are summarised in Tables 5.3 and 5.4. It is obvious that device-side energy consumption is closely associated to the total number of communication rounds and the total time span until the convergence. Algorithms using the full aggregation method have slower convergence and thus generate higher energy cost on the devices. For example, with a small selection fraction C=0.05 in the L-Task, FedAvg and CFCFM both consumed over 60 Wh to reach the accuracy mark. In this case, the proposed algorithm manages to reduce the energy consumption by more than a half (28.22 Wh). With the partial aggregation mode, the total energy consumption is cut down by up to 62% using the proposed algorithm (S-Task, $C = 0.2$). Considering all the cases, the proposed algorithm manages to achieve the target accuracy with the least cost, resulting in a reduction of device energy consumption by 29% $\sim$ 53%.

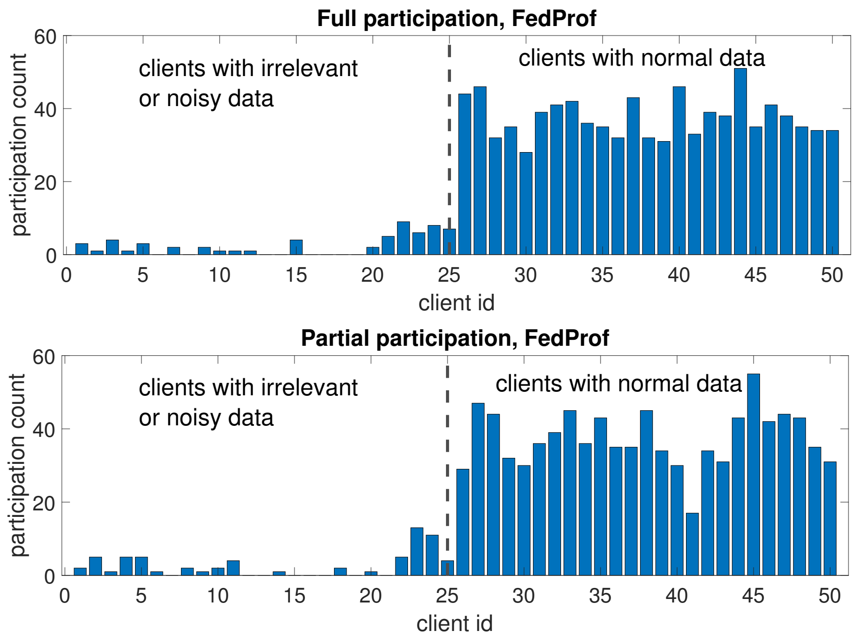


Figure 5.11: Total counts by client of participation (i.e., being selected) in S-Task. For clarity, clients are indexed according to their local data quality.

6) Differentiated participation with FedProf: The proposed FedProf dynamically adjusts each client's probability of being selected. In this way some clients participate at a largely reduced rate if the algorithm reports a high divergence between their local representation profiles and the baseline profile, which means that their local data are very likely to be low-quality. Fig. 5.11 reflects the preference of the proposed selection strategy in S-Task with C=0.2; one can observe that clients with useless samples or noisy data get significantly less involved (<10 on average). Fig. 5.12 gives

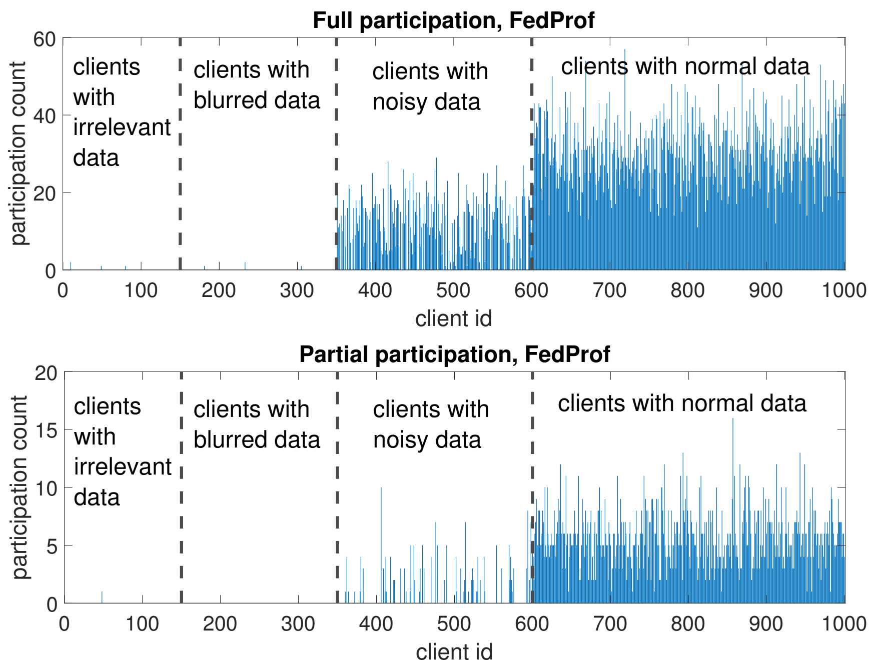


Figure 5.12: Total counts by client of participation (i.e., being selected) in L-Task. For clarity, clients are indexed according to their local data quality.

the participation counts by client in the L-Task. In this task, the proposed algorithm also effectively limits (basically excludes) the clients who are training on image data of poor quality (i.e., irrelevant or severely blurred), whereas the clients with noisy images (blended with moderate salt-and-pepper noise) are selected with a reduced probability as compared to those with normal data. A potential issue of having the preference towards some of the devices is about fairness. Nonetheless, one can apply the proposed algorithm together with an incentive mechanism (e.g., [77]) to address the issue.

5.7 Conclusion

Federated learning provides a privacy-preserving approach to decentralised training but the low-quality data on end devices could significantly affect the convergence and the quality of the global model that the system aims to build. In this work, a novel FL algorithm FEDPROF is proposed to address the issue without violating the data locality restriction of FL. The basic idea is to discriminate clients by their local data and the key designs of the proposed algorithm are i) a data representation profiling and matching protocols for encoding the hidden data representation with the global model and computing dissimilarities between the local profiles from devices and the baseline profiles on the server, and ii) a client selection strategy that adaptively adjusts clients' participating chance based on profile dissimilarities. Results of comprehensive experiments under various FL settings show that the proposed algorithm significantly improves the global model's quality and reduces the costs for the global model's convergence.

The future plan of study will be focused on methods for optimising local training by means of acquiring more comprehensive knowledge on local data distribution.

Chapter 6

Conclusions and Future Work

This thesis presents a comprehensive study of the emerging federated learning framework with focuses on the approaches to efficient horizontal FL. The problem background and research motivations are first introduced in support of the necessity of this research. Three novel solutions to the efficiency optimisation of horizontal FL are then presented with extensive experimental evaluation results.

This chapter concludes the whole thesis by summarising the key content of the work presented with the emphasis on the research contributions made. Open challenges in the development of realistic FL systems are then discussed. Finally, light will be cast on several prospective research directions to provide an outlook for future study.

6.1 Summary of the Thesis

The first part of the thesis introduces the background of the emerging FL framework, elaborates the research motivation and defines the efficiency metrics for FL. The second chapter formulates the federated optimisation problem and gives important preliminaries including a theoretical analysis that reveals the impacts of 'data islands' on distributed gradient-based training and the core value of FL (which is listed as one of the key conclusions at the end of this section). Chapter 2 presents a comprehensive literature review that basically covers the entire research spectrum of FL as well as a survey on open platforms, developed systems and representative applications. As the body of this thesis, Chapters 3, 4 and 5 correspond to three threads of research by the author, where each of the three chapters presents one novel approach to the optimisation of horizontal FL in terms of efficiency.

The **main contributions** (and the **key conclusions** associated) are summarised as follows:

1. A theoretical analysis (Section 2.1.2) on the consequences of the 'data islands' problem for gradient-based training. The analysis provides case-by-case results of the gradient divergence caused by training on decentralised data. Based on the analytical results, two key conclusions are drawn:

   - **Over non-IID data across the clients, distributed training using SGD cannot guarantee an unbiased estimate of the centralised model trained on centralised data even with the gradients uploaded and aggregated every update step.**
   - **FL is essentially a multi-step SGD paradigm with more flexibility in the number of steps, batch size as well as the optimisation algorithm for local update. The key idea behind FL is relaxing the gradient divergence in local training in exchange for significantly improved communication efficiency.**

2. A semi-asynchronous FL algorithm SAFA to address the problems in federated learning such as low round efficiency and poor convergence rate over unreliable devices. Novel designs are introduced in the steps of model distribution, client selection and global aggregation to mitigate the impacts of stragglers, crashes and model staleness in order to boost efficiency and improve the quality of the global model. Extensive experiments were conducted with typical machine learning tasks. The results demonstrate that the proposed algorithm is effective in terms of shortening federated round duration, reducing local resource wastage and improving the accuracy of the global model at an acceptable communication cost. This research also leads to an important insight:

   - **Beneath the design of any synchronous or asynchronous FL algorithms is the trade-off between convergence speed and communication cost. Although researchers keep adding tricks to the bag for better convergence guarantees, the core value of FL as a *communication-efficient* machine learning framework should always be considered as the primary principle in algorithm design.**

3. A multi-layer federated learning algorithm, namely HybridFL, designed for the

Mobile Edge Computing architecture. HybridFL adopts two levels (the edge level and the cloud level) of model aggregation enacting different aggregation strategies to enable scalable FL at the network edge. In order to mitigate stragglers and end device drop-out, controlling variables called regional slack factors are introduced into the stage of client selection performed on the edge nodes using a probabilistic approach without identifying or probing the state of end devices (whose reliability is unknown). The effectiveness of the proposed solution is demonstrated and convergence analysis is also provided. Results of extensive experiments with machine learning tasks at multiple system scales show that HybridFL improves the FL training process significantly in terms of shortening the federated round length, speeding up the global model's convergence (by up to 12×) and reducing end device energy consumption (by up to 58%). Through reviewing the relevant studies and the observations obtained in the research, the following conclusion is made:

- **The pervasive computing power and data at the network edge provide a natural greenhouse for the applications of federated learning, yet most of the existing approaches are constrained by the canonical two-layer system architecture and suffer poor scalability in the MEC environment. The presented work shows the possibilities of efficiency improvement for FL from the architectural prospective.**

