FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

DISTRIBUTED MACHINE LEARNING ALGORITHMS FOR RESOURCE-CONSTRAINED HETEROGENEOUS INTERNET-OF-THINGS ENVIRONMENTS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Ahmed Imteaj

2022

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Ahmed Imteaj, and entitled Distributed Machine Learning Algorithms for Resource-Constrained Heterogeneous Internet-of-Things Environments, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Sundaraja Sitharama Iyengar

Leonardo Bobadilla

Ananda M. Mondal

B. M. Golam Kibria

M. Hadi Amini, Major Professor

Date of Defense: May 20, 2022

The dissertation of Ahmed Imteaj is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2022

DEDICATION

To my parents, partner, and siblings. Without your love, patience, and encouragement this dissertation would not have been possible.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor Professor M. Hadi Amini, for letting me conduct this research work under his supervision and for his support and encouragement throughout the past few years. In addition, I would like to thank Professors Sundaraja Sitharama Iyengar, Leonardo Bobadilla, Ananda M. Mondal, and B. M. Golam Kibria of the Knight Foundation School of Computing and Information Sciences (KFSCIS) at Florida International University (FIU), and Professor Shabnam Rezapour of the Enterprise and Logistics Engineering Department at FIU, for the suggestions they provided. I also would like to thank my friends and colleagues from the Sustainability, Optimization, and Learning for InterDependent networks (solid) laboratory at FIU. Last, but not least, I am extremely grateful for the sacrifices and deep love from my kind parents, siblings, and my supportive partner, Saika, for being such a significant influence in my life. I would never have been able to finish my dissertation without their support and encouragement.

ABSTRACT OF THE DISSERTATION

DISTRIBUTED MACHINE LEARNING ALGORITHMS FOR RESOURCE-CONSTRAINED HETEROGENEOUS INTERNET-OF-THINGS ENVIRONMENTS

by

Ahmed Imteaj

Florida International University, 2022

Miami, Florida

Professor M. Hadi Amini, Major Professor

With the improvement of network infrastructures and advancement of IoT technologies, now it is desirable to perform computation at the edges, rather than sharing data with a central fusion center, which is privacy-intrusive. Both conventional (centralized) and distributed machine learning (ML) algorithms fail to address underlying challenges related to users' privacy or capturing global knowledge of the whole network. To properly handle such challenges, a recently invented distributed ML technique, called Federated Learning was invented that shows us a pathway to construct a global model without exposing any user's private data through on-device model training utilizing edge resources. However, FL may face various challenges due to the lack of a convenient mechanism to prepare a federated dataset, and also the heterogeneous nature of the resource-constrained agents such as systems, statistical and model heterogeneity. We developed a distributed sensing mechanism through which any federated agents can be triggered and activated for sensing the environment. That novel approach shows a pathway to carry out the FL process in a real-world environment. Following this, we developed an FL model, FedAR by monitoring agent activities and leveraging available local computing resources, particularly for resource-constrained IoT devices (e.g., mobile robots), to accelerate

the learning process. Besides, we propose a tri-layer FL framework, FedPARL that helps resource-constrained FL agents consume less resources during training, and avoid untrustworthy and out-of-resource agents (e.g., low battery life) during agent selection for training and perform variable local epochs based on the agent's resource availability. We perform model pruning to reduce the size of the agent model that is effective for an FL-IoT setting. Afterwards, we proposed a coupling of distillation and dynamic local task allocation technique through which we can effectively handle model and systems heterogeneity of FL-IoT environments. Further, we focused on applying our developed distributed ML algorithm to improve the resilience of critical infrastructures through knowledge exchange. We extended the work by proposing a novel technique, FedResilience, to handle weak critical infrastructure agents by enabling partial computational tasks from the resource-constrained agents and exchanging information without sharing any raw data. Besides, we implemented two other FL applications to recognize human activities and forecast customers' financial distress in resource-constrained environments. From the experience we gathered over the past few years on FL, we conducted a comprehensive survey on FL, particularly for resource-constrained IoT environments that discussed the existing FL works, present challenges, and their potential solutions, applications, and future research directions in this domain.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

The ubiquitous nature of IoT devices causes huge data streams due to their widespread applications. Storing and processing such vast amounts of data in a centralized data storage units and computing servers is costly, highly insecure, and time-consuming. In order to attain a better machine learning (ML) model under the conventional centralized approach, the users may need to compromise their privacy by sending private data to the data center. However, in some cases, the nature of local data may be sensitive that requires further security measures to ensure user confidentiality and data privacy while exchanging data for computation and decision-making purposes. For instance, smartphone users may prefer to not share their images with other entities, or the clients would not prefer to share their messages with a advertising company. Alternatively, they may prefer to bring code or algorithm to the data locally, instead of passing local data to central fusion centers. Various distributed methods [Ami19, AMK19] are developed to minimize the computational burden and optimize the decision-making process. Federated Learning (FL) has come to the light because of its promising paradigm that enables distributed machine learning training over a network of available devices. Numerous studies from a wide range of research disciplines, including databases, distributed systems, cryptography, machine learning, and data mining, explored FL methods from various perspectives. The prevailing goal is to learn from the distributed dataset and simultaneously preserve privacy by not exposing the data. To this end, the introduction of FL maintains privacy by storing client data only on-device, eliminates the dependency on a single server to generate

prediction models by performing computation on client devices, and builds a smarter model by learning from various client models. It is to be noted that any resource-constrained device could be a server in an IoT environment; therefore, it is not a good solution to consider such a device to store all the extracted data of the available clients and generate a model likewise as conventional ML approach. Rather than, the server can only be used to perform aggregation on the collected local models to generate an updated global model. In this dissertation, we focus on the development, deployment, and ultimately implementation of FL in an IoT environment, where the IoT nodes are considered as clients with limited resources. These resources include computation power, communication bandwidth, memory, and battery power. The IoT clients may have different technical characteristics and available resources, and that is why all the clients can not be treated the same.

FL has a unique way of generating a cumulative global model by learning from the client's model parameters, and it has two distinctive challenges from conventional distributed optimization: systems heterogeneity and statistical heterogeneity [MMR+17, LSZ+20, YLCT19a, ZLL+18]. Although the earlier proposed FL algorithm (FedAvg) has a significant contribution in FL settings, it has missed some underlying challenges that can be observed in a heterogeneous FL-IoT setting. First, the FedAvg algorithm does not instruct how to uniquely identify each of the FL clients and trigger them for performing any actions. Second, the FedAvg assumes all the available clients as uniform capabilities and randomly selects a fraction of local clients for the training phase. However, in a real-world FL setting, we may observe a marginal difference in various clients in terms of their system configurations. Third, resource-constrained IoT devices may face difficulty in performing on-device training with large model sizes. Fourth, FedAvg does not entitle the participated clients to perform variable or partial amounts of work; rather, it simply drops the

participants that fail to perform a task within a specified time window. Fifth, the performance of FedAvg diverges significantly when the client has non-identically distributed data across their devices, i.e., there remains statistical heterogeneity within the FL networks. Therefore, if we apply the FL algorithms in a distributed heterogeneous IoT environment (with potentially several IoT agents that have limited local computing resources), then most of the IoT devices (considered as FL clients) may fail to accomplish a given task due to limited resources in terms of computational power, battery life, bandwidth, or memory availability, and as a consequence, the FL process may hamper. As the IoT devices are vulnerable and prone to attacks, they may generate inappropriate models (i.e., bad model injection) that may prolong the global model convergence and waste resources of the participated FL devices. The authors in [LSZ+18] proposed a framework that can handle both systems and statistical heterogeneity. However, they randomly select a subset of clients like the FedAvg algorithm [MMR+17], which would not be effective in an FL-IoT environment as most of the selected candidates could be inactive or out-of-resources. In the worst case, the random selection of the participants may lead them to choose all the straggler devices that could hardly perform an iteration. Besides, in their simulation, they consider that the straggler or inactive client would take a random local iteration between 1 to E, where E is the local epoch defined by the task publisher for the overall task. In the worst case, it is possible that most of the stragglers need to perform local epochs close to the E. That means, instead of considering the resource availability or previous history, they randomly assign a local epoch for the straggler or inactive clients. Particularly, in a real-life FL setting, such random assigning of local epoch to the stragglers would result in an ineffective model update. Moreover, the deployment of an effective FL strategy in solving real-world applications can be promising. For instance, critical infrastruc-

tures (e.g., power systems, transportation) are essential lifelines for most modern sectors and have utmost significance in our daily lives. However, such important domains can fail to operate due to systems failure or natural disasters. Though the major disturbances in such critical infrastructures are rare, the severity of such an event calls for developing effective resilience assessment strategies to mitigate relative loss. The traditional critical infrastructure resilience approaches consider that the available critical infrastructure agents are resource-sufficient and agree to exchange local data with the server and other agents. Such assumptions open up the door of two issues: (1) slow learning due to straggler effect or uncertainty in reaching convergence while applying on resource-constrained critical infrastructure agents, and (2) huge risk of privacy leakage. By understanding the pressing urgency of constructing an effective resilience model for the resource-constrained critical infrastructures, we aim to leverage FL in solving such real-world problems.

1.2 Background

The invention of new distributed optimization and learning techniques has recently been popular due to the extensive growth of data that opens the door to rethink the design of ML and data center settings [WRXM18]. On one side, the improvements of internet availability, speed, and architecture bring more convenience for Internet-of-things (IoT) services, while on the other hand, the ever-growing development of modern edge devices (e.g., smartphone, wearable devices, drones, and sensors) enables performing computation at the edge without passing local sensitive data to the server. The Federated Learning (FL) technique was invented after being motivated by the same theme [MMR+17]. Though FL faces many challenges in terms of systems and statistical heterogeneity, privacy, communication

overhead, and massively distributed federated network [YLCT19a, ITW+20], the wide popularity of the FL approach motivates researchers to develop new optimization techniques suitable for a federated setting. Such novel federated optimization technique outperforms the conventional distributed methods, e.g., mini-batch gradient descent [DGBSX12], or ADMM [BPC11]. The distributed optimization technique, e.g., [MMR+17], [BPC11, SCST17, ZLL+18] allows for inexact local model updating that would help to balance between computation and communication in large-scale networks and permit to active a small subset of devices at any iteration period. Besides, to avoid the issues regarding active clients and statistical heterogeneity of FedAvg algorithm, a couple of works, e.g., [HKMC19, BDKD19] have shown efforts to analyze FedAvg algorithm considering the non-federated setting, i.e., they assume the data to be identical and uniformly distributed. However, in a heterogeneous setting, it is not proper to assume that each local solver can perform the same stochastic process using their local data and can handle large model sizes. As the clients within the FL network may possess heterogeneous systems resources, such heterogeneity results in incomplete task completion by the participated clients. This may exacerbate straggler issues and degrade system performance. If the number of stragglers becomes high, then it may take a long time or even fail to reach the target convergence. One solution could be to avoid the resource-constrained clients or not select them during a training phase [BEG+19a]. However, dropping the stragglers could limit the number of active clients and it could bring bias during training or even some dropping clients may have important data with higher volume [LSZ+18]. Beyond systems heterogeneity of the FL clients, statistical heterogeneity or divergence of client model update is also a concern in federated networks. To handle statistical heterogeneity, some works proposed the idea of sharing either client's local data or server's proxy data [ZLL+18, JOK+18]. However, the assumption of

passing client data to the server or disseminating proxy data to all the clients could violate the privacy [HYF+20]. The authors in [LSZ+18] proposed a framework that can handle both systems and statistical heterogeneity. By a generalization of the FedAvg algorithm and adding a proximal term, they handle statistical diversity and allow partial work. However, they randomly select a subset of clients like the FedAvg algorithm [MMR+17], which would not be effective in an FL-IoT environment as most of the participants would be inactive or out-of-resources. In the worst case, the random selection of the participants may lead them to choose all the straggler devices that could hardly perform an iteration. Inspired by FedAvg [MMR+17] and FedProx [LSZ+18], we propose to design a tri-layer FL model, *FedPARL* that can be effective, specially in an FL-IoT settings. Our tri-layer FL model will be capable to perform model pruning, selecting proficient clients for the FL training, and enabling accepting partial works from the resource-constrained straggler clients. Our tri-layer FL framework would be able to accelerate convergence and improve robustness in a resource-constrained FL-IoT environment.

FL fits best in applications where we need to deal with sensitive information, and therefore, on-device training is more important than passing local data to the server. Several FL applications are available, e.g., smart healthcare [HYF+20], recommendation systems [YJSD20], mobile keyboard prediction [HRM+18], Google keyboard query suggestions [YA+18], etc. We observe that most of the existing FL approaches are designed to skip the challenges that could arise in a resource-constrained IoT environment. The authors in [ZGH+22] discussed the applications, challenges, and potentials of FL in IoT environments. Most of the existing FL applications are based on labeled data collected from clients or user activities (e.g. type URLs or keyboard, click button). Therefore, it is necessary to develop a framework that could help the FL client to carry-out on-device training with reduced model size and can

also handle the systems and statistical heterogeneity in presence of the participated clients. Besides, we figure out that there is currently no work that could tackle the challenges of resource-constrained critical infrastructures and improve resilience. We developed a technique utilizing local resources and exchanging local models of the infrastructure agents instead of sharing sensitive data, and finally, exploited the collaboratively learned knowledge about probable outage and resource availability status of the whole FL network to enhance the resilience operations.

1.3 Research Questions and Objectives

Question #1	How can we develop an effective strategy to construct distributed dataset for leveraging distributed learning process?
Objective #1 (Published Work)	An intelligent distributed sensing mechanism would help us to show a pathway to federated learning for making intelligent decisions. (Conf. Papers: CSCI'19 [IA19])
Question #2	How can we apply distributed machine learning technique by handling systems, statistical, and model heterogeneity of an IoT environment?
Objective #2 (Published Work)	A distributed learning model that facilitates lightweight model training, examines client resources and activities, allows variable local computational tasks, and leverages distillation technique would handle the systems, statistical, and model heterogeneity of an resource-constrained environment. (Conf. Papers: ICMLA'20 [IA20], SenSys'20 [Imt20a], Journal: Frontiers in Communications and Networks [IA21]).
Question #3	Can distributed machine learning technique be effective in improving real-world applications?
Objective #3 (Published Work)	A distributed machine learning strategy for a resilient management system could help to improve resilience operations of critical infrastructures, human-activity recognition, and financial distress prediction. (Conf. Papers: PESGM'20 [IAM20], SEST'20 [AIM20], IHCI'21 [IAA21], Elsevier ISWA [IA22])

Table 1.1: Published research papers during my Ph.D. in the theory and application domains of Federated Learning.

Theory	**Applications**
FL-IoT Survey [IEEE IoT journal'22]	IoT [CSCI'19, Mobiquitous'19]
Distributed Sensing Mechanism [CSCI'19]	Robotics [ACM SenSys'20]
FedAR: Activity and Resource-aware Federated Learning Model [ICMLA'19]	Activity Recognition [IHCI'21]
FedPARL: Tri-layer Lightweight FL Framework [Frontier Journal'20]	Finance [Elsevier Intelligent Systems with Applications journal]
FedMDP: FL Framework to Handle Model and System Heterogeneity [Journal of Parallel and Distributed Computing (under review)]	Energy Systems [SEST'20]
Federated Deep Learning [ICMLA'21]	Power System Resilience [Electronics Journal, IEEE PES GM'20]
	COVID-19 Prediction [Patterns'21 Journal]

1.4 Research Contributions

We published several peer-reviewed journals and conference papers both in the theory and applications of distributed machine learning. We listed all the publications that are out of my PhD research in Table **1.1**.

1.5 Dissertation Outline

The outline of the rest of the dissertation is as follows: The review of the related works is presented in chapter **2**. The detail of our proposed distributed sensing mechanism for smart-end user devices is detailed in chapter **3**. Chapter **4** presents our developed activity and resource-aware FedAR model. Chapter **5** discusses the proposed lightweight Federated Learning framework for resource-constrained devices. Chapter **6** covers our work on handling model and system heterogeneity via knowledge distillation and dynamic local task allocation. Chapter **7** presents several

Table 1.2: Published research papers during my Ph.D. in the theory and application domains of Federated Learning.

Chapter/Section	**Re-Used Published Research**
Chapter 2	FL-IoT Survey *IEEE Internet of Things Journal*, 2022 [ITW+22]
Chapter 3	Distributed sensing mechanism, *IEEE CSCI'19* [IA19]
Chapter 4	FedAR model: Activity and Resource-aware Federated Learning model, *IEEE ICMLA'20* [IA20], Robotics, *ACM SenSys'20* [Imt20a]
Chapter 5	FedPARL: Tri-layer Federated Learning Framework, *Frontiers in Communications and Networks* [IA21]
Chapter 6	FedMDP: Federated Learning Model to handle Model and Systems Heterogeneity
Chapter 7.1	Resilience [Electronics Journal [IKKA21], IEEE PES GM'20 [IAM20]], Energy Systems, *IEEE SEST'20* [AIM20]
Chapter 7.2	Human Activity Recognition, *IHCI'21* [IAA21]
Chapter 7.3	Finance [Elsevier Intelligent Systems with Applications journal] [IA22]

Federated Learning applications including improving resilience of critical infrastructures, human-activity recognition and financial distress prediction. Finally, chapter **8** concludes the dissertation with a discussion on the limitations of the proposed works and different directions for our future works. Table 1.2 provides an overview of my published research papers that have been used in some parts of this Ph.D. dissertation.

CHAPTER 2

A SURVEY ON FEDERATED LEARNING FOR RESOURCE-CONSTRAINED IOT DEVICES

[1] In this chapter, we propose to answer this question: *how to train distributed machine learning models for resource-constrained IoT devices?* To this end, we first explore the existing studies on FL, relative assumptions for distributed implementation using IoT devices, and explore their drawbacks. We then discuss the implementation challenges and issues when applying FL to an IoT environment. We highlight an overview of FL and provide a comprehensive survey of the problem statements and emerging challenges, particularly during applying FL within heterogeneous IoT environments. Finally, we point out the future research directions for scientists and researchers who are interested in working at the intersection of FL and resource-constrained IoT environments.

2.1 Introduction

In this section, we thoroughly explain the motivations of focusing on FL for resource-constrained IoT devices by conducting a comprehensive survey on this topic, followed by a literature review of recently published prior works. We then elaborately explain how our proposed survey is fundamentally necessary for the FL domain. We then discuss our contributions and the necessity of conducting this research. Finally, we highlight the organization of this comprehensive survey.

[1]This chapter is an edited version of the author's previous work published in [ITW+22]

2.1.1 Motivation

The ever-growing data collected/produced at edge devices is the result of billions of connected Internet-of-Things (IoTs) devices as every active IoT client extracts their observed data and pushes those data to the edge. The traditional machine learning (ML) approaches need to perform aggregation of that extracted data element on a data center or a single machine, and such a learning scheme is common in different AI-based giant companies such as Facebook and Google. Companies store all the data collected in their data center, where they train the respective ML model. To attain a better ML model under the conventional centralized approach, the users may need to compromise their privacy by sending private data to the data center. Such a model training strategy is privacy-intrusive, particularly when the clients need to address their personal or sensitive data to achieve a better training model.

Federated learning (FL) is such an approach that is capable of training a model, leveraging the private data of clients without ever sharing it with other entities. However, the client may possess a lack of resources to perform on-device computation and may fail to reach the target convergence within an expected time. Moreover, we may face some unique challenges that could not be observed in a traditional FL-based approach in terms of communication, computation, privacy, storage, power, and energy utilization, e.g., straggler issue, high energy consumption, handling dropped participants. This chapter reveals the challenges of FL setting in such a situation and describes the impact of having such resource-constrained clients within a network by considering their practical constraints. To that end, we emphasize the open research issues in this area and enumerate numerous future directions.

2.1.2 Related works and Contributions

Numerous studies from a wide range of research disciplines, including databases, distributed systems, cryptography, machine learning, and data mining, explored FL methods from various perspectives. It is a prevailing goal to learn from the distributed dataset and simultaneously preserve privacy by not exposing the data. In 1982, a cryptographic mechanism was developed to apply on encrypted data [Yao82]. The works of [AS00, BA05, VYJ08, GIKM00, HLN$^+$07] are some of the early examples to discover knowledge from local data while maintaining privacy. To that end, the introduction of FL maintains privacy by storing client data only on-device, eliminates the dependency on a single server to generate prediction model by performing computation on client devices, and builds a smarter model by learning from various client models. It is to be noted that any resource-constrained device could be a server in an IoT environment; therefore, it is not a good solution to consider such a device to store all the extracted data of the available clients and generate a model like conventional ML approach. Instead, the server can only be used to perform aggregation on the collected local models to generate an updated global model. In this chapter, we focus on the deployment and implementation of FL in an IoT environment, where the IoT nodes are considered as clients with limited resources. These resources include computation power, communication bandwidth, memory, and battery power. The IoT clients may have different technical characteristics and available resources, and that is why all the clients can not be treated the same.

Several detailed surveys on FL have already been conducted by assuming that all clients within the network are resource-unbounded. Li *et al.* [LSTS20] presented an overview of challenges, open problems, and issues associated with FL by considering the heterogeneity of devices; however, they assumed that all clients are resource-boundless. The authors of [YLCT19a] focused on the categorization of FL settings

while the authors in [NDR19] presented the issues of FL in a wireless environment. Besides, a federated optimization-based framework is proposed in [LLH+20], which is constructed by addressing challenges related to both system and statistical heterogeneity. They mentioned that straggler client is responsible for increasing statistical heterogeneity which put adverse impact during convergence. By adding proximal terms during local training, they obtained faster convergence and were able to analyze the effect of heterogeneity. Another exciting paper [WHW+19] discussed FL from the perspective of mobile edge computing (MEC), including caching and communication mechanism at the edge, while a detailed survey is presented in [KM+21] by analyzing the recent advancement and issues of FL.

The existing FL survey papers [LSTS20, YLCT19a, NDR19, LLH+20, LSTS20, KM+21, WHW+19] are mostly focused on FL settings, system design, and components, implementation challenges, or on recent advancements. On the other hand, edge computing surveys [Z+19, AZTS17, CY+18, MYZ+17, LO+18, WZZ+17, MB17] are mainly conducted on edge computing infrastructure, applications including ML and AI, resource-management, wireless communication, and security and privacy issues. However, all these works considered only heterogeneity of systems or their statistical data, and did not discuss the challenges that would arise when the clients are resource-bounded. Throughout this chapter, we point out the FL challenges while applying on a resource-constrained IoT environment, analyze the potential solutions towards those challenges, and reveal the future directions of this domain. This chapter is mainly a critical survey on the previous works that identifies gaps in resource-constrained FL implementation. To the best of our knowledge, this chapter is the first comprehensive survey on FL for resource-constrained IoT devices.

2.1.3 Organization

The rest of this chapter is organized as follows. In Section 2.2, we present an overview and taxonomy of FL with a comprehensive list of existing studies. In Section 2.3, we review distributed optimization and ML approaches. Section 2.4 presents a detail analysis of the major challenges of FL while applying on resource-constrained devices, which is followed by Section 2.5, where we discuss the potential solutions of those emerging challenges. After that, in Section 2.6, we present the existing FL applications, and in Section 2.7, we highlight the future research direction in FL-based IoT domain. Finally, in Section 2.8, we conclude this chapter.

2.2 An Overview of Existing Studies on Federated Learning Models

This section covers the definition of FL, a detailed description of the FL taxonomy, a brief highlight on the existing FL frameworks, and a comparison summary of the existing FL-based studies which are classified in terms of privacy maintenance, attack schemes, fairness, learning effectiveness, and resource utilization.

2.2.1 Definition of Federated Learning

Federated learning can be defined as a distributed machine learning approach where the clients train themselves locally without sharing their direct information to the server. By periodically updating a shared global model based on performing aggregation of each client model information, this approach trains each device to capture the global view [MMR+17]. A high-level architecture of FL process is presented in Fig. **2.1**. The FL process generally includes three steps:

Step 1 (Initiate training task and global model): In the initial phase, the central server decides the task requirement and target application. A global model (W_G^0) is initialized and the server broadcasts that global model to the selected local clients that are known as participants.

Step 2 (Local model update): Each participant generates a model utilizing their local data. Upon receiving the global model W_G^t (where t denotes the t-th iteration), each client k updates its model parameters W_i^t for finding optimal parameters that minimizes the local loss function $F_k(W_k^t)$. The local optimal models are then shared with the FL server.

Step 3 (Global aggregation): After receiving the local models from the participants, the FL server performs aggregations and generates an updated global model (W_G^{t+1}). The latest global model is again shared with all the new participants.

Steps 2 and 3 are repeated until the central server reaches a convergence by minimizing the global loss function $F(W_G^t)$ which can be expressed as follows [LSZ+18]:

$$\min_w f(w) = \sum_{k=1}^{N} P_k F_k(w)$$

where, N is the total number of available devices, $P_k(\geq 0)$ indicates the relative impact of each device k while satisfying $\sum_k P_k = 1$, and $F_k(w)$ is the expected prediction loss on a sample input of k^{th} device on parameter w. Each device k possesses n_k samples (where $n = \sum_k n_k$). thus, the relative impact of each local device can be expressed as $P_k = \frac{n_k}{n}$.