4. A novel algorithm named FedProf for optimising the efficiency and efficacy of FL under the circumstances where a large proportion of the clients are probably in possession of low-quality data that are biased, noisy or even irrelevant. Model updates from these low-value clients could significantly slow down the convergence of the global model and also compromise its quality. The key of the proposed approach is a data representation profiling and matching protocol that uses the global model to dynamically profile latent data representations, which allows for low-cost, lightweight representation profile matching. By matching local profiles from clients against a baseline profile on the server, client participation probabilities are dynamically adjusted so as to mitigate the impact of the clients with low-quality data on the training process. Results of extensive experiments on public datasets using various FL settings demonstrates the effectiveness of

FedProf in reducing the number of communication rounds and overall time for convergence and improving the accuracy of the final global model. The outcome of this research inspires the conclusion below:

- **Federated learning maximises its utility when it comes to deep models and difficult learning tasks. The latent representations from the model provide useful information for optimising the FL process given a colossal volume of data (though they are decentralised) and the complexity of intrinsic data structure (e.g., images and sentences). The presented work finds a link between the representation distribution and the training value of local data.**

6.2 Challenges for Industrial Practice

A relatively comprehensive survey is provided in Chapter 2 (Section 2.2) for the development and deployment of federated learning in the industry. Conceptually, the federated learning framework opens a door to the development of data-driven AI applications that rely on sensitive data. The broad technical spectrum of FL provides great opportunities for research but also poses vast engineering challenges to the design and development of any realistic FL system. Rapid advances have been made in the research centred around FL. However, it is also true that *more efforts need to be paid to narrow the gap between what FL can provide in the papers and what FL can actually bring to our industry and society in the real world.*

To benefit from research outcomes and push FL to a stage of maturity for commercial applications, a number of open challenges stand out as follows.

Algorithm Reproduction

The advantages of the open platforms such as `FATE` and `FedML` are drawing increasing attention from both researchers and developers. They usually provide a suite of libraries (typically python-based) that support the functionality of an FL system. Many of them also incorporate preset system configurations, pre-defined models and pre-partitioned datasets.

On the one hand, these out-of-the-box features well facilitate agile development of prototype systems and applications based on the generic FL logic. On the other hand, by constantly adding new modules and new features, these platforms are making themselves increasingly bulky and complicated, which inevitably brings about problems including bugs, redundant functions, poor backward compatibility and over-encapsulation. For algorithm reproduction, users have to deal with system-related, data-related and model-related configurations and spend much time on figuring out which APIs are useful and how they work. This can turn out to be a disadvantage of these platforms for researchers who prioritise fast algorithm reproduction and validation — a self-built simulation can be much handier in some cases. This also in some ways explains why most of the existing research work was not carried out on open FL platforms. For example, the proposed SAFA algorithm requires model versioning, crash detection/round timer and flexible control over the synchronisation process, whilst FedProf demands extraction of data representations (as intermediate results of model inference), statistical compression and non-model information exchange. These algorithmic mechanisms are non-generic (e.g., representation extraction is intrusive) and thus hard to be realised using the basic APIs provided by these platforms.

Through investigating the existing libraries and platforms, one can recognise the trade-off between flexibility and usability in the design of APIs. For platform and system developers, it is extremely challenging to devise a fine hierarchy of APIs that cover high-level abstractions (e.g., for application developers who want low-code implementation) and low-level fundamental operations (e.g., for researchers who want flexibility) at the same time. In spite of that, it is also the users' responsibility to understand their needs (industry oriented or research oriented) before making their choice.

The Necessity of FL

The core value of FL rests on the federation of data, i.e., the performance gain (for the model to build) from the knowledge provided by a multitude of datasets. The advantage of FL over independent training (i.e., training with local data only) is obvious in most of the cases where each local dataset is hardly big enough for the learning task — this is the basic assumption made by the majority of FL research though it may not be mentioned explicitly. However, it should not be ignored that there are also situations where the adoption of FL is (almost) pointless. First, the difficulty of the learning tasks matters a lot. When every local dataset is large and informative enough to support independent training and yield a strong model (though it could be little bit stronger if trained collaboratively), federated training turns out to be unnecessary as the performance gain can hardly make up the cost (unless every 0.01% accuracy improvement is vital). On the other hand, the efficacy of FL also diminishes over extremely non-IID data, which means that local datasets exhibit a strong discrepancy in data distribution and could even be contradictory. In this case, the global model in FL may work poorly on any client and thus turns out to be valueless from the client's standpoint.

Therefore, it is very necessary to evaluate whether FL is really needed over the 'local only' solution especially when the finalised global model is expected to benefit the clients realistically.

Device Heterogeneity

Devices are more heterogeneous than they are assumed in the papers. Taking mobile phones as an example, they are different in models, OS and OS versions, hardware specifications and battery levels whilst they also differ in active hours, charging periods and network accessibility and bandwidth due to the different behaviour patterns of the device owners. Device heterogeneity poses a great challenge to the development of FL systems. From the algorithmic standpoint, studies (including the work presented in this thesis) have shown that the degree of device heterogeneity has a strong impact on the convergence of the global model [137, 139]. From the engineering standpoint, it definitely puts a heavy burden on the client-side application development to be compatible to a diversity of device models, multiple ML frameworks and different operating systems.

Certain mechanisms are also needed to ensure the model exchange is really secure on the channel and the local training does not overload user devices.

Privacy and Security

As figured out in the literature [49, 255, 256], privacy and security always need to be concerned in any realistic FL system. Although privacy protection features have been incorporated in the majority of the open FL platforms (as discussed in Section 2.2.1), the cost of these techniques for secure multi-party computation (SMC) is never negligible. For example, homomorphic encryption (HE) typically involves a significant amount of extra computation when doing the calculation in the ciphertext space. Artificial noise-based methods like Differential Privacy (DP) incurs extra communication between the client and the server and usually require the participation fraction to be big enough for noise neutralisation. The adoption of privacy-preserving schemes are usually unavoidable, hence the computation cost should be taken into account as well as their potential impact on the model performance.

In addition, recent research has revealed FL's vulnerability to adversarial clients [283] which attempt to manipulate the global model with data poisoning [155] or model poisoning [215] and dishonest clients who intend to breach the privacy hedge by means of data reconstruction [259, 262] or membership inference [91, 121]. Therefore, the risk of being attacked by potentially malicious clients needs to be considered in the design and maintenance of public FL systems.

Incentive Mechanisms

Most of the existing FL algorithms in the literature rely on an ideal collaboration of honest and selfless clients. 'Honesty' refers to the intention of the participants, as discussed in the previous *Privacy and Security* topic, whilst 'selflessness' implies that the clients totally follow the server's coordination to perform local training regardless of its own gain. Simply said, clients are assumed to be free labours. But this is apparently unrealistic in many application scenarios where the devices are more often than not self-interested. On this point, the way how FL needs to be organised is similar to crowdsourcing where participants are recruited and they get rewarded for their contribution. The design of incentive mechanisms can be helpful for scale-sensitive FL applications in such situations [78, 78, 217].

While incentives can attract more devices to participate, a certain reputation mechanism can make the FL system more resilient to 'unwanted' clients. Also termed Byzantine machines in the literature, some individual clients tend to be stragglers or exhibit suspicious intentions. These low-quality clients are very likely to slow down the overall training process or increase the risk of model manipulation. To address the problem, certain mechanisms such as contribution measurement (e.g., [219, 300]) and reputation management (e.g., [179]) are indispensable for FL systems deployed in a potentially unfriendly environment.

6.3 Future Work

Whilst federated learning provides a promising roadmap to learning from geographically decentralised data without compromising the privacy of data owners, it remains a big challenge when it comes to optimising the efficiency of horizontal FL considering the complexity of distributed systems (especially at scale), the diversity and difficulty of machine learning tasks and the range of associated problems both technically and societally. *The research presented in this thesis provides algorithmic approaches towards efficient horizontal FL, yet there is still a lot to explore in this direction.* Future research work by the author is expected to cover a number of different aspects as below.

Federated loss functions for addressing or mitigating local data bias and imbalance. Loss function design is vital to the performance of any machine learning task that adopts gradient-based training methods. A loss function serves as the objective of model training and determines the optimisation direction in the model parameter space given a set of data. The choice of loss function directly affects the convergence property [4]. However, the role of loss function is overlooked in the majority of researches on FL even though the improvement of loss design (e.g., Focal Loss [301] and CBLoss [286]) has been proven very effective in standalone training. Considering the potentially strong data heterogeneity in many FL applications, it is of great necessity to study how to improve the performance of FL (in convergence rate and global model accuracy) by means of loss adaptation for more effective local training. A few steps have been taken by previous studies on personalised FL, where the common trick is adding a proximal term to the local loss function so as to reduce global–local model divergence and produce accurate local models. As an alternative, data-aware or data-sensitive loss adaptation may be a better option for achieving a stronger global consensus model; how to design such loss functions over the 'data islands' leaves a lot to desire.

A comprehensive investigation of the utility of FL and the contribution measures for clients under varied distribution of data. The framework of FL (especially horizontal FL) is so advocated that the actual gains from using federated learning rather than local-only training are hardly investigated. It is reasonable to raise the doubt because running hundreds of rounds of FL is not a cost-friendly solution in most scenarios and thus it is definitely necessary to evaluate if it is worth the cost. The utility of FL can be gauged through comparison between the final global model and a baseline model from purely local training or non-collaborative training with almost no communication. For example, one or a few models solely trained by the clients or an aggregated model via 'one-shot averaging' [229] can be used as the baseline. It is possible that, under certain data distributions (e.g., almost IID and with sufficient local data), the gains of FL tend to be marginal. Another intriguing line of research is on how to measure the contributions of participating clients. Existing solutions based on data deletion and the Shapley value [219, 300] are fine-grained but extremely costly. Nonetheless, improvements have been made by taking advantage of the progressive process of FL [97]. Motivated by theses studies and considering the importance of profit allocation in realistic FL systems, lightweight contribution measuring schemes will be a focus in the future.

Developing more flexible FL frameworks, training algorithms and communication protocols for generalised network topologies. The prevalence of federated learning is in a large part credited to the prevailing use of the client–server communication model. The star topology is communication-efficient and easy to manage from the central server's perspective. Indeed, this centralised topology is the natural choice for horizontal FL because it provides consistency with the least communication cost. Despite the fact, there are still some situations where the network topology does not have an eligible central node. A cross-silo scenario can be an example of such situation where multiple organisations want to collaborate via FL but none of them has the full trust to become an aggregator. Some research efforts have been taken towards federated learning in a Peer-to-Peer (P2P) fashion [59, 243] where nodes in the network communicate with each other without a central coordinator. The Gossip protocol is a common choice in such topologies whilst more possibilities, e.g., duplication, segmentation [58] and grouping can be explored on top of that.