2.2.2 A Taxonomy of FL-based Systems

Federated learning system (FLS) can be categorized according to data sample, communication, prediction model, scale, privacy, and participation motivation (see

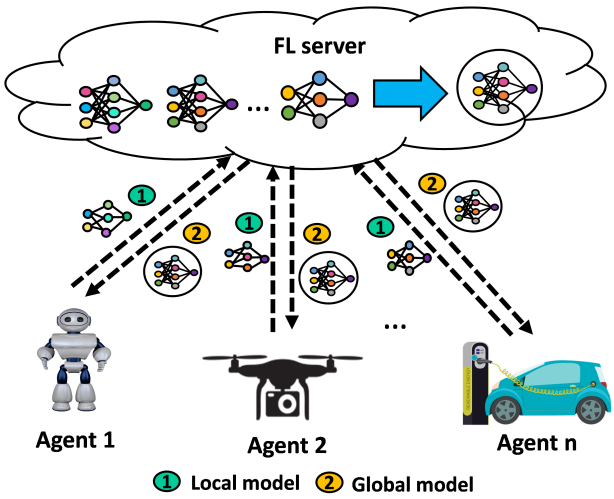


Figure 2.1: Federated learning procedure considering N number of participants.

Fig. **2.2**). In this segment, we discuss each individual categorization instance with proper examples.

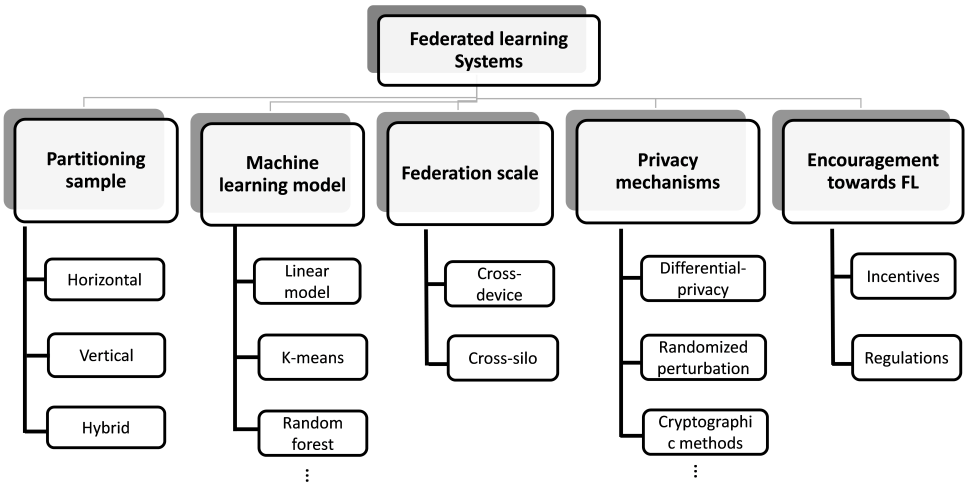


Figure 2.2: A taxonomy of federated learning based systems.

Partitioning sample

While designing an FL model, we need to analyze the data distribution records by utilizing both the features and non-overlapped instances. We can categorize FL into

(a) Horizontal FL, **(b)** Vertical FL, and **(c)** Hybrid FL based on the data samples distributed over networks and features space of those samples.

(a) Horizontal FL: Horizontal or sample-based FL have different data samples, but they share the same feature space. In Fig. **2.3**, we can see that two client devices have a data sample that is generated using some similar applications, and each client device has an identical feature space. Each client generates a local model by utilizing the data samples and carry out the FL process. We can also consider horizontal FL from the perspective of real-life scenarios. Assume that two local superstores have different customers, thus, the user intersection set would be minimal. However, the business structure and policy of the two superstores may be similar, i.e., the feature spaces are aligned. In such a case, we can apply horizontal FL to perform the learning action. Most of the FL studies conform to the horizontal FL strategy, where the local participants train their model by sharing the same feature space, and a similar global model is generated. Next-word prediction [HRM+18], wake-word detector [LCL+19], recommendation system [CLD+18] are some examples of horizontal FL.

(b) Vertical FL: In vertical or feature-based FL, the datasets share different sample spaces, but the sample IDs are the same (see Fig. **2.4**). For instance, consider a bank and a superstore in the same area. Most of their customers may be the same, but their business structure, i.e., the feature space, is different, and thus the user-space intersection is quite large. We can consider another example. Suppose we want to make a prediction model for product purchases based on user information, credit card rating, and purchasing history. In such a case, vertical FL can perform aggregation of these different features and collaboratively construct a prediction model. SecureBoost [CFJ+19], FedBCD [LKZ+19] are some of the examples of vertical FL.

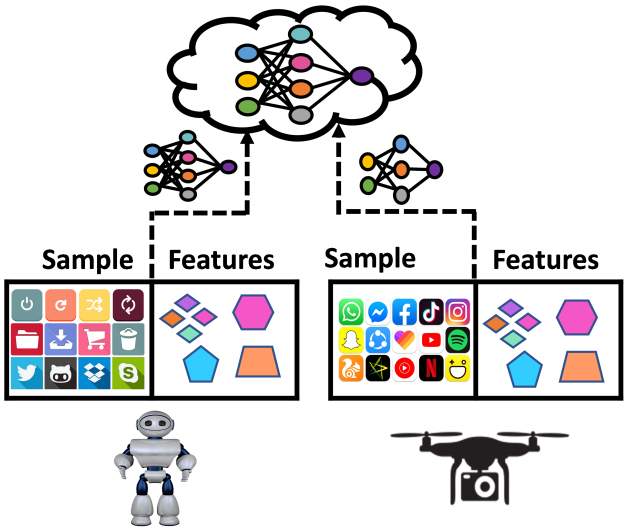


Figure 2.3: Horizontal Federated Learning scenario.

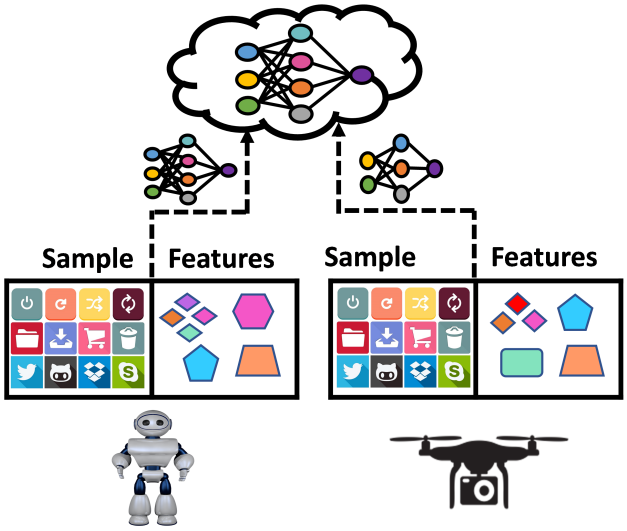


Figure 2.4: Vertical Federated Learning scenario.

(c) Federated Transfer Learning (FTL): FTL [PY09] can be consideblack as the combination of both horizontal and vertical partitioning of data (see Fig. **2.5**). Horizontal and vertical FL would not be effective when two clients (A and B) have small overlapping data samples and feature space, and we need to learn all the sam-

ple labels of a client (e.g., client A). FTL is applicable in such scenarios where the data samples and feature spaces are both different in the two clients' datasets. In other words, FTL can be applied when the clients' local data can differ in terms of both data samples and feature space. For instance, a group of research labs wants to invent a COVID-19 vaccine, but their samples (e.g., testing samples may contain different coronavirus categories) and feature spaces (i.e., strategic plan, test results) may be dissimilar. Similarly, two different multinational companies located in different countries may have different customers (i.e., samples) as well as distinguishable rules and regulations (i.e., features). Due to geographical location difference of the two companies, the overlapping data sample would be negligible, while due to different business types, there may be a very small intersection in the feature space. In such a case, FTL can be applied to handle variance in data sample and feature space while performing on-device learning. In FTL, an overlapping representation between two feature space of the clients are learned utilizing the small common data samples and each client obtains predictions for local samples using one-side features. Liu *et al.* [LCY18] designed a framework that can learn a feature representation of multiple parties based on common instances.

Machine learning Model

The appropriate ML model needs to adapt based on the training objective. For instance, if we want to classify the objects from an image, we need to train the FL model using convolutional neural networks (CNN). Several existing studies develop ML models for FL settings. The most popular ML model that is used in FL is Federated Stochastic Gradient Descent (Fed-SGD) coupled with neural network (NN), e.g., image classification [MMR$^+$17], word prediction [YA$^+$18, HRM$^+$18]. The decision tree (DT) is another popular and widely used ML method that is highly

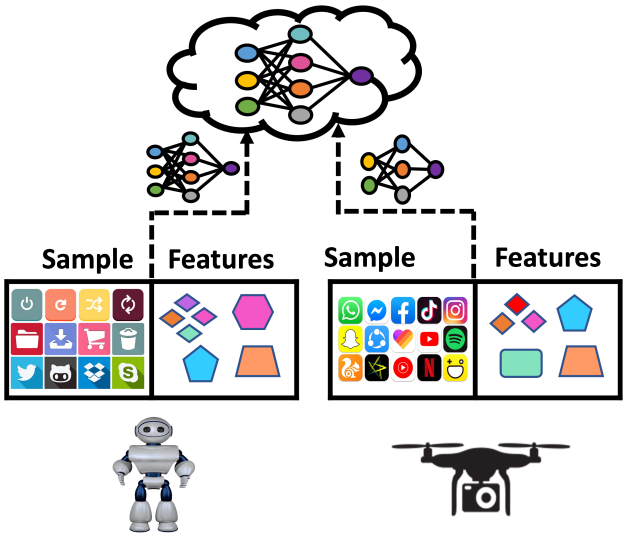


Figure 2.5: Federated Transfer Learning (FTL) scenario.

efficient for training models. In tree-based FL, a model is generated for training single or multiple decision trees. The authors in [LWH19, CFJ+19] designed a gradient boosting decision trees (GBDTs) by considering both horizontal and vertical partitioned data schemes. Different linear models (e.g., linear regression and classification, logistic regression, support vector machine (SVM) [HHIL+17, NWI+13]) are convenient to handle. Such linear models are easier to learn than different complex models (e.g., DTs, NNs). In a nutshell, many FL applications and frameworks are proposed on FedSGD [MMR+17, KMY+16, LSZ+18, WYS+20]. SGD is basically a common optimization technique that can be applied in different models, including SVM, linear regression, and NN. To improve the model accuracy in a large-scale FLS and to cover the gap between FLS with state-of-the-art ML models, it is necessary to exploit the ML architecture for obtaining better FL training.

Federation scale

FLS can be divided into cross-silo and cross-device categories based on the scale of federation [KM+21, LSTS20]. This categorization is performed based on the number of clients and their data quantity.

Cross-device: In cross-device FL, the number of clients can be large, but each client has a limited size of data. Different smartphones or IoT devices can be considered as the clients of such a system, which could be millions or billions in number. Recently, Google has invented an FL-based keyboard suggestion [YA+18] by training the model on-device of the user and aggregate the model information in the server. However, in such an approach, the clients may not be able to train themselves in a complex training environment because of resource scarcity. Thus, the server needs to be capable enough to process all the model information to generate a global training model.

Cross-silo: Cross-silo FL holds a relatively small number of clients, but they own a large amount of data. Typically, in cross-silo FL, the clients are data centers or different organizations. For instance, Amazon recommends products by training models using the collected data from hundreds of data centers, where each data center stores large amounts of data and configured with sufficient computational resources.

Encouragement towards FL

In real-world FL applications, the clients need encouragement or motivation to participate in the training phase and that can be carried out through regulations or incentives mechanism. For instance, Google FL keyboard suggestion [YA+18] can not force the users to provide data, but they ensure better keyboard suggestions

to the users who upload their data. Such incentives motivate the users to share information or performing on-device training.

2.3 Distributed Learning and Optimization Algorithms

In this section, we discuss the areas of research related to distributed learning and optimization techniques. Even though the main focus of the chapter is not in such domains, a brief highlight on these areas could motivate researchers to bring with a new or improved version of distributed learning setting or optimization techniques.

2.3.1 Federated Learning Algorithms

As we discussed earlier, after introducing FL by [MMR+17], several modified versions of the algorithms are proposed that can be effective in different circumstances. In this segment, we present some of the well-known and effective FL algorithms that would motivate researchers to introduce an improved version of FL.

Federated Averaging (FedAvg)

FedAvg algorithm [MMR+17] performs training operation via a central server that propagates a shared global model w_t, where indicates t the communication round. However, each client orchestrates local optimization using the concept of SGD. This algorithm has five hyperparameters: a fraction of clients or, participants C that takes part in the training round, size of local mini-batch B, learning rate η, number of local epoch on the client-side before updating of the global model E, and a learning rate decay λ. The algorithm is presented in Algorithm 1. When the system starts, the global model parameter w_o is randomly initialized (line **1**). At each communication

round of the server, a fraction of clients is selected (line **3**), and a random set of the client is chosen for the training phase (line **4**). Each client sends his/her local optimal model parameter, which is then aggregated onto the server (line **5-7**). The iteration period continues until a certain number of iterations, or if the update is small enough, or reaches a convergence.

Algorithm 1: Federated Averaging (FedAvg) [MMR+17]. The index of N clients are denoted by k; B represents the minibatch size of local client, E is the number of local epochs, n_k is the local examples of a client while n denotes the total data points, P_k stores a client data samples, and η represents the learning rate.

1 initialize w_o
2 for each round $t = 0, 1, 2, \ldots$ do
3 $\quad c \leftarrow \max(\lfloor C \cdot N \rfloor, 1)$
4 $\quad R_t =$ random set of c clients
5 $\quad$ for each client $k \in R_t$ in parallel do
6 $\quad\quad w_{t+1}^k =$ UpdateFromClient (k, w_t)
7 $\quad w_{t+1} \leftarrow \sum_{k=1}^{N} \frac{n_k}{n} w_{t+1}^k$
8 **UpdateFromClient** (k, w) : // Run on client k
9 $\quad Batch \leftarrow$ (split P_k into batches of size B)
10 $\quad$ for each local epoch i from 1 to E do
11 $\quad\quad$ for batch $b \in Batch$ do
12 $\quad\quad\quad w \leftarrow w - \eta \nabla \ell(w; b)$
13 $\quad$ return w to server

Local Gradient Descent

Large-scale models are often constructed, and first-order techniques are applied to solve related problems as they scale well in terms of dimension and data size. One popular choice is to use the Local Gradient Descent approach, where the optimization process is divided into epochs. Each iteration initiates to perform averaging steps across available N devices. The rest of the other epoch does not involve any further communication. Each client device implements a fixed number of Gradient

Descent (GD) steps (declares from the average model) using their local function independently in parallel [KMR19b]. See the details in Algorithm **2**.

Algorithm 2: Local Gradient Descent [KMR19b]. $\eta > 0$ **represents the learning rate, and** t_p **denotes a particular communication time and** g **indicates fixed number of Gradient Descent (GD) steps.**

Initialize vector w_0
Initialize $w_0^k = w_0$ for all $k \in [N] \overset{\text{def}}{=} \{1, 2, \ldots, N\}$
for $t = 0, 1, 2, \ldots$ do
    for $k = 1, 2, \ldots, N$ do
        $w_{t+1}^k = \begin{cases} \frac{1}{N} \sum_{g=1}^{N} \left(w_t^g - \eta \nabla f_g \left(w_t^g \right) \right), & \text{if } t = t_p,\ p \in \{1, 2, \ldots\} \\ w_t^k - \eta \nabla f_k \left(w_t^k \right), & \text{otherwise} \end{cases}$
    end for
end for

FedProx

In FL settings, the clients may need to perform a nonuniform amount of tasks that can handle the negative effect of system heterogeneity. Still, too many clients' updates can diverge the overall methods in the results of underlying heterogeneous data. The authors in [LSZ+18] proposed an algorithm named *FedProx* that is particularly useful for resource-constrained FL-based IoT environment. They enable variable local updates from the participated devices by adding a proximal term within the local subproblems. The proximal term is useful in two aspects. First, it limits the client's local updates to address the statistical heterogeneity issue. Second, it helps incorporate a variant amount of clients to work safely. We summarize the technique in Algorithm **3**.

q-FedAvg

Though the state-of-the-art FedAvg significantly accelerates the convergence speed [MMR+17], it fails to allocate client resources fairly (performs uniform allocation

Algorithm 3: FedProx [LSZ+18].

for $t = 0, \dots$ do
  Server randomly chooses a subset R_t of N devices (each client k is chosen with probability P_k)
  Server sends latest global model w_t to all chosen clients
  Each device $k \in R_t$ finds a w_{t+1}^k where, $w_{t+1}^k \approx \arg\min_w h_k(w; w_t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in R_t$ sends w_{t+1}^k back to the server
  Server aggregation, $w_{t+1} = \frac{1}{N}\sum_{k \in R_t} w_{t+1}^k$
end for

of resources). The allocation of resources is significantly crucial when we consider resource-constrained devices for the FL process. With this motivation, the authors in [LSS19] proposed q-FedAvg algorithm that can impose fairness based on the clients' contributions. In q-FedAvg algorithm, for a given cost functions F_k and parameter $q > 0$ (which is the fairness amount we wish to impose), the FL objective is defined as:

$$\min_w f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w),$$

where $F_k^{q+1}(\cdot)$ denotes $(q+1)$ as a power of $F_k(\cdot)$. Here, q is a parameter that tunes the amount of fairness we wish to impose. When $q > 0$, it prevents the execution of local SGD. To solve the issues of the local updating approach, particularly while allocating resources, the authors in [LSS19] proposed a heuristic solution by replacing gradient with the client's local updates obtained by running SGD on each local device. The algorithm is depicted in Algorithm **4.**

2.3.2 Distributed Learning

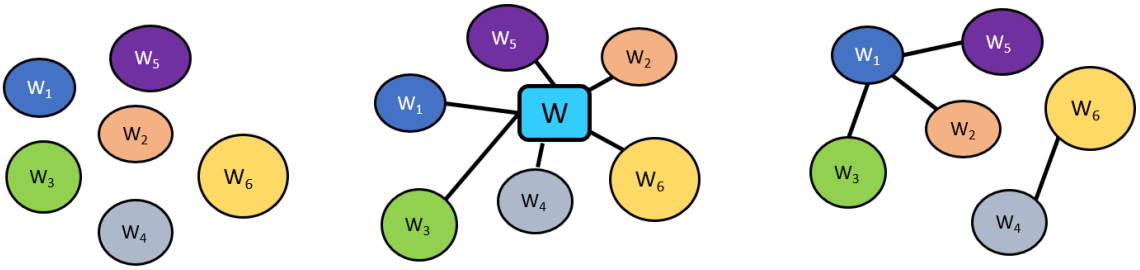
As discussed in Section **II**, the central server of the FL process orchestrates the learning process by managing the contributions of its clients. Thus, it can be considered as a single point of failure (see Fig. **2.6(b)**). Though large organizations and com-

Algorithm 4: q-FedAvg [LSS19]

for $t = 0, 1, 2, \cdots$ do
  Server randomly selects a subset R_t of N devices (each client k is chosen with probability P_k)
  Server sends latest global model w_t to all chosen clients
  Each chosen client device k updates w_t by performing SGD for E epochs with η to obtain $\bar{w}_{t+1}^k$
  Each selected client k computes:
    $\Delta w_t^k = L\left(w_t - \bar{w}_{t+1}^k\right)$
    $\Delta_t^k = F_k^q\left(w_t\right) \Delta w_t^k$
    $h_t^k = qF_{q-1}^k\left(w_t\right) \left\|\Delta w_t^k\right\|^2 + LF_q^k\left(w_t\right)$
  Each selected client k sends his/her parameters Δ_t^k and h_t^k to the server
  Server update: $w_{t+1} = w_t - \frac{\sum_{k \in S_t} \Delta_t^k}{\sum_{k \in S_t} h_t^k}$
end for

panies may afford to place a powerful, robust, and secure central server to carry out the training process, all types of sectors can not adapt that [VBT17]. Besides, some clients within the network can slow down the overall process [LZZ+17, BEG+19b]. The main idea of fully distributed and decentralized learning is peer-to-peer communications of the clients that eliminates the central server (see Fig. **2.6(c)**). In contrast, on-device training without learning from a server or its peers is shown in (see Fig. **2.6(a)**). In these figures, the communication topology looks like a connected graph, where each node represents a client, and the line between two nodes specifies a communication channel. In a distributed learning mechanism, each round corresponds to a local update by the clients and information exchange with peers. Though we do not have any global model or state as in standard FL, still, we can design the process such that all clients reach a global solution through local models. The local models can be converged through on-device training and learning from their peers [KM+21]. Fully decentralized SGD and other optimization algorithms are recently getting popular for scalability in large-scale systems [ALBR18] and de-

centralization of networks devices [VBT17, CBSC16, TLY+18, KSJ19, BGTT17, EPB+19, LKJK19, KTD+13, LASY14]. Note that even in the decentralized distributed setting, a central authority may need that will be in charge of setting up system configuration, learning tasks, hyper-parameters, algorithm selection, or resolve system failure. A degree of trust needs to establish among the clients to replace the central authority. Alternatively, such decisions can be made by a leader client, through a collaborative consensus scheme [W+14, MRTZ17, BIK+17].



(a) Learn model on device, not learn from peers.

(b) Learn through global model and on-device model.

(c) Learn model on device and from peers.

Figure 2.6: Different modeling approaches in federated networks. Depending on properties of the data, network, and application of interest, one may choose to (a) learn separate models for each device, (b) fit a single global model to all devices, or (c) learn related but distinct models in the network.

2.3.3 Distributed and Federated Optimization

The early trend of distributed optimization was naive distributed variants of corresponding serial algorithms, which is often inefficient in terms of communication. The second trend is to design communication-efficient algorithms. The idea is to perform a lot of local computation that is further followed by a communication round. Such technique is useful in practice and Distributed Approximate Newton (DANE) [SSZ14], CoCoA [JST+14], DiSCO [ZX15] are some of the examples of such

distributed optimization techniques. In distributed optimization, the data-centers possess huge data with relatively few devices. Later on, federated optimization is introduced to protect privacy in a better way. In that concept, the users keep their data private and provide the computational power of resources. Consequently, the data points are relatively smaller, the number of devices is huge, and data patterns vary on different devices.

There have been various methods to deal with distributed online optimization and distributed learning, including Stochastic Variance Reduced Gradient (SVRG) [JZ13, KR13], DANE [SSZ14] that is particularly for distributed optimization, naive Federated SVRG and Federated SVRG (FSVRG). The desirable properties while designing an algorithm for unbalanced, non-IID, and massively distributed can be stated as follows [KMY+16]:

1. An algorithm stays there in case it is initialized to the optimal solution.
2. In case a single node possesses all data, the algorithm should converge in $\mathcal{O}(1)$ communication rounds.
3. If all available features within the system occur on a single node, then the problem can be decomposed, and the algorithm is supposed to converge in $\mathcal{O}(1)$ rounds of communication.
4. If we assume that each node has an identical dataset, then the algorithm converges in $\mathcal{O}(1)$ communication rounds.

Property **(1)** is valuable for any optimization setting whereas properties **(2)** and **(3)** are applicable in federated optimization systems (e.g. unbalanced, non-IID, massively distributed). To the end, property **(4)** is an extreme case, particularly for a distributed optimization setting where we have a large number of IID data per device.

2.4 Learning on Resource-Constrained Devices

Before discussing the challenges associated with resource-constrained devices, it is essential to understand the definition of on-device learning of edge devices clearly. We can define an edge device as a resource-constrained entity with limited computational power, storage capacity, transmission range, and battery [DGL+19]. We consider an object as an edge device if it cannot be integrated with additional resources, i.e., the device resources can not be increased or decreased. For instance, a workstation cannot be considered as an edge device as we can integrate additional resources within that device. However, a manufactured robot can be considered as an edge device since we cannot directly incorporate any more support to the robot's capability. If we look at our today's IoT world, then we can see the use of resource-constrained devices in every aspect, from monitoring the environment to controlling human life. Within such an IoT environment, edge devices are utilized as they are smaller in size and are more transportable. Different kinds of robots, drones, and smartphones can be considered as edge devices possessing limited resources that communicate remotely. To attain optimal service performance from such IoT components, we need to train those edge devices that prevent them from being stragglers during the learning process. Those devices should be trained with diversified sample environments to perform accurate prediction in a various of testing data. It is not feasible to train those edge devices with a large dataset due to their limited resource availability. In this section, we discuss the potential challenges we may face while considering such resource-bounded IoT nodes in the FL environment.

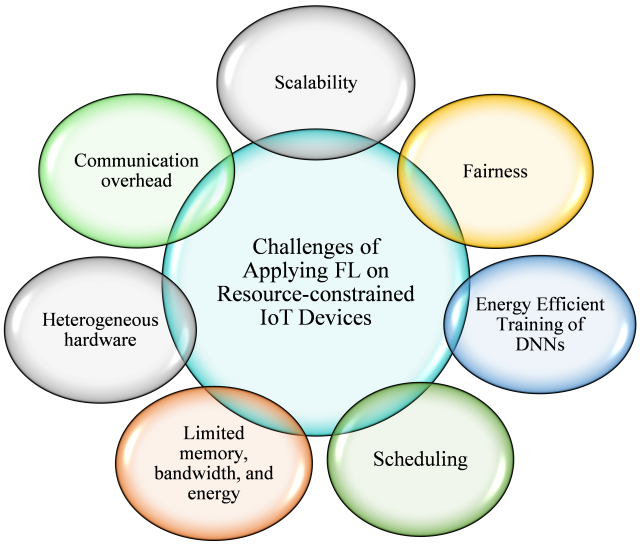


Figure 2.7: Core challenges of Federated Learning considering resource-constrained IoT devices.