Though theoretical guarantee is provided by some studies [58, 60, 118], it is still unclear how much the heterogeneity impacts the model performance on different workers. Heterogeneous connectivity and adjacency between the nodes also need to be considered in the future work for fully decentralised FL algorithms.

Bibliography

[1] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A Jarvis. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70(5):655–668, 2021.

[2] Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1539–1551, 2021.

[3] Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Fedprof: Selective federated learning with representation profiling. *arXiv preprint arXiv:2102.01733. Under review, ICLR'22*, 2021.

[4] Wentai Wu, Ligang He, and Weiwei Lin. Variation-incentive loss re-weighting for regression analysis on biased data. *arXiv preprint arXiv:2109.06565.*, 2021.

[5] Wentai Wu, Ligang He, Weiwei Lin, Yi Su, Yuhua Cui, Carsten Maple, and Stephen A Jarvis. Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality. *IEEE Transactions on Knowledge and Data Engineering. DOI: 10.1109/TKDE.2020.3035685*, 2020.

[6] Wentai Wu, Weiwei Lin, Ligang He, Guangxin Wu, and Ching-Hsien Hsu. A power consumption model for cloud servers based on elman neural network. *IEEE Transactions on Cloud Computing*, 9(4):1268–1277, 2021.

[7] Weiwei Lin, Wentai Wu, and Ligang He. An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers. *IEEE Transactions on Services Computing*, 2019.

[8] Tiansheng Huang, Weiwei Lin, Wentai Wu, Ligang He, Keqin Li, and Albert Zomaya. An efficiency-boosting client selection scheme for federated learning

with fairness guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[9] Weiwei Lin, Fang Shi, Wentai Wu, Keqin Li, Guangxin Wu, and Al-Alas Mohammed. A taxonomy and survey of power models and power modeling for cloud servers. *ACM Computing Surveys (CSUR)*, 53(5):1–41, 2020.

[10] Weiwei Lin, Wentai Wu, and Keqin Li. Energy-saving technologies of servers in data centers. In Hwaiyu Geng, editor, *Data Center Handbook: Plan, Design, Build, and Operations of a Smart Data Center*, chapter 19, pages 337–347. John Wiley & Sons, New Jersey, 2021.

[11] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[12] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *the 2nd SysML Conference*, 2019.

[13] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. Fedgraphnn: A federated learning system and benchmark for graph neural networks, 2021.

[14] Chaoyang He, Alay Dilipbhai Shah, Zhenheng Tang, Di Fan, Adarshan Naiynar Sivashunmugam, Keerti Bhogaraju, Mita Shimpi, Li Shen, Xiaowen Chu, Mahdi Soltanolkotabi, et al. Fedcv: A federated learning framework for diverse computer vision tasks. https://fedml.ai/files/FedCV.pdf, 2021. Accessed: 2021-08-15.

[15] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[16] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[17] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[18] Deepak Ravichandran and Sergei Vassilvitskii. Evaluation of cohort algorithms for the floc api. Whitepaper, Google Research & Ads, 2020. URL `https://github.com/google/ads-privacy/raw/master/proposals/FLoC/FLOC-Whitepaper-Google.pdf`.

[19] Micah J Sheller, Brandon Edwards, G Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R Colen, et al. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports*, 10(1):1–12, 2020.

[20] Yi Liu, JQ James, Jiawen Kang, Dusit Niyato, and Shuyu Zhang. Privacy-preserving traffic flow prediction: A federated learning approach. *IEEE Internet of Things Journal*, 7(8):7751–7763, 2020.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[22] John McCarthy. From here to human-level ai. *Artificial Intelligence*, 171(18): 1174–1182, 2007.

[23] Edward A Feigenbaum. Knowledge processing: From fileservers to knowledge servers. *Applications of Expert Systems*, 2:3–11, 1988.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[25] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[29] Ivan Habernal, Omnia Zayed, and Iryna Gurevych. C4corpus: Multilingual web-size corpus with free license. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 914–922, 2016.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[31] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[32] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[33] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. *arXiv preprint arXiv:1609.08976*, 2016.

[34] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[35] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[36] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. In *Proceedings of the*

2013 Conference on Empirical Methods in Natural Language Processing, pages 1044–1054, 2013.

[37] Yingwei Pan, Tao Mei, Ting Yao, Houqiang Li, and Yong Rui. Jointly modeling embedding and translation to bridge video and language. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4594–4602, 2016.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Jiaxiang Wu, Weidong Huang, Junzhou Huang, and Tong Zhang. Error compensated quantized sgd and its applications to large-scale distributed optimization. In *International Conference on Machine Learning*, pages 5325–5333. PMLR, 2018.

[40] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[41] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720, 2017.

[42] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

[43] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. Workshop track, ICLR 2016, 2016.

[44] Robert van der Meulen. What edge computing means for infrastructure and operations leaders. https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/, 2018. Accessed: 2021-07-10.

[45] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. *arXiv preprint arXiv:2104.07586*, 2021.

[46] Ligeng Zhu and Song Han. Deep leakage from gradients. In *33rd Conference on Neural Information Processing Systems (NeurIPS'19)*, 2019.

[47] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282. PMLR, 2017.

[48] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[49] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[50] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[51] Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 685–693, 2015.

[52] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, pages 1223–1231, 2012.

[53] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[54] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[55] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[56] Farzin Haddadpour and Mehrdad Mahdavi. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*, 2019.

[57] Rudrajit Das, Anish Acharya, Abolfazl Hashemi, Sujay Sanghavi, Inderjit S Dhillon, and Ufuk Topcu. Faster non-convex federated learning via global and local momentum. *arXiv preprint arXiv:2012.04061*, 2020.

[58] Chenghao Hu, Jingyan Jiang, and Zhi Wang. Decentralized federated learning: A segmented gossip approach. *arXiv preprint arXiv:1908.07782*, 2019.

[59] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.

[60] Anusha Lalitha, Shubhanshu Shekhar, Tara Javidi, and Farinaz Koushanfar. Fully decentralized federated learning. In *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[61] John Duchi, Michael I Jordan, and Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. *Advances in Neural Information Processing Systems*, 26:2832–2840, 2013.

[62] Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, pages 456–464, 2010.

[63] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008. PMLR, 2014.

[64] Alessandro Rigazzi. Dc-s3gd: Delay-compensated stale-synchronous sgd for large-scale decentralized neural network training. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pages 62–68. IEEE, 2019.

[65] Gideon Mann, Ryan T McDonald, Mehryar Mohri, Nathan Silberman, and Dan Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*, 2009.

[66] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

[67] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Asaga: asynchronous parallel saga. In *Artificial Intelligence and Statistics*, pages 46–54. PMLR, 2017.

[68] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

[69] John Langford, Alexander Smola, and Martin Zinkevich. Slow learners are fast. *arXiv preprint arXiv:0911.0491*, 2009.

[70] Junyu Li, Ligang He, Shenyuan Ren, and Rui Mao. Developing a loss prediction-based asynchronous stochastic gradient descent algorithm for distributed training of deep neural networks. In *49th International Conference on Parallel Processing-ICPP*, pages 1–10, 2020.

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

[72] Michael R Sprague, Amir Jalalirad, Marco Scavuzzo, Catalin Capota, Moritz Neun, Lyman Do, and Michael Kopp. Asynchronous federated learning for geospatial applications. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 21–28. Springer, 2018.

[73] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 28:2737–2745, 2015.

[74] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International Conference on Artificial Intelligence and Statistics*, pages 803–812. PMLR, 2018.

[75] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. *arXiv preprint arXiv:1511.05950*, 2015.

[76] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4427–4437, 2017.

[77] Han Yu, Zelei Liu, Yang Liu, Tianjian Chen, Mingshu Cong, Xi Weng, Dusit Niyato, and Qiang Yang. A fairness-aware incentive scheme for federated learning. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 393–399, 2020.

[78] Jiawen Kang, Zehui Xiong, Dusit Niyato, Han Yu, Ying-Chang Liang, and Dong In Kim. Incentive design for efficient federated learning in mobile networks: A contract theory approach. In *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, pages 1–5. IEEE, 2019.

[79] Jiawen Kang, Zehui Xiong, Dusit Niyato, Shengli Xie, and Junshan Zhang. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal*, 6(6):10700–10714, 2019.

[80] Laizhong Cui, Xiaoxin Su, Yipeng Zhou, and Yi Pan. Slashing communication traffic in federated learning by transmitting clustered model updates. *IEEE Journal on Selected Areas in Communications*, 2021.

[81] Jakub Konečný, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[82] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. Slowmo: Improving communication-efficient distributed sgd with slow momentum. In *International Conference on Learning Representations*, 2020.

[83] Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. In *33rd Conference on Uncertainty in Artificial Intelligence (UAI 2017)*, pages 675–684. Curran Associates, Inc., 2017.

[84] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Communication-efficient algorithms for statistical optimization. *The Journal of Machine Learning Research*, 14(1):3321–3363, 2013.

[85] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

[86] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *arXiv preprint arXiv:1809.07599*, 2018.

[87] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700, 2019.

[88] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *The 3rd MLSys Conference*, 2020.

[89] Zhenguo Ma, Yang Xu, Hongli Xu, Zeyu Meng, Liusheng Huang, and Yinxing Xue. Adaptive batch size for federated learning in resource-constrained edge computing. *IEEE Transactions on Mobile Computing*, 2021.

[90] Viraaji Mothukuri, Prachi Khare, Reza M Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava. Federated learning-based anomaly detection for iot security attacks. *IEEE Internet of Things Journal*, 2021.

[91] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 1–11, 2019.

[92] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. Energy efficient federated learning over wireless communication

networks. *IEEE Transactions on Wireless Communications*, 20(3):1935–1949, 2020.

[93] Fan Ang, Li Chen, Nan Zhao, Yunfei Chen, Weidong Wang, and F Richard Yu. Robust federated learning with noisy communication. *IEEE Transactions on Communications*, 68(6):3452–3464, 2020.

[94] Mohammad Mohammadi Amiri and Deniz Gündüz. Federated learning over wireless fading channels. *IEEE Transactions on Wireless Communications*, 19 (5):3546–3557, 2020.

[95] Nir Shlezinger, Mingzhe Chen, Yonina C Eldar, H Vincent Poor, and Shuguang Cui. Federated learning with quantization constraints. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8851–8855. IEEE, 2020.