2.4.1 Communication Overhead

Communication overhead is considered one of the major challenges in an FL-based IoT environment. The communication cost mainly increases due to the large sizes of data passing during the process and the iterative and non-optimized approach of conducting communication between the server and the clients. This problem becomes adverse when clients possess insufficient resources. For instance, if a client possesses limited bandwidth, then the client would not be able to communicate with the FL server effectively during model training. Similarly, if a client has weak processing capability, then performing an assigned local computational task would be infeasible for that client. Further, there could be large data across the network that could produce a large model size, and eventually, the resource-constrained clients would struggle in dealing with such a large model. To carry-out efficient training in a large data network, the client models need to be compressed so that the clients do not have to waste extra resources in training a large model. If a majority

of the FL clients are resource-constrained, then the FL process requires more server-client interaction to reach a target convergence, and the clients would not be able to afford such a high communication cost. While frequent FL server-client interaction can reduce convergence time, recurrent communication can encounter high costs. Therefore, it is required to design an effective optimization technique that can handle the trade-off between communication overhead and resource utilization of an FL setting. The authors in [MKJ+17] analyzed the trade-offs of communication and resource expense; however, they did not address the complexity of the clients' local model solutions.

2.4.2 Heterogeneous Hardware

The training phase of FL can run on multiple devices which may belong to various generations of products. Such product variation creates a network that consists of heterogeneous devices with a discrepancy in computational ability, memory size, or battery life. Therefore, the training period may vary significantly across clients, and it is not effective to consider all participants with the same scale. To achieve optimal results in training, FL needs to be aware of heterogeneous hardware configurations [LSTS20]. The proficient and trusted clients need to be selected in the training phase considering system requirements. After selecting suitable clients, it may be possible that a model fails to send its local model due to connection error or out of a battery issue (see Fig. **2.8**). However, due to system requirements (e.g. memory, bandwidth), most of the clients may not be able to be a part of the training round. Besides, it is possible that the majority of proficient clients go out of networks, and we may end up with a few clients that do not satisfy system requirements. Thus, carrying out the FL training process in such a situation is challenging.

2.4.3 Limited Memory and Energy Budget

In Section 2.4.2, we discuss heterogeneous hardware challenges, and in this segment, we describe the memory availability and energy budget issue across heterogeneous FL clients. Any FL client may have a very limited memory size, or a client having a larger memory size may not have space. Besides, the FL clients may have a preset energy budget, which may not fulfill the system requirements during the training process. While limited computational ability takes more processing time, memory shortage leads to over-flooding of the device. Such situations encounter extra communication overhead and degrade the system performance. Hard *et al.*

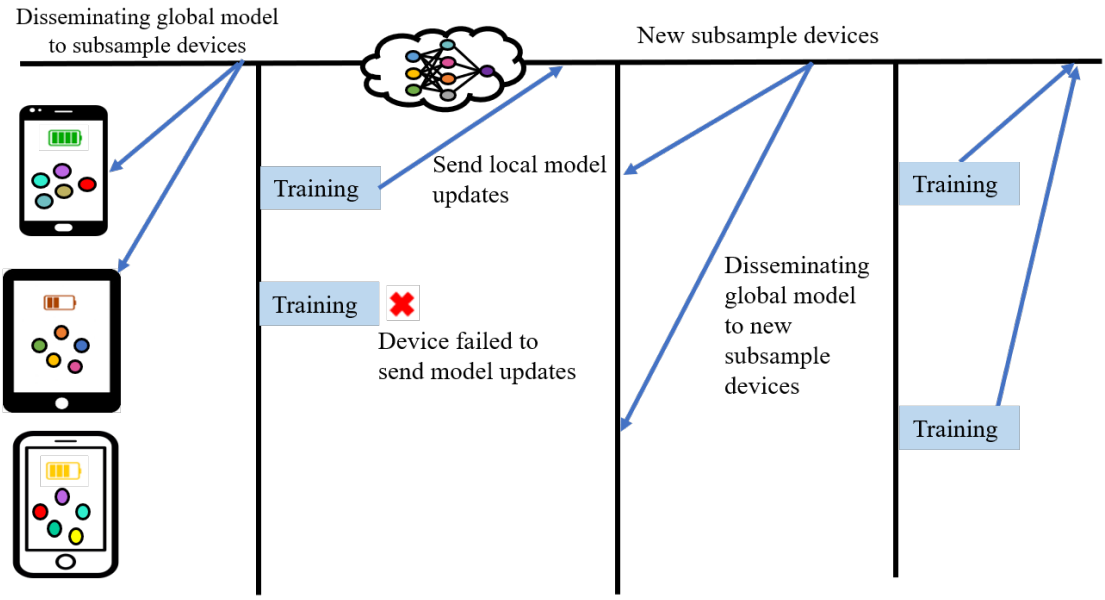


Figure 2.8: Systems heterogeneity scenario in Federated Learning.

[HRM+18] pointed out the necessary hardware requirement, including the required memory size and processing ability during their implementation of next-word prediction on the keyboard. They mentioned that to simulate their application, the device should have a minimum of 2 GB of free memory, whereas many IoT devices hardly possess even free megabytes of memory. Considering such memory

constraints, the authors in [HK+19] proposed an approach of distributing shards of data across FL clients to attain the target model swiftly. In their approach, they selected proficient clients who possessed a greater memory size, energy budget, higher bandwidth, and processing capability. However, they did not discuss the memory management and data handling for FL clients with limited available memory. We can manage such memory limitation by storing limited sizes of data, and in case of memory shortage, data aggregation technique can be applied to avoid memory outburst. The authors in [DB19, JWK+19, WTS+19, XY+19] analyzed hardware limitation challenges in the implementation of FL by considering Raspberry Pi and other types of resource-limited clients. They studied the feasibility of implementing FL on resource-constrained edge devices but did not cover the way of leveraging optimal memory requirement and quantifying energy budget throughout the FL process.

2.4.4 Scheduling

Existing federation optimization techniques can be classified into synchronous and asynchronous training. The authors in [KMR15, KMY+16, MMR+17, BEG+19a] focused on analyzing federated optimization that considers synchronous communication during training between the FL server and clients. In every training round, a subset of clients is triggered to perform a task. However, device or network issues can compel some clients to be unresponsive in the process, and the server needs to wait until getting a response from sufficient clients. Otherwise, the server drops that epoch as time-out and proceeds on to the next iteration. On the contrary, asynchronous optimization enables FL participants to directly send gradients to the FL server after every local update that is excluded in synchronous FL optimiza-

tion. Asynchronous training [ZLS09] is applied in some recent works because of its faster convergence when communication latency is comparatively higher and heterogeneous across the clients. The authors in [XKG19, LZZL17] analyzed asynchronous FL training with provable convergence by combining it with federated optimization. We present the synchronous and asynchronous FL behaviors in Fig. **2.9**.

In an FL process, it is indispensable to set the training phase of the participants, which is called scheduling. Scheduling is explicitly important when there exists resource-constrained IoT devices within the networks, and frequent interaction with the server costs more resources. Optimized scheduling can play a vital role in minimizing energy consumption as well as utilizing less bandwidth. Scheduling should be carried out in such a manner so that there remains less possibility of possessing old data by the participants. It is possible that some participants can generate local models utilizing their old data repeatedly while skipping new data [BEG+19a]. Such a situation can lead to resource-wastage without bringing any variations or improvements in the model. Besides, any participant can collect data by using a malicious application, and recognizing such harmful application data can be a challenge as it needs extra resources. Moreover, improper scheduling can lead to slow learning or straggler issues, which is considered as one of the reasons of a performance bottleneck, particularly for a resource-constrained FL-based IoT environment. By straggler clients, we mean the IoT devices that fail to respond within a specified period while the other clients react to the server successfully. Due to the slow response, the server needs to wait for the straggler client model, resulting in a delay in performing aggregation of the model parameters in synchronizing FL. If the number of such straggler clients is high, then the overall model convergence would be at stake [CFF+19, HZSK19]. Besides, in the conventional FL approach, the straggler clients are simply dropped [ITW+20]. However, if a significant portion

of the clients is straggler, and we drop all of them, then the model quality would be extremely low [XY+19]. Therefore, it is challenging to leverage a proper scheduling and guarantee model convergence even when a major portion of the clients are stragglers.

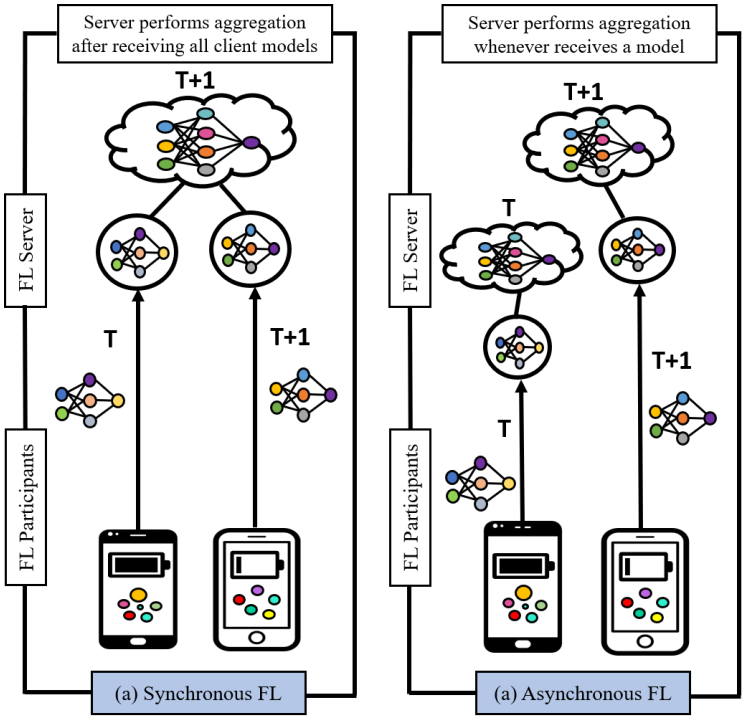


Figure 2.9: Difference between a synchronous and asynchronous Federated Learning.

2.4.5 Energy Efficient Training of DNNs

Deep neural networks (DNNs) are applied in various artificial intelligence and deep learning-based applications where the sample dataset is large. A lot of edge applications are now using DNN based algorithms [ZMR21, BRL+21] and there is an increasing focus on making DNN inference efficient on edge devices [BZF+21, CGW+20]. Additionally, FL requires edge devices to perform on-device training. However, training DNNs requires high computation capability, large memory and

energy availability, and most of the IoT clients may lack such system configurations. Wu *et al.* [WLCS18] proposed an approach to reduce the cost of training and inference by using lower-bitwidth integers for both stages of the application. Jiang *et al.* proposed an efficient learning technique using pruned models, while the authors in [PWE+19] proposed a strategy to generate a high-quality ML model through on-device model output, parameters, and data aggregation. Another interesting approach to training high capacity models having fewer parameters is discussed in [LCL+19]. Specifically, in case the size and features of the training dataset is huge, we need to devise energy-efficient training on resource-constrained clients, perhaps a challenging approach.

The memory requirement of the training phase of DNN has been an issue that is well studied for training on large GPU and CPU server clusters. Training on edge devices can benefit by adapting techniques that have served well in managing this problem in the server context like efficient gradient checkpointing [SAL+19], tensor rematerialization [JJN+20] and recompute [CXZG16]. The authors in [CGZH21] proposed another way to reduce the memory required for on-device training by introducing a lite residual module that can be adapted to new data. By only changing this lightweight module and keeping the other parameters constant, they reduce the memory requirement of the training process.

Another factor that can cost extra energy resources during training is mislabeled or unlabeled training data, particularly when the size of the dataset is large. The existing FL-based applications consider that all extracted data are appropriately labeled. Nevertheless, this assumption can be disproved if the collected data is mislabeled via security holes or unlabeled due to a network connection error. Mislabeled data would generate a wayward model that eventually affects the global model update. In case we have unlabeled data, it costs extra resources to put labels,

and that would be crucial for resource-constrained IoT settings. Gu *et al.* [GJ+19] proposed a framework to identify the mislabeled data which are injected through data poisoning attacks. Using representation-based fingerprints, they detect the malicious or compromised participant's data label while coming across erroneous predictions during runtime. Tuor *et al.* [TWK+20] proposed a method of finding and ignoring irrelevant data (possibly due to mislabelling) from FL. To come up with a solution of unlabeled data, the authors in [LLH+19] proposed a strategy to make labeling of unlabeled data through applying collaborative learning with the neighbor clients. Implementing the same procedure for resource-constrained clients would be challenging in real-time as it needs additional resources.