[96] Youyang Qu, Shiva Raj Pokhrel, Sahil Garg, Longxiang Gao, and Yong Xiang. A blockchained federated learning framework for cognitive computing in industry 4.0 networks. *IEEE Transactions on Industrial Informatics*, 17(4):2964–2973, 2020.

[97] Tianshu Song, Yongxin Tong, and Shuyue Wei. Profit allocation for federated learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2577–2586. IEEE, 2019.

[98] Nguyen H Tran, Wei Bao, Albert Zomaya, Minh NH Nguyen, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1387–1395. IEEE, 2019.

[99] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*, pages 2350–2358. PMLR, 2021.

[100] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020.

[101] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[102] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020.

[103] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 16(10):6532–6542, 2019.

[104] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.

[105] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10713–10722, 2021.

[106] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

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

[108] Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516*, 2020.

[109] Felix Yu, Ankit Singh Rawat, Aditya Menon, and Sanjiv Kumar. Federated learning with only positive labels. In *International Conference on Machine Learning*, pages 10946–10956. PMLR, 2020.

[110] Nir Shlezinger, Mingzhe Chen, Yonina C Eldar, H Vincent Poor, and Shuguang Cui. Uveqfed: Universal vector quantization for federated learning. *IEEE Transactions on Signal Processing*, 69:500–514, 2020.

[111] Mehdi Salehi Heydar Abad, Emre Ozfatura, Deniz Gunduz, and Ozgur Ercetin. Hierarchical federated learning across heterogeneous cellular networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8866–8870. IEEE, 2020.

[112] Wei Liu, Li Chen, Yunfei Chen, and Wenyi Zhang. Accelerating federated learning via momentum gradient descent. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1754–1766, 2020.

[113] Amirhossein Reisizadeh, Farzan Farnia, Ramtin Pedarsani, and Ali Jadbabaie. Robust federated learning: The case of affine distribution shifts. *arXiv preprint arXiv:2006.08907*, 2020.

[114] Junjie Pang, Yan Huang, Zhenzhen Xie, Qilong Han, and Zhipeng Cai. Realizing the heterogeneity: A self-organized federated learning framework for iot. *IEEE Internet of Things Journal*, 8(5):3088–3098, 2020.

[115] Sabrina Kall and Slim Trabelsi. An asynchronous federated learning approach for a security source code scanner. In *ICISSP*, pages 572–579, 2021.

[116] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019.

[117] Howard H Yang, Zuozhu Liu, Tony QS Quek, and H Vincent Poor. Scheduling policies for federated learning in wireless networks. *IEEE Transactions on Communications*, 68(1):317–333, 2019.

[118] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173*, 2019.

[119] Jinhyun So, Başak Güler, and A Salman Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[120] Mingzhe Chen, H Vincent Poor, Walid Saad, and Shuguang Cui. Convergence time optimization for federated learning over wireless networks. *IEEE Transactions on Wireless Communications*, 20(4):2457–2471, 2020.

[121] Yi Liu, Jialiang Peng, Jiawen Kang, Abdullah M Iliyasu, Dusit Niyato, and Ahmed A Abd El-Latif. A secure federated learning framework for 5g networks. *IEEE Wireless Communications*, 27(4):24–31, 2020.

[122] Wenlin Chen, Samuel Horvath, and Peter Richtarik. Optimal client sampling for federated learning. *arXiv preprint arXiv:2010.13723*, 2020.

[123] WANG Luping, WANG Wei, and LI Bo. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 954–964. IEEE, 2019.

[124] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security*, 15:911–926, 2019.

[125] Ming Xie, Guodong Long, Tao Shen, Tianyi Zhou, Xianzhi Wang, and Jing Jiang. Multi-center federated learning. *arXiv preprint arXiv:2005.01026*, 2020.

[126] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 13–23, 2019.

[127] Sagar Dhakal, Saurav Prakash, Yair Yona, Shilpa Talwar, and Nageen Himayat. Coded federated learning. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2019.

[128] Canh T. Dinh, Nguyen H. Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 33, 2020.

[129] Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

[130] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[131] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12):2663, 2018.

[132] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[133] Mingzhe Chen, Zhaohui Yang, Walid Saad, Changchuan Yin, H Vincent Poor, and Shuguang Cui. A joint learning and communications framework for federated learning over wireless networks. *IEEE Transactions on Wireless Communications*, 20(1):269–283, 2020.

[134] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.

[135] Mingzhe Chen, Nir Shlezinger, H Vincent Poor, Yonina C Eldar, and Shuguang Cui. Communication-efficient federated learning. *Proceedings of the National Academy of Sciences*, 118(17), 2021.

[136] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

[137] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. Towards taming the resource and data heterogeneity in federated learning. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, pages 19–21, 2019.

[138] Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019.

[139] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[140] Zhouyuan Huo, Qian Yang, Bin Gu, Lawrence Carin Huang, et al. Faster on-device training using new federated momentum algorithm. *arXiv preprint arXiv:2002.02090*, 2020.

[141] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Camtepe. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

[142] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1698–1707. IEEE, 2020.

[143] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

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

[145] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations (ICLR)*, 2020.

[146] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. Billion-scale federated learning on mobile clients: a submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[147] Xin Yao, Chaofeng Huang, and Lifeng Sun. Two-stream federated learning: Reduce the communication costs. In *2018 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2018.

[148] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9):3400–3413, 2019.

[149] Kai Yang, Tao Jiang, Yuanming Shi, and Zhi Ding. Federated learning via

over-the-air computation. *IEEE Transactions on Wireless Communications*, 19 (3):2022–2035, 2020.

[150] Hangyu Zhu and Yaochu Jin. Multi-objective evolutionary federated learning. *IEEE transactions on neural networks and learning systems*, 31(4):1310–1322, 2019.

[151] Hung T Nguyen, Vikash Sehwag, Seyyedali Hosseinalipour, Christopher G Brinton, Mung Chiang, and H Vincent Poor. Fast-convergent federated learning. *IEEE Journal on Selected Areas in Communications*, 39(1):201–218, 2020.

[152] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

[153] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[154] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[155] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*, pages 480–501. Springer, 2020.

[156] Wenqi Shi, Sheng Zhou, and Zhisheng Niu. Device scheduling with fast convergence for wireless federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[157] Dipankar Sarkar, Ankur Narang, and Sumit Rai. Fed-focal loss for imbalanced data classification in federated learning. *arXiv preprint arXiv:2011.06283*, 2020.

[158] Dongdong Ye, Rong Yu, Miao Pan, and Zhu Han. Federated learning in vehicular edge computing: A selective model aggregation approach. *IEEE Access*, 8:23920–23935, 2020.

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

[160] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345. IEEE, 2019.

[161] Xin Yao, Tianchi Huang, Chenglei Wu, Ruixiao Zhang, and Lifeng Sun. Towards faster and better federated learning: A feature fusion approach. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 175–179. IEEE, 2019.

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

[163] Rudrajit Das, Anish Acharya, Abolfazl Hashemi, Sujay Sanghavi, Inderjit S Dhillon, and Ufuk Topcu. Faster non-convex federated learning via global and local momentum. *arXiv preprint arXiv:2012.04061*, 2020.

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

[165] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031. PMLR, 2020.

[166] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606*, 2020.

[167] Laizhong Cui, Xiaoxin Su, Yipeng Zhou, and Yi Pan. Slashing communication traffic in federated learning by transmitting clustered model updates. *IEEE Journal on Selected Areas in Communications*, 2021.

[168] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*, 2020.

[169] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 125–136, 2020.

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

[171] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, 33:3557–3568, 2020.

[172] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pages 8253–8265. PMLR, 2020.

[173] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *International Conference on Machine Learning*, pages 3973–3983. PMLR, 2020.

[174] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2020.

[175] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021.

[176] Luca Corinzia, Ami Beuret, and Joachim M Buhmann. Variational federated multi-task learning. *arXiv preprint arXiv:1906.06268*, 2019.

[177] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, 2019.

[178] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[179] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 27(2):72–80, 2020.

[180] Bill Yuchen Lin, Chaoyang He, Zihang Zeng, Hulin Wang, Yufen Huang, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. Fednlp: A research platform for federated learning in natural language processing. *arXiv preprint arXiv:2104.08815*, 2021.

[181] Liu Yang, Ben Tan, Vincent W Zheng, Kai Chen, and Qiang Yang. Federated recommendation systems. In *Federated Learning*, pages 225–239. Springer, 2020.

[182] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley. Mobile keyboard input decoding with finite-state transducers. *arXiv preprint arXiv:1704.03987*, 2017.

[183] Cyril Allauzen and Michael Riley. Bayesian language model interpolation for mobile speech input. In *Interspeech 2011*, pages 1429–1432, 2011.

[184] Chetna Bindra. Building a privacy-first future for web advertising. Google Ads, blog, Jan 2021. URL `https://blog.google/products/ads-commerce/2021-01-privacy-sandbox/`.

[185] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112: 59–67, 2018.

[186] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

[187] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[188] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[189] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13172–13179, 2020.

[190] Jianguo Chen, Kenli Li, Qingying Deng, Keqin Li, and S Yu Philip. Distributed deep learning model for intelligent video surveillance systems with edge computing. *IEEE Transactions on Industrial Informatics*, 2019.

[191] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Federated learning for data privacy preservation in vehicular cyber-physical systems. *IEEE Network*, 34(3):50–56, 2020.

[192] DP Bertsekas, A Nedic, and A Ozdaglar. Convex optimization. *Athena Scientific*, 1999.

[193] Elad Hazan. Introduction to online convex optimization. *arXiv preprint arXiv:1909.05207*, 2019.

[194] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.

[195] Manoj Kumar, William P Horn, Jeremy Kepner, José E Moreira, and Pratap Pattnaik. Ibm power9 and cognitive computing. *IBM Journal of Research and Development*, 62(4/5):10–1, 2018.

[196] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[197] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[198] Martin Zinkevich, Markus Weimer, Alexander J Smola, and Lihong Li. Parallelized stochastic gradient descent. In *NIPS*, volume 4. Citeseer, 2010.

[199] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[200] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *NIPS*, 2015.

[201] Xianfeng Liang, Shuheng Shen, Jingchang Liu, Zhen Pan, Enhong Chen, and Yifei Cheng. Variance reduced local sgd with lower communication complexity. *arXiv preprint arXiv:1912.12844*, 2019.

[202] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 5451–5452. IEEE, 2012.

[203] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[204] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M Hellerstein. Distributed graphlab: A framework for machine learning in the cloud. *arXiv preprint arXiv:1204.6078*, 2012.