2.4.6 Fairness in Federated Learning

Fairness in the FL process means the distribution of client resources in an equitable manner. We can think of the global model as a resource, which is responsible for serving the client devices. However, the service that each user receives needs to be fair, i.e., the resource allocation and accuracy distribution across the client devices are unprejudiced. A minimax optimization framework named Agnostic FL [MSS19] is developed, which can optimize the target distribution of the centralized model and is formed as a mixture of participated client distributions. However, their proposed approach is applied only at small scales. The authors in [LSS19] used a α-fairness metric and proposed a q-Fair FL to ensure fair accuracy distribution. Their proposed strategy can tune resource distribution by considering the desired amount of fairness. A collaborative fair FL framework is proposed in [LXW20], which utilizes client reputation and compels them to converge to different models. They achieved fairness without degrading predictive performance. In [YLL+20a], an

FL-based client selection process is investigated to minimize clients' model exchange time that guarantees long-term, flexible fairness in the presence of rigid system constraints. However, they could not figure out a way to quantify how the fairness factor would impact the convergence speed and final target accuracy.

Moreover, some recent works on the optimization of resource allocation with incentive mechanisms for the FL process can be found in [KPT+20, YLL+20a, LW20, BSX+19, NSM20, CYW+20]. The authors in [KPT+20] designed an incentive-based FL model via a Stackelberg game for motivating client participation in the learning process. With the motivation of addressing issues related to costs and mismatch between client's contributions and receiving incentives, the authors in [YLL+20a, YLL+20b] proposed a payoff-sharing scheme named Federated Learning Incentivizer (FLI). Their proposed scheme can dynamically distribute a given budget among data owners by ensuring maximization of collective utility and minimization of inequality, considering the received rewards and waiting time for receiving those rewards. A trust and incentive-based FL model is designed in [BSX+19], where they proposed to add local computation results of the clients using the concept of blockchain consensus to establish a public auditable and decentralized FL ecosystem. In their model, honest clients can receive incentives while malicious clients are punished heavily in terms of payoffs. Besides, the authors in [NSM20] proposed a strategy of estimating the contributions of each client in an FL process and provide incentives accordingly, reducing the communication and computation overhead. Similarly, the authors in [CYW+20] designed a client contribution-based incentive method for FL but using the concept of Vickrey-Clarke-Groves (VCG) mechanism.

After analyzing the above-mentioned FL-fairness strategy, we can conclude that any client within an FL-IoT environment may have resource scarcity. Therefore, designing a fair resource allocation and distribution scheme is necessary to reduce

communication overhead, computation power and to achieve higher accuracy. We need to check clients' activities, resource status, and contributions towards model convergence to ensure fairness in FL resource allocation.

2.4.7 Scalability of Federated Learning

In a realistic FL-based IoT environment, we may observe a large number of IoT devices that are heterogeneous in nature and possess limited resources. In such a situation, FL training can be executed through effective client selection and optimal resource utilization. The authors in [CYS+19] developed a framework via joint learning and establishing wireless communication among the FL clients. They discussed that the FL process can be hampered due to packet errors or the unavailability of wireless resources (e.g., limited wireless bandwidth). Considering the factors, they formulate an optimization problem considering joint learning, resource block allocation, and effective user selection with a goal of minimizing FL loss function. They derive a closed-form expression for FL convergence by considering the effect of the wireless channel. Their proposed framework ensures scalability and sparsification. The authors in [NY19] design a client selection protocol, using FL edge server. Their proposed model can manage the communication resources between the FL server and the clients, choosing clients based on their resource conditions. Besides, an activity and resource-aware FL model is presented in [IA20]. They proposed a strategy of examining client's resources and assigning trust scores to clients as per their contributions towards model convergence. On the basis of sufficient resources and a higher trust score, they only select a subset of eligible clients for the training round from a large number of available clients. Their proposed model ensures scalability, robustness, and sparsification of the FL process. The authors

in [YYPH20] proposed a selective client model aggregation-based FL framework for vehicular edge computing. Instead of a random selection of FL clients, they leverage a technique of selecting clients based on contract theory. Moreover, a tri-layer lightweight FL framework is proposed in [IA21] that is capable to handle a large number of clients and their huge data streams across the networks. They shrink large model size through pruning mechanism, select clients based on their resource status and previous activities, handle divergent local model update, and also allow a variable local model update. Their proposed framework ensures scalability, quantization, robustness, and sparsification.

2.4.8 Privacy Issues

In federated settings, we keep the raw data of each client on-device due to privacy concerns. However, it is possible to leak sensitive information [MSDCS19, CLK+18, BDF+18, FJR15] through sharing model update during the training process. For instance, the authors in [CLK+18] presented that sensitive patterns (e.g., credit card numbers) can be extracted from a user-trained model based on recurrent neural networks. In the case of having sensitive datasets distributed across several data owners, privacy can be preserved via Secure Multiparty Computation (SMC) or Secure Function Evaluation (SFE). The protocol outcome enables multiple data owners to collaboratively agree to generate a function without leaking any information [RWT+18, Cha88, GT+19]. Though several privacy definitions for FL are stated in [GKN17, HLL+19, BDF+18, NSH19, ASY+18, CLK+18, LKCT19, LMX+19, MRTZ17, GPV19], we can classify them as global and local privacy. In the global privacy setting, the server is assumed to be trusted, and local model updates are private. In local privacy, individual local model updates are generated

on the client-side and aggregated on the server. However, due to the presence of resource-constrained devices, the existing privacy-preserving FL algorithms may not be suitable for running on those devices. Thus, beyond ensuring rigorous privacy guarantees, novel methods need to be designed that are communication-efficient, computationally cheap, and capable of handling dropped participants.

2.5 Potential Solutions of Emerging Challenges

In the previous section, we explored the implementation challenges of the FL process during on-device training with resource-scarce devices. A clear direction towards possible solutions for those emerging challenges can be effective in future research of this domain. This section describes the existing works and possible solutions of emerging challenges during training of resource-constrained devices in an FL environment.

2.5.1 Deploying Existing Algorithms to Reduce Communication Overhead

We explored a couple of key approaches that aim to reduce communication costs and can be classified into three categories: decentralized training, model compression, and importance-based updating. The integration of such strategies can be useful to overcome the trade-offs and shortcomings in this area. Haddadpour *et al.* [HK+19] proposed an approach to infuse redundancy among the clients to bring diversity and reach convergence taking less communication round. Chen *et al.* [CYS+19] also designed a framework of joint-learning by considering the effect of wireless factors on participants in the FL scenario. Some of these methods adapted model

compression strategies, but those methods may deteriorate model accuracy and encounter high computational costs. Such trade-offs are empirical, i.e., we need to conduct several local training rounds to find an optimal number of iterations before making a communication. FL method can be more scalable if we can apply effective optimization techniques that are formalized theoretically, and implemented and tested empirically. Apart from compressing the model size, FL approaches can be motivated by MEC paradigms and their applications. For instance, the authors in [LZSL19] considered an intermediate model aggregator for reducing instances during device-cloud communication. However, their model costs more time to converge when the number of clients or edge servers increased. The situation becomes adverse when there exists non-IID data across the network. Through multi-task learning [SCST17], such a statistical challenge can be handled. Moreover, FL models can be exploited to efficiently utilize the storage and computing power for facilitating efficient FL.

To reduce communication overhead, the authors in [IA19, SZG19] discussed infusing redundancy among the client dataset to reach convergence with fewer communication rounds. In Fig. **2.10**, we see that a particular data collection point $\mathcal{L}_1$ is used by two clients $\mathcal{D}_1$ and $\mathcal{D}_4$. Similarly, other data collection points i.e., $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$ are utilized by $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D}_2$ and $\mathcal{D}_3$, and $\mathcal{D}_3$ and $\mathcal{D}_4$, respectively. This setting leads to infusing redundant data samples among the client devices. According to [KMY+16, HWL20], a novel approach is proposed to share compressed sizes of message and carry out the reduced number of communication rounds to attain the target model. With the same motivation, the authors in [CKMT18] applied lossy compression and federated dropout to train a smaller subset of local clients and reduce client-to-server interaction and local computation (see Fig. **2.11**). Though frequent communication may accelerate convergence, recurrent interaction incurs more com-

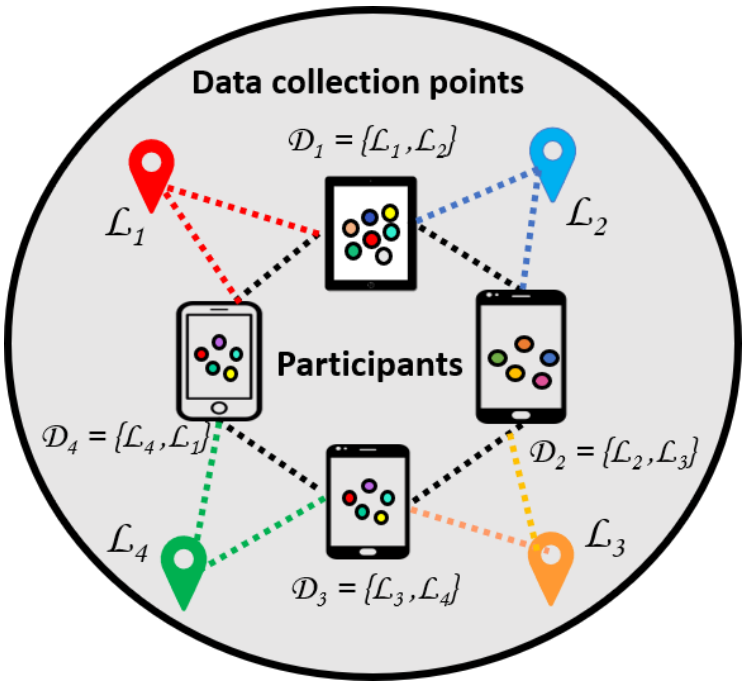


Figure 2.10: Infusing data redundancy through overlapping data collection.

munication costs. Every time a client interacts with the server, it has to compromise its resources. To handle the limited resources of the clients, a resource-optimization algorithm is necessary to consider such a trade-off. In this regard, the authors in [MKJ+17, SLY+20, LGN+20, LKZ+19, DTN20, LHZ+20, FMO20] studied the relation between communication cost and effective resource utilization, though they did not discuss the complications of the local problem's solution. Wang *et al.* [WTS+19] presented a distributed control algorithm for minimizing the training loss under a given resource budget. Besides, the authors in [WHW+19] designed a framework to exchange learning parameters of the clients through the collaboration for generating better local training models. Hence, this reduces communication overhead and ensures both system and application level improvement, which generates additional energy cost. A detailed discussion on the trade-offs between FL training period and energy requirement cost can be seen in [YCS+19, LLH+19, YJSD20, TBZ+19].

They minimized the weighted sum of the training completion period and energy consumption, applying an iterative algorithm. In the case of delay-sensitive scenarios, they adjusted the weights so that FL participants would expend more energy to achieve time minimization.

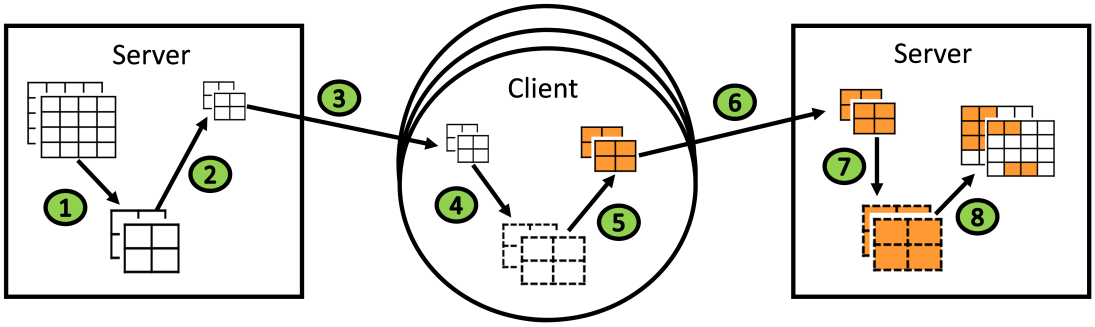


Figure 2.11: Reducing size of the model by (1) generating a sub-model applying Federated Dropout, (2) lossily compressing the obtained resulting object which is passed to the client, who (4) applies decompression and trains that using its own local data, and (5) again compresses the update, which is sent back to network server. There it is (7) again decompressed and finally, (8) aggregated to be a part of the global model.

However, most of the studies that we discussed do not consider the heterogeneity of client resources. Due to such heterogeneity, some of the approaches can not be adapted in such a resource-constrained FL-based IoT environment. For instance, the key idea of [MMR+17] was to allow for more computation on the mobile-edge side, e.g., by conducting more local updates before interacting with the server. Such an application requires processing power, which may not be feasible for IoT clients with weak-processing units. Finally, the resource-limitations may cause a straggler effect.