[205] Jean Jacques Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 255:2897–2899, 1962.

[206] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(2), 2013.

[207] Paul Tseng and Sangwoon Yun. Incrementally updated gradient methods for constrained and regularized optimization. *Journal of Optimization Theory and Applications*, 160(3):832–853, 2014.

[208] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

[209] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pages 4615–4625. PMLR, 2019.

[210] Yichen Ruan, Xiaoxi Zhang, Shu-Che Liang, and Carlee Joe-Wong. Towards flexible device participation in federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3403–3411. PMLR, 2021.

[211] Chen Chen, Wei Wang, and Bo Li. Round-robin synchronization: Mitigating communication bottlenecks in parameter servers. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 532–540. IEEE, 2019.

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

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

[214] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

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

[216] Yufeng Zhan, Peng Li, and Song Guo. Experience-driven computational resource allocation of federated learning by deep reinforcement learning. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 234–243. IEEE, 2020.

[217] Shashi Raj Pandey, Nguyen H Tran, Mehdi Bennis, Yan Kyaw Tun, Aunas Manzoor, and Choong Seon Hong. A crowdsourcing framework for on-device federated learning. *IEEE Transactions on Wireless Communications*, 19(5): 3241–3256, 2020.

[218] Yuntao Wang, Zhou Su, Ning Zhang, and Abderrahim Benslimane. Learning in the air: Secure federated learning for uav-assisted crowdsensing. *IEEE Transactions on network science and engineering*, 2020.

[219] Guan Wang, Charlie Xiaoqian Dang, and Ziye Zhou. Measure contribution of participants in federated learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2597–2604. IEEE, 2019.

[220] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[221] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

[222] Ikuo Keshi, Yu Suzuki, Koichiro Yoshino, and Satoshi Nakamura. Semantically readable distributed representation learning for social media mining. In *Proceedings of the International Conference on Web Intelligence*, pages 716–722, 2017.

[223] Miao Fan, Qiang Zhou, Thomas Fang Zheng, and Ralph Grishman. Distributed representation learning for knowledge graphs with entity descriptions. *Pattern Recognition Letters*, 93:31–37, 2017.

[224] Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *International Conference on Machine Learning*, pages 1764–1773. PMLR, 2019.

[225] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *arXiv preprint arXiv:2103.00710*, 2021.

[226] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, 33:3557–3568, 2020.

[227] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.

[228] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758*, 2020.

[229] Neel Guha, Ameet Talwalkar, and Virginia Smith. One-shot federated learning. *arXiv preprint arXiv:1902.11175*, 2019.

[230] Tiffany Tuor, Shiqiang Wang, Bong Jun Ko, Changchang Liu, and Kin K. Leung. Overcoming noisy and irrelevant data in federated learning. *arXiv preprint arXiv:2001.08300*, 2020.

[231] Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*, 2013.

[232] Kai Yang, Tao Fan, Tianjian Chen, Yuanming Shi, and Qiang Yang. A quasi-newton method based vertical federated learning framework for logistic regression. *arXiv preprint arXiv:1912.00513*, 2019.

[233] Yang Liu, Xiong Zhang, and Libin Wang. Asymmetrical vertical federated learning. *arXiv preprint arXiv:2004.07427*, 2020.

[234] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824*, 2019.

[235] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[236] Siwei Feng and Han Yu. Multi-participant multi-class vertical federated learning. *arXiv preprint arXiv:2001.11154*, 2020.

[237] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 2021.

[238] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 653–664, 2007.

[239] Richard Nock, Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Entity resolution and federated learning get a federated resolution. *arXiv preprint arXiv:1803.04035*, 2018.

[240] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020.

[241] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[242] Qiong Wu, Kaiwen He, and Xu Chen. Personalized federated learning for intelligent iot applications: A cloud-edge based framework. *IEEE Open Journal of the Computer Society*, 1:35–44, 2020.

[243] István Hegedűs, Gábor Danner, and Márk Jelasity. Gossip learning as a decentralized alternative to federated learning. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 74–90. Springer, 2019.

[244] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[245] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine*, 58(6):46–51, 2020.

[246] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

[247] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[248] Shaohuai Shi, Kaiyong Zhao, Qiang Wang, Zhenheng Tang, and Xiaowen Chu. A convergence analysis of distributed sgd with communication-efficient gradient sparsification. In *IJCAI*, pages 3411–3417, 2019.

[249] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

[250] Lumin Liu, Jun Zhang, Shenghui Song, and Khaled B Letaief. Hierarchical quantized federated learning: Convergence analysis and system design. *arXiv preprint arXiv:2103.14272*, 2021.

[251] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization. *arXiv preprint arXiv:2102.07623*, 2021.

[252] Shashank Jere, Qiang Fan, Bodong Shang, Lianjun Li, and Lingjia Liu. Federated learning in mobile edge computing: An edge-learning perspective for beyond 5g. *arXiv preprint arXiv:2007.08030*, 2020.

[253] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[254] Yi Liu, Xingliang Yuan, Zehui Xiong, Jiawen Kang, Xiaofei Wang, and Dusit Niyato. Federated learning for 6g communications: Challenges, methods, and future directions. *China Communications*, 17(9):105–118, 2020.

[255] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[256] Chuan Ma, Jun Li, Ming Ding, Howard H Yang, Feng Shu, Tony QS Quek, and H Vincent Poor. On safeguarding privacy and security in the framework of federated learning. *IEEE network*, 34(4):242–248, 2020.

[257] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

[258] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in Neural Information Processing Systems*, 32:14774–14784, 2019.

[259] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[260] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[261] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[262] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018.

[263] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[264] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL `http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[265] Jiahuan Luo, Xueyang Wu, Yun Luo, Anbu Huang, Yunfeng Huang, Yang Liu, and Qiang Yang. Real-world image datasets for federated learning. *arXiv preprint arXiv:1910.11089*, 2019.

[266] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[267] Chris Pemberton. 3 ai trends for enterprise computing. Gartner, Oct. 5 2017. URL `https://www.gartner.com/smarterwithgartner/3-ai-trends-for-enterprise-computing/`.

[268] Inci M Baytas, Ming Yan, Anil K Jain, and Jiayu Zhou. Asynchronous multi-task learning. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 11–20. IEEE, 2016.

[269] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[270] En Li, Zhi Zhou, and Xu Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pages 31–36, 2018.

[271] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[272] Qianyu Meng, Kun Wang, Xiaoming He, and Minyi Guo. Qoe-driven big data management in pervasive edge computing environment. *Big Data Mining and Analytics*, 1(3):222–233, 2018.

[273] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[274] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3): 1628–1656, 2017.

[275] Edge computing market growth & trends. Grand View Research, May 2021. URL `https://www.grandviewresearch.com/press-release/global-edge-computing-market`.

[276] Ying Yu, Min Li, Liangliang Liu, Yaohang Li, and Jianxin Wang. Clinical big data and deep learning: Applications, challenges, and future outlooks. *Big Data Mining and Analytics*, 2(4):288–305, 2019.

[277] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[278] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018.

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

[280] Weiwei Lin, Wentai Wu, Haoyu Wang, James Z Wang, and Ching-Hsien Hsu. Experimental and quantitative analysis of server power model for cloud data centers. *Future Generation Computer Systems*, 86:940–950, 2018.

[281] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[282] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[283] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[284] Jaejun Yoo, Namhyuk Ahn, and Kyung-Ah Sohn. Rethinking data augmentation for image super-resolution: A comprehensive analysis and a new strategy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8375–8384, 2020.

[285] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu, and Jing-Shang Jhang. Clustering-based undersampling in class-imbalanced data. *Information Sciences*, 409:17–26, 2017.

[286] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9268–9277, 2019.

[287] Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. How does data augmentation affect privacy in machine learning? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10746–10753, 2021.

[288] AJ Feelders. Learning from biased data using mixture models. In *KDD*, pages 102–107, 1996.

[289] Guoqiang Zhong, Li-Na Wang, Xiao Ling, and Junyu Dong. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, 2(4):265–278, 2016.

[290] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1920–1929, 2019.

[291] Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5732–5741, 2018.

[292] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[293] Fenglin Liu, Xian Wu, Shen Ge, Wei Fan, and Yuexian Zou. Federated learning for vision-and-language grounding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11572–11579, 2020.

[294] Don S Lemons. *An Introduction to Stochastic Processes in Physics*. Johns Hopkins University Press, 2003.

[295] Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.

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

[297] Stephen J Roberts and Will D Penny. Variational bayes for generalized autoregressive models. *IEEE Transactions on Signal Processing*, 50(9):2245–2257, 2002.

[298] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *Advances in Neural Information Processing Systems*, 31: 4447–4458, 2018.

[299] Jie Song, Tiantian Li, Zhi Wang, and Zhiliang Zhu. Study on energy-consumption regularities of cloud computing systems by a novel evaluation model. *Computing*, 95(4):269–287, 2013.

[300] Tianhao Wang, Johannes Rausch, Ce Zhang, Ruoxi Jia, and Dawn Song. A principled approach to data valuation for federated learning. In *Federated Learning*, pages 153–167. Springer, 2020.

[301] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Appendix A

Supplementary Results for SAFA

The following are the supplementary statistics as part of the experimental evaluation presented in Chapter 3.

Table A.1: Best accuracy of the global model on Task 1

Best accuracy (Task 1: regression)					
Fully local					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.6154	0.6308	0.5820	0.5423	0.5270
0.3	0.5806	0.6363	0.6145	0.5843	0.5443
0.5	0.5180	0.6043	0.6276	0.6181	0.5978
0.7	0.4443	0.5480	0.6327	0.6409	0.6361
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.6055	0.6413	0.6411	0.6417	0.6424
0.3	0.6117	0.6418	0.6415	0.6419	0.6418
0.5	0.4432	0.6164	0.6421	0.6419	0.6413
0.7	0.3763	0.5576	0.6283	0.6413	0.6418
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.6109	0.6415	0.6412	0.6417	0.6423
0.3	0.6077	0.6417	0.6416	0.6420	0.6418
0.5	0.4097	0.6073	0.6423	0.6418	0.6413
0.7	0.2882	0.5999	0.6297	0.6293	0.6418
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.6419	0.6414	0.6413	0.6417	0.6423
0.3	0.6426	0.6419	0.6416	0.6417	0.6419
0.5	0.6423	0.6415	0.6422	0.6419	0.6415
0.7	0.6402	0.6422	0.6417	0.6412	0.6420

Table A.2: Synchronization Ratio and futility percentage on Task 1

SR / futility percentage (Task 1: regression)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.100/0.04	0.300/0.03	0.500/0.05	0.700/0.06	1.000/0.06
0.3	0.100/0.09	0.300/0.14	0.500/0.14	0.700/0.17	1.000/0.16
0.5	0.100/0.26	0.300/0.27	0.500/0.27	0.700/0.23	1.000/0.25
0.7	0.100/0.33	0.300/0.36	0.500/0.31	0.700/0.38	1.000/0.35
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.080/0.21	0.300/0.07	0.500/0.04	0.700/0.04	1.000/0.06
0.3	0.100/0.17	0.300/0.12	0.500/0.14	0.648/0.32	1.000/0.16
0.5	0.079/0.39	0.234/0.41	0.500/0.28	0.700/0.25	1.000/0.25
0.7	0.078/0.49	0.300/0.33	0.500/0.35	0.644/0.51	1.000/0.35
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.910/0.00	0.894/0.00	0.900/0.00	0.906/0.00	0.882/0.00
0.3	0.722/0.00	0.714/0.00	0.716/0.01	0.678/0.00	0.700/0.00
0.5	0.498/0.02	0.502/0.01	0.470/0.00	0.536/0.01	0.506/0.01
0.7	0.368/0.04	0.346/0.02	0.362/0.03	0.342/0.04	0.354/0.03

Table A.3: Best accuracy of the global model on Task 2

best accuracy (Task 2: CNN)					
			Fully local		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.8849	0.9066	0.9026	0.9131	0.9019
0.3	0.8909	0.8932	0.8937	0.8909	0.9126
0.5	0.8649	0.8898	0.9021	0.8932	0.9081
0.7	0.8518	0.8956	0.9026	0.8959	0.9093
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9407	0.9664	0.9738	0.9766	0.9796
0.3	0.9326	0.9640	0.9705	0.9745	0.9755
0.5	0.9178	0.9532	0.9652	0.9696	0.9738
0.7	0.8818	0.9452	0.9534	0.9591	0.9672
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9423	0.9666	0.9732	0.9762	0.9791
0.3	0.9328	0.9626	0.9702	0.9741	0.9765
0.5	0.9232	0.9529	0.9650	0.9699	0.9321
0.7	0.8962	0.9434	0.9546	0.9599	0.9673
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9748	0.9746	0.9764	0.9779	0.9787
0.3	0.9698	0.9696	0.9727	0.9753	0.9781
0.5	0.9658	0.9672	0.9686	0.9697	0.9714
0.7	0.9604	0.9632	0.9652	0.9603	0.9645

Table A.4: Synchronization Ratio and futility percentage on Task 2

SR / futility percentage (Task 2: CNN)					
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.100/0.03	0.300/0.04	0.500/0.05	0.700/0.05	1.000/0.05
0.3	0.100/0.16	0.300/0.15	0.500/0.14	0.700/0.15	1.000/0.15
0.5	0.100/0.24	0.300/0.24	0.500/0.25	0.700/0.22	1.000/0.26
0.7	0.100/0.36	0.300/0.36	0.500/0.35	0.700/0.35	1.000/0.35
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.092/0.04	0.300/0.05	0.500/0.06	0.693/0.06	1.00/00.05
0.3	0.099/0.16	0.296/0.16	0.500/0.15	0.692/0.15	1.000/0.16
0.5	0.100/0.23	0.297/0.28	0.500/0.25	0.700/0.25	0.990/0.25
0.7	0.100/0.33	0.300/0.33	0.495/0.35	0.700/0.36	0.990/0.36
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.896/0.00	0.902/0.00	0.891/0.00	0.900/0.00	0.894/0.00
0.3	0.704/0.00	0.710/0.00	0.704/0.00	0.707/0.00	0.709/0.00
0.5	0.524/0.01	0.517/0.01	0.521/0.01	0.521/0.01	0.509/0.01
0.7	0.341/0.04	0.351/0.04	0.359/0.04	0.342/0.04	0.350/0.04

Table A.5: Best accuracy of the global model on Task 3

best accuracy (Task 3: SVM)					
Fully local					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.7793	0.6603	0.6307	0.6307	0.6307
0.3	0.7477	0.6363	0.6307	0.6307	0.6307
0.5	0.8419	0.6859	0.6339	0.6307	0.6307
0.7	0.9530	0.7886	0.6673	0.6491	0.6442
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9935	0.9942	0.9962	0.9992	0.9992
0.3	0.9961	0.9961	0.9962	0.9963	0.9992
0.5	0.9961	0.9960	0.9961	0.9962	0.9963
0.7	0.9961	0.9961	0.9961	0.9957	0.9962
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9959	0.9962	0.9961	0.9991	0.9993
0.3	0.9961	0.9961	0.9962	0.9963	0.9992
0.5	0.9961	0.9962	0.9959	0.9962	0.9962
0.7	0.9961	0.9776	0.9960	0.9960	0.9960
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.9962	0.9961	0.9962	0.9992	0.9992
0.3	0.9960	0.9961	0.9962	0.9991	0.9991
0.5	0.9959	0.9961	0.9962	0.9961	0.9962
0.7	0.9960	0.9934	0.9961	0.9958	0.9960

Table A.6: Synchronization Ratio and futility percentage on Task 3

SR / futility percentage (Task 3: SVM)					
FedAvg					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.100/0.05	0.300/0.05	0.500/0.05	0.700/0.05	1.000/0.05
0.3	0.100/0.15	0.300/0.15	0.500/0.15	0.700/0.15	1.000/0.15
0.5	0.100/0.25	0.300/0.25	0.500/0.25	0.700/0.25	1.000/0.25
0.7	0.100/0.35	0.300/0.35	0.500/0.35	0.700/0.35	1.000/0.35
FedCS					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.100/0.05	0.299/0.05	0.499/0.05	0.697/0.05	0.998/0.05
0.3	0.099/0.15	0.300/0.15	0.499/0.15	0.698/0.15	0.998/0.15
0.5	0.099/0.24	0.300/0.25	0.499/0.25	0.697/0.25	0.998/0.25
0.7	0.100/0.36	0.300/0.35	0.498/0.36	0.699/0.36	0.996/0.36
SAFA					
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.901/0.00	0.900/0.00	0.900/0.00	0.901/0.00	0.901/0.00
0.3	0.703/0.00	0.700/0.00	0.701/0.00	0.702/0.00	0.703/0.00
0.5	0.512/0.01	0.514/0.01	0.516/0.01	0.513/0.01	0.512/0.01
0.7	0.345/0.04	0.343/0.04	0.344/0.04	0.342/0.04	0.344/0.04

Appendix B

Homomorphic Encryption for FedProf

The proposed representation profiling and matching scheme encodes the latent representations of data into a list of distribution parameters, namely $RP(w, D) = \{(\mu_i, \sigma_i^2)|i = 1, 2, \ldots, q\}$ where q is the dimension of FC-1. Theoretically, the information leakage (in terms of the data in D) by exposing $RP(w, D)$ is very limited and it is basically impossible to reconstruct the samples in D given $RP(w, D)$. Nonetheless, Homomorphic Encryption (HE) can be applied to the footprints (both locally and on the server) so as to guarantee zero knowledge disclosure while still allowing profile matching under the encryption. The following content gives details on how to encrypt a representation profile and compute profile dissimilarity under Homomorphic Encryption (HE).

To calculate (5.21) and (5.22) under encryption, a client needs to encrypt (denoted as $[[\cdot]]$) every single μ_i and σ_i^2 in its profile $RP(w, D)$ locally before upload whereas the server does the same for its $RP^*(w, D^*)$. Therefore, according to Eq. (5.22), it follows that

$$\begin{aligned}[[\mathrm{KL}(\mathcal{N}_i^{(k)}||\mathcal{N}_i^*)]] =& \frac{1}{2}\log[[(\sigma_i^*)^2]] - \frac{1}{2}\log[[(\sigma_i^{(k)})^2]] - [[\frac{1}{2}]] \\ &+ \frac{([[(\sigma_i^{(k)})^2]] + ([[\mu_i^{(k)}]] - [[\mu_i^*]])^2}{2[[(\sigma_i^*)^2]]}, \end{aligned} \tag{B.1}$$

where the first two terms on the right-hand side require logarithm operation on the ciphertext. However, this may not be very practical because most HE schemes are designed for basic arithmetic operations on the ciphertext. Thus it is also necessary to consider the situation where HE scheme at hand only provides *additive and multiplicative* homomorphisms [53]. In this case, to avoid the logarithm operation, the client needs to keep every σ_i^2 in $RP^*(w, D_i)$ as plaintext and only encrypts μ_i, likewise for the server. As a result, the KL divergence can be computed under encryption as:

$$\begin{aligned}[[\mathrm{KL}(\mathcal{N}_i^{(k)}||\mathcal{N}_i^*)]] =& \Big[\Big[\frac{1}{2}\log(\frac{\sigma_i^*}{\sigma_i^{(k)}})^2 + \frac{1}{2}(\frac{\sigma_i^{(k)}}{\sigma_i^*})^2 - \frac{1}{2}\Big]\Big] \\ &+ \frac{1}{2(\sigma_i^*)^2}([[\mu_i^{(k)}]] - [[\mu_i^*]])^2 \end{aligned} \tag{B.2}$$

where the first term on the right-hand side is encrypted after calculation with plaintext values $(\sigma_i^k)^2$ and $(\sigma^*)^2$ whereas the second term requires multiple operations based on the ciphertext values $[[\mu_i^k]]$ and $[[\mu^*]]$.

Now, in either case, one can compute profile dissimilarity under encryption by summing up all the KL divergence values in ciphertext:

$$[[div(RP_k, RP^*)]] = \frac{1}{q}\sum_{i=1}^{q}[[\mathrm{KL}(\mathcal{N}_i^{(k)}||\mathcal{N}_i^*)]] \tag{B.3}$$

Appendix C

Convergence Analysis for FedProf

In this section the proof of the proposed Theorem 4.1 is provided. The analysis is mainly based on the results provided by ref. [85].Several notations are first introduced to facilitate the analysis.