2.5.2 Convergence Guarantee in Asynchronous FL

In Section 2.4.4, we highlighted the difference between synchronous and asynchronous communication. Most of the existing FL approaches are implemented on the concept of synchronous FL, where the global model aggregation depends on the receiving of all the local model parameters of the participants. Previous works obtained fast convergence in such synchronous FL procedures, as they assumed all participants have sufficient resources (e.g., computation, bandwidth, memory). In consequence, even the slowest participant does not affect much the overall accuracy; and eventually, the model convergences. On the other hand, in asynchronous FL, the server performs aggregation whenever a model is received and may include a participant in the middle of the training phase. This approach enables scalability within the system and reduces the straggler impact, but cannot guarantee convergence. Sprague *et al.* [SJ+18] analyzed the issues of ensuring convergence of asynchronous FL but did not present a solution to overcome such issues. The authors in [XKG19] proposed asynchronous federated optimization, [SJ+18, CNDK19] discussed on the asynchronous FL for geospatial applications, [LHD+19b, WHL+19, LYRZ19] proposed a DP-based asynchronous FL strategy for MEC, and [ZLH+20] presented a blockchain-based secure data sharing strategy for asynchronous FL. Still, none of these works guarantee convergence during asynchronous FL communication of resource-constrained clients. Thus, formulating a method to ensure convergence in asynchronous FL can be a new research direction.

2.5.3 Quantification of Statistical Heterogeneity

FL training becomes complicated when the training data across devices are not identical in terms of data modeling and convergence behavior of the training process. Several ML works focused on the designing of statistical heterogeneity via meta-learning [VD02, T$^+$12], multi-task learning [Car97, E$^+$04], which are further extended to FL settings [SCST17, CLD$^+$18, ZLL$^+$18, C$^+$19, E$^+$19, KFBT19]. For instance, an optimization framework MOCHA [SCST17] allows for personalization through multi-task learning; however it considers convex objectives and is limited to its ability while scaling to massive networks. The authors in [C$^+$19] modeled a Bayesian network by performing variational inference during learning. Though their proposed approach can handle both convex and non-convex models, it encounters high cost while generalizing to large federated networks. Besides, the authors in [E$^+$19] aimed to identify cyclic patterns within data samples. While the client data tend to be heterogeneous in terms of the number of samples, dataset structure, and format in a non-IID setting, all existing works on FL adjust the statistical heterogeneity after the training phase begins. It impacts the training quality, and the lack of proper quantification of such heterogeneity can cause poor training performance. A local dissimilarity approach is proposed [ES10] to quantify statistical sample variation, where the resource-quantification starts after the training starts. The authors in [LW19] proposed a centralized approach of handling heterogeneity of FL model training but did not consider specific support and analysis for statistical heterogeneity. Li et al. [LSZ$^+$18] proposed a reparametrization of the FedAvg [MMR$^+$17] algorithm that can scale up divergent model updates and guarantees convergence while learning over statistical heterogeneous networks. However, they did not quantify the level of statistical heterogeneity while selecting clients during training or performing model aggregation.

2.5.4 Data Cleaning and Handling False Data Injection

In a real-world FL-based IoT environment, the IoT clients generate their models based on their extracted data. In a conventional FL-based IoT approach, there is no intermediate stage to refine the sensor data, which may cause a falsified local model with an erroneous update that eventually misleads the global model aggregation. As the number of such false data injected clients increases, the model accuracy reduces at the same phase. In time, it brings down the chance of reaching convergence. Bagdasaryan *et al.* [BVH+18] proposed a backdoor FL to identify malicious attacks during federated aggregation. They developed a train-and-scale scheme to restrict anomaly detectors from looking at the client's model weights or accuracy during FL tasks. The authors in [FYB18] explained the vulnerability of sybil attacks in the FL process. They proposed a defense mechanism that can identify poisoning sybils by analyzing the diversity of FL clients during model training. However, none of them considered real-time false-data injection onto the IoT clients, which needs further research.

2.5.5 Reducing Energy Consumption and On-device Model Training

In line with our previous discussion, the clients of the IoT domain may possess a weak-processing unit. Therefore, it is challenging to conduct inference, training on devices, and executing timely interaction with the server through an energy-efficient communication scheme. However, on-device training causes two problems. First, the generated on-device model size needs to be small enough so that it fits within the device memory and still captures most of the data complexity to compute an effective model. The on-device inference problems are solved in [KG+17] and [TB+19],

but the on-device training issues are not expounded. Second, the system can require high computational and storage availability for on-device training than these IoT clients can provide. Section 2.4.5 presented some approaches suitable for specialized neumorphic or field-programmable gate array (FPGA) hardware or miss the combative constraints observed in the FL-IoT domain. Figuring out the solution to this dual problem is paramount. A potential direction can be found in [TBG+19]. They proposed an IoT-based network architecture to enable creating high-capacity client models with 15-38x fewer parameters compared to the conventional model experienced for such applications. Kumar *et al.* [KG+17] proposed a tree-based approach to predict 2KB RAM IoT devices, e.g., Arduino Uno board that possessed 8-bit ATmega328P microcontroller without any floating-point support, and 32 KB size of the read-only flash. Their proposed algorithm attains standard prediction accuracy by constructing a tree model that shrinks the model size and reduces prediction costs. They learned a sparse tree with high-powered nodes, carrying out the tree's learning process through sparsely projecting data within a low-dimensional space, and collaboratively learning all projection parameters and trees. Investigating such architecture to enable learning within the resource-constrained FL environments is an unexplored domain.

As we discussed, FL needs on-device training; therefore, any research that can enable energy-efficient execution of ML algorithms can help FL in turn. If we consider resource-constrained nodes for an IoT environment, then prolonging battery-power life duration is a challenge. Due to repeated interactions with the server, the battery charge can be reduced significantly. Minimizing the depreciation of battery power while interacting with the server is challenging. [KG+17] developed a tree-based algorithm for making a prediction on resource-constrained IoT devices (i.e., Arduino) that possess only 2KB RAM. Still, they did not perform the training operation

on resource-constrained IoT nodes. [GS+17] designed a kNN-based algorithm that works on resource-scarce IoT nodes ($\leq$ 32kB RAM and 16MHz processor) to predict through supervised learning, but the edge devices are not trained locally. Therefore, it is essential to design an improved version of the FL algorithm that can handle small computational power as well as storage to train IoT nodes on edges, and how we can manage the energy consumption of client nodes during the training phase is also an open issue. An exciting direction in this front are dynamic computation technologies. Dynamic computation techniques activate only a part of the neural network for an input. This can help achieve both efficient training and inference as only a part of the neural network is updated for each input.

Beyond algorithm innovations, there has been a surge of work in the domain of design of software and hardware that executes ML algorithms efficiently. Here, efficiency refers to any or all of reduced energy consumption, faster runtime, and smaller memory footprint. The works in this area can be categorized in the domain of novel instructions for executing ML in CPU [LDT+16, SBB+17], design of specialized accelerators [CKES17, CDS+14, SMGK18], optimized software library [TDBM19, LSC18], development of new memory technologies [KOY+19] and near data processing to enable large storage using smaller energy budget [ISK+18]. However, the vast majority of these works are focused on the efficient inference of ML algorithms. A lot of these optimizations could be tailored to enable suitable training. Thus, further research in understanding the training algorithms of ML and how they execute on hardware can help tailor these solutions to solve this issue. The work described above modifies traditional hardware to make them amenable to machine learning. In traditional hardware, the unit responsible for processing information (processing unit) is separated from the unit responsible for storing data (memory). The instructions and data are fetched from memory and executed in the processing

unit. This is called the von Neumann architecture. Apart from this, there is an entire body of works in the domain of neuromorphic computing [SPP+17, KKC+16] dedicated to replicating the extreme power efficiency of the human brain by developing new hardware that mimics its synaptic structure. The main difference with traditional hardware lies in the non von Neumann architecture of these hardware as the processing and memory elements are not separate. We refer the readers to [SPP+17] to get a better overview of this field. FL can also benefit from edge units built using this neuromorphic hardware that can enable efficient on-device learning.

2.5.6 Managing Dropped Participants

Internet availability and network connection power are crucial, particularly while applying FL in an IoT environment. Any FL-IoT participant may go out of network in the middle of the training phase or during the interaction with the server due to mobility, bandwidth shortage, lack of transmission power, or out of battery life. Most of the recent works considered that all FL participants maintain a continuous connection with the server and cannot drop connection in the middle. In the real-world FL-IoT environment, such a scenario is not feasible, and any participant may go offline due to out-of-resources. Dropping off a significant portion of the participants would fail to generate an effective global model. It is difficult to understand whether a client gives a slow response because of the network issue or resource-shortage. Figuring out the potential problem may help us act according to the problem scenario. The authors in [DB19] presented a solution to handle straggler clients by acknowledging their resource utilization (i.e., computation power) after each local update. They formed a predictive model by analyzing the client's resource

utilization and adjusting local computation accordingly. Another strategy is to perform asynchronous training, i.e., updating the global model whenever it receives a model update from any of the participated clients [WTS+19, CSJ19]. Moreover, a recently invented FL framework, FedProx [LSZ+20] can handle heterogeneity in federated networks. FedProx allows a partial amount of work from each client device through a re-parameterization of the conventional FedAvg algorithm. However, when most of the clients within the network perform a low amount of partial works, their approach may take longer to reach convergence. The authors in [IA21] proposed activity and resource-aware FL strategy that can handle straggler issues by examining resource status, labeling clients with trust values in accordance to their contributions towards model convergence, and accepting variable works from the participated clients. However, further research needs to be conducted to optimize hyperparameters while enabling variable or partial works from the clients.

2.5.7 Privacy Preservation

The existing FL approaches aim to improve privacy by adapting classical cryptographic protocols and algorithms such as DP, SMC. An FL-based SMC protocol is proposed in [BEG+19a] to protect client model updates. Through this method, the server cannot see the local update parameter but can still extract some information by observing the aggregated results after each round. However, this approach encounters a high communication cost, which is not feasible for a resource-constrained FL-IoT environment. The authors in [GKN17, MRTZ17, ZZY+20] applied DP to FL to achieve a global DP, but the hyperparameters of these approaches affect the communication and model accuracy. An adaptive gradient clipping technique is presented in [TAM19] to handle this issue. In [BDF+18], a modified version of local

privacy is designed to limit the power of adversaries that guarantees more robust privacy than global privacy and results in better model accuracy. Another interesting approach to a DP mechanism based on meta-learning is proposed in [LKCT19] that can be used in FL through personalization. Besides, DP can be coupled with model compression strategies to reduce communication overhead and attain an improved version of privacy simultaneously [LSTS20, ASY+18]. Further, some prior works [QGL+20, LHD+19a, ZHS+20, LZJ+20] proposed mechanisms to preserve privacy in a blockchain-enabled FL-based IoT environment. However, most of these approaches did not consider the heterogeneous resources of clients. They did not analyze the feasibility of applying a robust privacy-preserving algorithm that can be adapted without a straggler effect. Further research needs to be conducted to obtain maximum privacy benefits for resource-constrained heterogeneous FL-based IoT environment.

2.6 Applications of Federated Learning

FL fits best in applications where we need to deal with sensitive information, and therefore, on-device training is more important than passing local data to the server. Most of the existing FL applications are based on labeled data collected from clients or user activities (e.g. type URLs or keyboard, click button). In this section, we discuss some existing FL applications to better understand the real-world impact of FL.

2.6.1 Resource-Sufficient Federated Learning Applications

Recommendation System: A recommendation system can be compared to an information filtering scheme that tries to predict user preference or rating for an

item. In the conventional recommendation system, user preference or rating would be shared with other users, and privacy is not maintained in many cases. Instead of sharing such private data, the authors in [CLD+18] proposed a federated meta-learning framework through which each local client shares his/her algorithm rather than his/her data or local model. In particular, federated meta-learning is useful when the model size is large; therefore, sharing algorithm is more flexible than sharing a model.

Next-word Prediction: A popular ML-based application is next-word prediction where a model is constructed that can predict what the next probable word would be. Such a centralized ML application may transfer private user data (e.g. SMS, URLs) to the server and may leak any sensitive information about the user. From that motivation, an on-device distributed ML-based framework is designed by [HRM+18], which is inspired by the FedAvg algorithm. They performed on-device model training of each of the participating devices and obtained a higher recall than the conventional FL-based approaches. In this way, FL helps a user by enabling his/her devices to make predictions by learning from his/her typing behaviors and indirectly reading the users' minds.

Keyword Spotting: Wake-word detector applications are prevalent nowadays. For instance, Amazon's 'Hey Alexa' wake-word detector is used to play different songs, or execute different commands, while Google's wake-word detector 'Hey Google' is used for different purposes including driving e.g. to get direction on a map. However, most of those applications are based on the cloud-based system and pass user data to the server. Unlike this, an embedded speech model is proposed [LCL+19], where they used a wake-word detector 'Hey Snips' to recognize user's voice. They used a crowd-sourced dataset and applied the FL strategy by keeping user information private.