C.1 Notations

Let U ($|U| = N$) denote the full set of clients and $S(t)$ ($|S(t)| = K$) denote the set of clients selected for participating. Let $w_k(t)$ denote the local model on client k at time step t. An auxiliary sequence $v_k(t)$ is defined for each client to represent the immediate local model after a local SGD update. Note that $v_k(t)$ is updated from $v_k(t-1)$ with learning rate η_{t-1}:

$$v_k(t) = w_k(t-1) - \eta_{t-1}\nabla F_k(w_k(t-1), \xi_{k,t-1}), \tag{C.1}$$

where $\nabla F_k(w_k(t-1), \xi_{k,t-1})$ is the stochastic gradient computed over a batch of data $\xi_{k,t-1}$ drawn from D_k with regard to $w_k(t-1)$.

This work also defines two virtual sequences $\bar{v}(t) = \sum_{k=1}^{N} \rho_k v_k(t)$ and $\bar{w}(t) = Aggregate(\{v_k(t)\}_{k=1}^{S(t)})$ for every time step t (Note that the actual global model $w(t)$ is only updated at the aggregation steps $T_A = \{E, 2E, 3E, \ldots\}$). Given an aggregation interval $E \geq 1$, the analysis is provided for the partial aggregation rule that yields

$\bar{w}(t)$ as:

$$\bar{w}(t) = \frac{1}{K} \sum_{k \in S(t)} v_k(t), \tag{C.2}$$

where $S(t)$ ($|S(t)| = K$) is the selected set of clients for the round $\lceil \frac{t}{E} \rceil$ that contains step t. At the aggregation steps T_A, $w(t)$ is equal to $\bar{w}(t)$, i.e., $w(t) = \bar{w}(t)$ if $t \in T_A$.

To facilitate the analysis, it is assumed that each client always performs model update (and synchronization) to produce $v_k(t)$ and $\bar{v}(t)$ (but obviously it does not affect the resulting $\bar{w}$ and w for $k \notin S(t)$).

$$w_k(t) = \begin{cases} v_k(t), \text{ if } t \notin T_A \\ \bar{w}(t), \text{ if } t \in T_A \end{cases} \tag{C.3}$$

For ease of presentation, two virtual gradient sequences are defined: $\bar{g}(t) = \sum_{k=1}^{N} \rho_k \nabla F_k(w_k(t))$ and $g(t) = \sum_{k=1}^{N} \rho_k \nabla F_k(w_k(t), \xi_{k,t})$. Thus it follows that $\mathbb{E}[g(t)] = \bar{g}(t)$ and $\bar{v}(t) = \bar{w}(t-1) - \eta_{t-1} g(t-1)$.

C.2 Assumptions

Four assumptions are made to support the analysis of convergence. Assumptions C.1 and C.2 are standard in the literature [55, 85, 162] defining the convexity and smoothness properties of the objective functions. Assumptions C.3 and C.4 bound the variance of the local stochastic gradients and their squared norms in expectation, respectively. These two assumptions are also made in refs. [85].

Assumption C.1. *$F_1, F_2, \ldots, F_N$ are L-smooth, i.e., for any $k \in U$, x and y: $F_k(y) \leq F_k(x) + (y-x)^T \nabla F_k(x) + \frac{L}{2} \|y - x\|_2^2$*

It is obvious that the global objective F is also L-smooth as a linear combination of $F_1, F_2, \ldots, F_N$ with $\rho_1, \rho_2, \ldots, \rho_N$ being the weights.

Assumption C.2. *$F_1, F_2, \ldots, F_N$ are μ-strongly convex, i.e., for all $k \in U$ and any x, y: $F_k(y) \geq F_k(x) + (y-x)^T \nabla F_k(x) + \frac{\mu}{2} \|y - x\|_2^2$*

Assumption C.3. *The variance of local stochastic gradients on each device is bounded: For all $k \in U$, $\mathrm{E}\|\nabla F_k(w_k(t), \xi_{t,k}) - F_k(w_k(t))\|^2 \leq \epsilon^2$*

Assumption C.4. *The squared norm of local stochastic gradients on each device is bounded: For all $k \in U$, $\mathrm{E}\|\nabla F_k(w_k(t), \xi_{t,k})\|^2 \leq G^2$*

C.3 Key Lemmas

To facilitate the proof of the main theorem, several key lemmas are first presented.

Lemma C.1 (Result of one SGD step). *Under Assumptions C.1 and C.2 and with $\eta_t < \frac{1}{4L}$, for any t it holds true that*

$$\mathbb{E}\|\bar{v}(t+1)-w^*\|^2 \leq (1-\eta_t\mu)\mathbb{E}\|\bar{w}(t)-w^*\|^2+\eta_t^2\mathbb{E}\|g_t-\bar{g}_t\|^2+6L\eta_t^2\Gamma+2\mathbb{E}\Big[\sum_{k=1}^{N}\rho_k\|w_k(t)-\bar{w}(t)\|^2\Big], \tag{C.4}$$

where $\Gamma = F^* - \sum_{k=1}^{N}\rho_k F_k^*$.

Lemma C.2 (Gradient variance bound). *Under Assumption C.3, one can derive that*

$$\mathbb{E}\|g_t - \bar{g}_t\|^2 \leq \sum_{k=1}^{N}\rho_k^2\epsilon_k^2. \tag{C.5}$$

Lemma C.3 (Bounded divergence of $w_k(t)$). *Assume Assumption C.4 holds and a non-increasing step size η_t s.t. $\eta_t \leq 2\eta_{t+E}$ for all $t = 1, 2, \ldots$, it follows that*

$$\mathbb{E}\Big[\sum_{k=1}^{N}\rho_k\|w_k(t) - \bar{w}(t)\|^2\Big] \leq 4\eta_t^2(E-1)^2G^2. \tag{C.6}$$

Lemmas C.1, C.2 and C.3 hold for both full and partial participation and are independent of the client selection strategy (See ref. [85] for their proofs). This following content is focused on opportunistic client selection.

Let q_k denotes the probability that client k gets selected. Given the optimal preference factors α_k for $k = 1, 2, \ldots N$ that satisfy $\alpha_k = -\frac{ln(\Lambda\rho_k)}{div(RP_k, RP^*)}$, we have $q_k = \frac{\lambda_k}{\Lambda} = \rho_k$ according to Eq. (5.25). The next two lemmas give important properties of the aggregated model $\bar{w}$ as a result of partial participation and non-uniform client selection/sampling.

Lemma C.4 (Unbiased aggregation). *For any aggregation step $t \in T_A$ and with $q_k = \rho_k$ in the selection of $S(t)$, it follows that*

$$\mathbb{E}_{S(t)}[\bar{w}(t)] = \bar{v}(t). \tag{C.7}$$

Proof. First, a key observation given by ref. [85] is presented as an important trick to handle the randomness caused by client selection with probability distribution $\{q_k\}_{k=1}^N$. By taking the expectation over $S(t)$, it follows that

$$\mathbb{E}_{S(t)} \sum_{k \in S(t)} X_k = K\mathbb{E}_{S(t)}[X_k] = K \sum_{k=1}^{N} q_k X_k. \tag{C.8}$$

Let $q_k = \rho_k$, take the expectation of $\bar{w}(t)$ over $S(t)$ and notice that $\bar{v}(t) = \sum_{k \in U} \rho_k v_k(t)$:

$$\begin{aligned}
\mathbb{E}_{S(t)}[\bar{w}(t)] &= \mathbb{E}_{S(t)}\big[\frac{1}{K} \sum_{k \in S(t)} v_k(t)\big] \\
&= \frac{1}{K}\mathbb{E}_{S(t)}\big[\sum_{k \in S(t)} [v_k(t)\big] \\
&= \frac{1}{K} K\mathbb{E}_{S(t)}[v_k(t)] \\
&= \sum_{k \in U} q_k v_k(t) \\
&= \bar{v}(t).
\end{aligned}$$

□

Lemma C.5 (Bounded variance of $\bar{w}(t)$). *For any aggregation step $t \in T_A$ and with a non-increasing step size η_t s.t. $\eta_t \le 2\eta_{t+E-1}$, it follows that*

$$\mathbb{E}_{S(t)} \|\bar{w}(t) - \bar{v}(t)\|^2 \le \frac{4}{K}\eta_{t-1}^2 E^2 G^2. \tag{C.9}$$

Proof. First, one can prove that $v_k(t)$ is an unbiased estimate of $\bar{v}(t)$ for any k:

$$\mathbb{E}_{S(t)}[v_k(t)] = \sum_{k \in U} q_k v_k(t) = \bar{v}(t). \tag{C.10}$$

Then by the aggregation rule $\bar{w}(t) = \frac{1}{K}\sum_{k\in S(t)} v_k(t)$, it follows that

$$\begin{aligned}
\mathbb{E}_{S(t)}\|\bar{w}(t) - \bar{v}(t)\|^2 &= \frac{1}{K^2}\mathbb{E}_{S(t)}\|K\bar{w}(t) - K\bar{v}(t)\|^2 \\
&= \frac{1}{K^2}\mathbb{E}_{S(t)}\|\sum_{k\in S(t)} v_k(t) - \sum_{k=1}^{K}\bar{v}(t)\|^2 \\
&= \frac{1}{K^2}\mathbb{E}_{S(t)}\|\sum_{k\in S(t)}\big(v_k(t) - \bar{v}(t)\big)\|^2 \\
&= \frac{1}{K^2}\Big(\mathbb{E}_{S(t)}\sum_{k\in S(t)}\|v_k(t) - \bar{v}(t)\|^2 \\
&\quad \underbrace{+\,\mathbb{E}_{S(t)}\sum_{i,j\in S(t), i\neq j}\langle v_i(t) - \bar{v}(t), v_j(t) - \bar{v}(t)\rangle\Big)}_{=0}, \qquad \text{(C.11)}
\end{aligned}$$

where the second term on the RHS of (C.11) equals zero because $\{v_k(t)\}_{k\in U}$ are independent and unbiased (see Eq. C.10). Further, by noticing $t - E \in T_A$ (because $t \in T_A$) which implies that $w_k(t-E) = \bar{w}(t-E)$ since the last communication, it follows that

$$\begin{aligned}
\mathbb{E}_{S(t)}\|\bar{w}(t) - \bar{v}(t)\|^2 &= \frac{1}{K^2}\mathbb{E}_{S(t)}\sum_{k\in S(t)}\|v_k(t) - \bar{v}(t)\|^2 \\
&= \frac{1}{K^2}K\mathbb{E}_{S(t)}\|v_k(t) - \bar{v}(t)\|^2 \\
&= \frac{1}{K}\mathbb{E}_{S(t)}\|\big(v_k(t) - \bar{w}(t-E)\big) - \big(\bar{v}(t) - \bar{w}(t-E)\big)\|^2 \\
&\leq \frac{1}{K}\mathbb{E}_{S(t)}\|v_k(t) - \bar{w}(t-E)\|^2, \qquad \text{(C.12)}
\end{aligned}$$

where the last inequality results from $\mathbb{E}[v_k(t) - \bar{w}(t-E)] = \bar{v}(t) - \bar{w}(t-E)$ and that $\mathbb{E}\|X - \mathbb{E}X\|^2 \leq E\|X\|^2$. Further,

$$\begin{aligned}
\mathbb{E}_{S(t)}\|\bar{w}(t) - \bar{v}(t)\|^2 &\leq \frac{1}{K}\mathbb{E}_{S(t)}\|v_k(t) - \bar{w}(t-E)\|^2 \\
&= \frac{1}{K}\sum_{k=1}^{N} q_k\mathbb{E}_{S(t)}\|v_k(t) - \bar{w}(t-E)\|^2 \\
&= \frac{1}{K}\sum_{k=1}^{N} q_k\underbrace{\mathbb{E}_{S(t)}\|\sum_{i=t-E}^{t-1}\eta_i\nabla F_k(w_k(i), \xi_{k,i})\|^2}_{Z_1}. \qquad \text{(C.13)}
\end{aligned}$$

Let $i_m = \arg\max_i \|\nabla F_k\big(w_k(i), \xi_{k,i}\big)\|, i \in [t-E, t-1]$. By using the Cauchy-Schwarz inequality, Assumption C.4 and choosing a non-increasing η_t s.t. $\eta_t \leq 2\eta_{t+E-1}$, it follows that

$$\begin{aligned}
Z_1 &= \mathbb{E}_{S(t)}\|\sum_{i=t-E}^{t-1}\eta_i\nabla F_k(w_k(i),\xi_{k,i})\|^2 \\
&= \sum_{i=t-E}^{t-1}\sum_{j=t-E}^{t-1}\mathbb{E}_{S(t)}\langle\eta_i\nabla F_k\big(w_k(i),\xi_{k,i}\big),\eta_j\nabla F_k\big(w_k(j),\xi_{k,j}\big)\rangle \\
&\le \sum_{i=t-E}^{t-1}\sum_{j=t-E}^{t-1}\mathbb{E}_{S(t)}\Big[\|\eta_i\nabla F_k\big(w_k(i),\xi_{k,i}\big)\|\cdot\|\eta_j\nabla F_k\big(w_k(j),\xi_{k,j}\big)\|\Big] \\
&\le \sum_{i=t-E}^{t-1}\sum_{j=t-E}^{t-1}\eta_i\eta_j\cdot\mathbb{E}_{S(t)}\|\nabla F_k\big(w_k(i_m),\xi_{k,i_m}\big)\|^2 \\
&\le \sum_{i=t-E}^{t-1}\sum_{j=t-E}^{t-1}\eta_{t-E}^2\cdot\mathbb{E}_{S(t)}\|\nabla F_k\big(w_k(i_m),\xi_{k,i_m}\big)\|^2 \\
&\le 4\eta_{t-1}^2E^2G^2. \qquad\qquad (C.14)
\end{aligned}$$

Plug Z_1 back into (C.13) and notice that $\sum_{k=1}^{N}q_k=1$,

$$\begin{aligned}
\mathbb{E}_{S(t)}\|\bar{w}(t)-\bar{v}(t)\|^2 &\le \frac{1}{K}\sum_{k=1}^{N}q_k4\eta_t^2E^2G^2 \\
&= \frac{4}{K}\eta_{t-1}^2E^2G^2.
\end{aligned}$$

□

C.4 Proof of Theorem 5.1

Proof. By taking expectation of $\|\bar{w}(t)-w^*\|^2$, it follows that:

$$\begin{aligned}
\mathbb{E}\|\bar{w}(t)-w^*\|^2 &= \mathbb{E}\|\bar{w}(t)-\bar{v}(t)+\bar{v}(t)-w^*\|^2 \\
&= \underbrace{\mathbb{E}\|\bar{w}(t)-\bar{v}(t)\|^2}_{A_1}+\underbrace{\mathbb{E}\|\bar{v}(t)-w^*\|^2}_{A_2}+\underbrace{\mathbb{E}\langle\bar{w}(t)-\bar{v}(t),\bar{v}(t)-w^*\rangle}_{A_3}
\end{aligned} \qquad (C.15)$$

where A_3 vanishes because $\bar{w}(t)$ is an unbiased estimate of $\bar{v}(t)$ by first taking expectation over $S(t)$ (Lemma C.4).

To bound A_2 for $t \in T_A$, Lemma C.1 is applied:

$$A_2 = \mathbb{E}\|\bar{v}(t) - w^*\|^2 \leq (1 - \eta_{t-1}\mu)\mathbb{E}\|\bar{w}(t-1) - w^*\|^2 + \underbrace{\eta_{t-1}^2 \mathbb{E}\|g_{t-1} - \bar{g}_{t-1}\|^2}_{B_1} + 6L\eta_{t-1}^2\Gamma + \underbrace{\mathbb{E}\Big[\sum_{k=1}^{N} \rho_k \|w_k(t-1) - \bar{w}(t-1)\|^2\Big]}_{B_2}. \quad \text{(C.16)}$$

Then Lemmas C.2 and C.3 are applied to bound B_1 and B_2 respectively, which yields:

$$A_2 = \mathbb{E}\|\bar{v}(t) - w^*\|^2 \leq (1 - \eta_{t-1}\mu)\mathbb{E}\|\bar{w}(t-1) - w^*\|^2 + \eta_{t-1}^2 \mathcal{B}, \quad \text{(C.17)}$$

where $\mathcal{B} = \sum_{k=1}^{N} \rho_k^2 \epsilon_k^2 + 6L\Gamma + 8(E-1)^2 G^2$.

To bound A_1, one can first take expectation over $S(t)$ and apply Lemma C.5 where the upper bound actually eliminates both sources of randomness. Thus, it follows that

$$A_1 = \mathbb{E}\|\bar{w}(t) - \bar{v}(t)\|^2 \leq \frac{4}{K}\eta_{t-1}^2 E^2 G^2 \quad \text{(C.18)}$$

Let $\mathcal{C} = \frac{4}{K}E^2G^2$ and plug A_1 and A_2 back into (C.15):

$$\mathbb{E}\|\bar{w}(t) - w^*\|^2 \leq (1 - \eta_{t-1}\mu)\mathbb{E}_{S(t)}\|\bar{w}(t-1) - w^*\|^2 + \eta_{t-1}^2(\mathcal{B} + \mathcal{C}). \quad \text{(C.19)}$$

Equivalently, let $\Delta_t = \mathbb{E}\|\bar{w}(t) - w^*\|^2$, then the following recurrence relation holds for any $t \geq 1$:

$$\Delta_t \leq (1 - \eta_{t-1}\mu)\Delta_{t-1} + \eta_{t-1}^2(\mathcal{B} + \mathcal{C}). \quad \text{(C.20)}$$

One can prove by induction that $\Delta_t \leq \frac{\nu}{\gamma + t}$ where $\nu = \max\left\{\frac{\beta^2(\mathcal{B}+\mathcal{C})}{\beta\mu - 1}, (\gamma+1)\Delta_1\right\}$ using an aggregation interval $E \geq 1$ and a diminishing step size $\eta_t = \frac{\beta}{t+\gamma}$ for some $\beta > \frac{1}{\mu}$ and $\gamma > 0$ such that $\eta_1 \leq \min\{\frac{1}{\mu}, \frac{1}{4L}\}$ and $\eta_t \leq 2\eta_{t+E}$.

First, for $t = 1$ the conclusion holds that $\Delta_1 \leq \frac{\nu}{\gamma+1}$ given the conditions. Then by assuming it holds for some t, one can derive from (C.20) that

$$\begin{aligned}
\Delta_{t+1} &\leq (1-\eta_t\mu)\Delta_t + \eta_t^2(\mathcal{B}+\mathcal{C}) \\
&\leq \Big(1-\frac{\beta\mu}{t+\gamma}\Big)\frac{\nu}{\gamma+t} + \Big(\frac{\beta}{t+\gamma}\Big)^2(\mathcal{B}+\mathcal{C}) \\
&= \frac{t+\gamma-1}{(t+\gamma)^2}\nu + \underbrace{\Big[\frac{\beta^2(\mathcal{B}+\mathcal{C})}{(t+\gamma)^2} - \frac{\beta\mu-1}{(t+\gamma)^2}\nu\Big]}_{\geq 0} \\
&\leq \frac{t+\gamma-1}{(t+\gamma)^2}\nu \\
&\leq \frac{t+\gamma-1}{(t+\gamma)^2-1}\nu \\
&= \frac{\nu}{t+\gamma+1},
\end{aligned} \tag{C.21}$$

which proves the conclusion $\Delta_t \leq \frac{\nu}{\gamma+t}$ for any $t \geq 1$.

Then by the smoothness of the objective function F, it follows that

$$\begin{aligned}
\mathbb{E}[F(\bar{w}(t))] - F^* &\leq \frac{L}{2}\mathbb{E}\|\bar{w}(t) - w^*\|^2 \\
&= \frac{L}{2}\Delta_t \leq \frac{L}{2}\frac{\nu}{\gamma+t}.
\end{aligned} \tag{C.22}$$

Specifically, by choosing $\beta = \frac{2}{\mu}$ (i.e., $\eta_t = \frac{2}{\mu(\gamma+t)}$), $\gamma = \max\{\frac{8L}{\mu}, E\} - 1$, it follows that

$$\begin{aligned}
\nu &= \max\Big\{\frac{\beta^2(\mathcal{B}+\mathcal{C})}{\beta\mu-1}, (\gamma+1)\Delta_1\Big\} \\
&\leq \frac{\beta^2(\mathcal{B}+\mathcal{C})}{\beta\mu-1} + (\gamma+1)\Delta_1 \\
&= \frac{4(\mathcal{B}+\mathcal{C})}{\mu^2} + (\gamma+1)\Delta_1.
\end{aligned} \tag{C.23}$$

By definition, it is always true that $w(t) = \bar{w}(t)$ at the aggregation steps. Therefore, for $t \in T_A$:

$$\begin{aligned}
\mathbb{E}[F(w(t))] - F^* &\leq \frac{L}{2}\frac{\nu}{\gamma+t} \\
&= \frac{L}{(\gamma+t)}\Big(\frac{2(\mathcal{B}+\mathcal{C})}{\mu^2} + \frac{\gamma+1}{2}\Delta_1\Big).
\end{aligned}$$

$\square$