Relevant Content Suggestions for On-Device Keyboard: Google has recently implemented a virtual keyboard application named Gboard, where they applied the FL strategy to suggest relevant content [YA+18]. It works on user-click or ignores situations that are stored in training cache and value is added when related contents are suggested. Based on user-click, the information is stored in the cache and feeds into the on-device training process. In this work, inference and training are performed on-device. Only the model updated parameters are shared with the server, while globally trained models are deployed on each client.

2.6.2 Resource-Constrained Federated Learning Applications

Smart Robotics: A lifelong reinforcement FL framework for mobile robots is proposed in [LWL19]. They designed an architecture to enhance navigation systems of mobile robots to learn efficiently from prior knowledge and adapt to a new environment effectively. They used two types of transfer learning for fast adaptation of the mobile robots within a new environment. Their proposed system is scalable but lacks security, privacy, robustness, and sparsification.

Smart Object Detection: The authors in [YL19] designed an approach of optimizing object detection by considering Kullback- Leibler divergence (KLD) during measurement of weights divergence of client's local models. They adapted the Abnormal Weight Suppression technique to reduce the effects of weight divergence that may be caused by unbalanced and non-IID data.

Smart Healthcare: In healthcare services, the FL-based IoT concept can be extremely effective to preserve the privacy of sensitive medical data. The IoT devices can be useful to generate data streams of patient's status, and FL can be used to

undertake early precautions or treatment utilizing the historical data. The authors in [YGX20] developed an FL framework for smart healthcare by applying the FL mechanism, and reduced computation load of IoT devices during training. Their proposed approach also took the edge of communication overhead during interaction of FL server and IoT devices. However, their developed framework does not guarantee convergence, and is incapable of performing a successful learning process in presence of malfunction or edge/cloud server failure.

On-device Ranking: Another application of FL is to rank a search result. For instance, if we query something in our device, an automated search result appeared. This is done by making an expensive call to the server. To reduce such cost, implementation of on-device training to generate a ranking of search results is proposed in [BEG+19a], which is particularly useful for resource-constrained devices. By observing the user's selected item from a ranked list, their system puts a label whenever a user interacts with the ranking feature. In this way, user preference is not revealed to anyone, and communication overhead is reduced by a significant margin.

Anomaly Detection: An autonomous self-learning scheme is proposed in [NMM+19] to identify compromised devices within IoT networks. Relying on unlabeled crowdsource data and depending on the device-type-specific behavior profiles, their proposed system can learn anomaly detection model without requiring any labeled data or human intervention to operate. They apply the FL strategy to aggregate behavior profiles for effective intrusion detection.

Resource-efficient Training of UAV-enabled IoT Devices: A particle swarm-based air quality monitoring framework is proposed in [LNL+20]. Their proposed system enables energy-efficient lightweight model training of Unmanned Aerial Vehicles (UAVs) using aerial haze images and predicts Air Quality Index (AQI) while preserving privacy. To sense ground systems, they proposed a Graph

CNN-based Long Short-Term Memory (LSTM) model for obtaining accurate and real-time AQI inference. Besides, the authors in [TZC+21] addressed the issue of reducing latency and improving the energy efficiency of UAV-enabled IoT devices by optimizing battery resources and wireless bandwidth. They employed a deep deterministic policy gradient (DDPG) strategy to evaluate their system cost.

2.7 Future Directions for FL Algorithms considering Resource-constrained Devices

As we discussed, FL is a recently invented distributed ML technique that can be considered as an emerging research area. After examining the core challenges of FL process while applying on resource-constrained IoT devices in Section IV, and analyzing some potential solutions of those emerging challenging in Section V, we point-out potential future directions in FL-based IoT environments. In this section, we highlight the future directions of this domain.

• In an FL-based IoT environment, it can be experienced that some clients possess more data (e.g., due to frequent use of a particular application, or having a greater memory size) compared to other clients within the underlying IoT network setting. Such a discrepancy in the number of data samples, particularly due to heterogeneous memory size and availability, leads to massive deviation in terms of training periods from participant-to-participant. The non-uniform data distribution raises issues in generating the representation of the population distribution of any client dataset. Handling such disparity within **local training dataset** needs further research.

• To ensure convergence for asynchronous learning in a Non-IID setting with a presence of resource-constrained devices, loss functions of the **non-convex problem** (i.e., an objective function that has multiple feasible regions, and each region has

multiple locally optimal points) need to be considered, and supportive algorithms should be proposed.

• In an FL system, we may need to choose a cluster head that would be responsible for passing the aggregated model parameter to the server for **energy efficiency**. The cluster head can collect client model parameters from its region in a synchronized fashion, while the central server can receive those locally aggregated models through a synchronous or asynchronous manner. Here, the leader can act as an intermediate aggregator and can avoid straggler nodes. It can marginally reduce power consumption and can minimize bandwidth requirements of the FL system, which could be effective for resource-constrained FL-based IoT environment. However, the leader node needs to be proficient to conduct swift operations and should be trustworthy to avoid false data injection. Thus, selection of effective leader nodes for managing the FL operations can be an interesting research direction.

• A **device-centric automatic wake-up mechanism** can be useful in determining the optimal period to carry out interaction with the server. Such an approach can reduce unnecessary communication with the server and avoid sending a model update when the client's local data does not change much. Besides, the automatic wake-up mechanism may help the resource-constrained devices to reserve energy, which could be utilized in further training.

• **Client mobility** can drastically change the overall system behavior. A network may hold a large number of active clients before the training starts, and after some period, most of them could go out of network. As a result, some areas may own a large number of clients, while some other regions may not be able to generate a feasible model due to a lack of active clients. Therefore, how to handle the mobility issues of the IoT devices and ensure successful federated model training is a potential research direction.

• During FL process, we may observe **statistical heterogeneity** among the client data. Such heterogeneity compels the clients to perform more interactions with the server, or with its neighbor nodes. The existing works did quantification of such statistical heterogeneity after initiating the training, which may cost extra resources and may have crucial impact on local training of the resource-constrained devices. Extensive research needs to be conducted to quantify the statistical heterogeneity even before the initialization of FL training to avoid idiosyncratic situations due to data sample variations.

• **Effective incentives mechanism design** is essential to encourage FL clients to share their model information. Some incentives or regulations schemes are implemented in blockchain [EGSVR16, ZLQ+20, ZN+15], and some incentive mechanisms are proposed for high-quality federated data [SE19, JF03]. Still, more extensive research needs to be conducted on incentives mechanism design to upgrade the effectiveness of FL. An example of such design is how game-theory models can be adapted in FLS or, in addition to the accuracy, what new benefit can be provided to the user to encourage them for joining in FL training. Besides, as the FL participants can be resource-bounded, or the participants can be business competitors, it is mandatory to design a strategy that divides the overall earnings to ensure the long-term engagement of the participants. Further, more focus needs to be placed on how to defend against adversarial attacks that try to collect the majority of the incentives.

• A **lightweight blockchain** framework for FL-IoT setting needs to be designed that can ensure robustness, enhance privacy and security while interacting with the server or neighbor clients. Blockchain can prevent model parameters or algorithm temperament and verify the model update and exchange. Some blockchain paradigm for on-device training is discussed in [KP+19, XCL20], but they designed

that framework without considering the challenges of the weak-processing unit and limited memory of IoT devices. Further research needs to be conducted on designing miner selection, block mining, consensus algorithm, validating a chain, atomicity, and blockchain interoperability, especially for FL with resource-constrained IoT clients.

• The FL structure leads us to think about integrating **trust model** to avoid adversarial clients during training. Selecting a client based on only resource availability would lead us to choose a malicious client. However, we can design a trust-based model based on the client's previous contribution to learning within the network and interacting with other clients. Typically, it is assumed that the server is trustworthy, and we can use the server to generate the trust model by analyzing the behavior of the clients. The incentive mechanism can be designed based on the generated trust model, and this may open us a new research direction.

2.8 Conclusion

This chapter presented a comprehensive survey on federated learning algorithms and analyzed the implementation challenges while performing on-device training. We particularly emphasized the issues of FL process while considering resource-constrained IoT devices as FL clients. Firstly, we presented a highlight over FL algorithms that can enable efficient and scalable model training in edge devices. Then, we presented an overview of FL taxonomy and analyzed the existing papers to distinguish our contribution as compared with prior surveys. We discussed distributed learning and optimization techniques and explained various aspects of distributed algorithms for decision-making purposes. We analyzed the challenges during the on-device learning of resource-constrained devices and discussed exist-

ing feasible solutions. After analyzing the challenges, we described the emerging challenges of FL implementation in resource-constrained IoT devices, which needs extensive further research. Afterwards, we explored the existing FL applications to provide a better understanding of the FL role in real-world applications. Finally, we listed the potential future directions of deploying FL within resource-constrained heterogeneous IoT environment.

According to the comprehensive literature survey that is provided in this chapter, we conclude that there are several research questions at the intersection of FL and resource-constrained IoT. The rest of this dissertation, will provide novel algorithms namely FedAR, FedPARL, and FedMDP, and their real-world applications in finance, IoT, robotics, and energy systems.

CHAPTER 3

DISTRIBUTED SENSING MECHANISM USING SMART END-USER DEVICES

[1] In the previous chapter, we presented a comprehensive analysis of applying Federated Learning (FL) in a resource-constrained environment. From our survey, we realized that, there was no existing work that show us a pathway to track and trigger distributed edge devices, and collect necessary information to prepare a federated dataset.

3.1 Abstract

There is a dearth of research in developing low-cost solutions for distributed decision making in IoT networks. Most studies in the literature require the deployment of additional sensors for data collection. In this study, we propose to leverage available sensors built-in smartphones, to properly collect and broadcast data for different decision-making purposes in smart cities infrastructures, for example, intelligent transportation networks, smart health services, security and emergencies, industrial control, smart agriculture, home automation and so on. To this end, we first introduce our new platform (including software and mobile app implementation) to identify available sensors at each end-user device. We have identified a wide range of sensors including gyroscope, ambient light sensor, temperature, magnetic field sensor, orientation sensor, game rotation vector, linear acceleration, relative humidity, gravity, geomagnetic rotation vector, etc. As the sensors are already integrated within the phone, therefore, using these sensors can be beneficial considering

[1]This chapter is an edited version of the author's previous work published in [IA19]

the complexity, efficiency, and cost of the overall system. The challenge is to design a system that can trigger distributed devices to be self-activated and agreed to generate all available sensors data. Besides, as devices can send a continuous stream of data, therefore, size of data could be mounted and could be in the haphazard structure, which would give us hurdles to identify a device sensor data from another and to make an intelligent decision. To tackle all of these, we propose a distributed sensing approach that is capable to identify a device using token, can activate distributed end-user devices to send data to the cloud whenever it requires and store data in the cloud server maintaining proper format. This approach enables remote data collection leveraging available end-user devices and reduces the cost of installing new sensors for autonomous IoT applications. We then build on our efficient sensing platform to enable distributed intelligence among a network of smart devices. To this end, we leverage the computational capacity of these devices for local decision making, i.e., instead of broadcasting all sensing information to a centralized agent and solve a large-scale decision-making problem, each smart device communicates with a limited set of neighboring devices. This will also pave the way for implementing federated learning as a promising solution for distributed decision making.

3.2 Introduction

3.2.1 Background and Literature Review

Smartphones, smart things, wireless and wired devices are some of the components of today's distributed networks that continuously generating streams of data. To store such huge data and consume minimal computational power, it is wise to pre-

serve those data locally and perform costly computation at the network edge. As we know, the sustainability of IoT depends on various factors e.g. power of connecting nodes, fault-tolerance, surrounding environment as well as the heterogeneity of the devices. One of the major concerns to prolong the IoT network is to use such a device that has better processing capability as well as battery longevity and IoT devices are mostly battery-powered [SG15]. Due to this, mobile crowdsensing has become an emerging field that opened the door of opportunity due to the prodigious growth of mobile devices, especially for IoT applications[GCZ+16]. Besides, today's smartphones are coupled with sensory components that enable us to collect data to understand the environment and crucial information about the user. The collection of such data generated from various devices can reveal significant information about the internet of thing enabled devices. Now, the problem is, as mobile crowdsourcing data is increased, it commenced communication overhead, processing delay, eventuates outburst of storage, leakage of security and privacy issues of data within the network. To overcome this, we propose intelligent sensing of distributed devices which helps us to show a pathway to federated learning for making intelligent decisions by avoiding communication overhead and considering the quality of sensor networks.

A ubiquitous learning environment to understand suitable learning contents and self−learning strategy, particularly for IoT devices, is proposed [XWC11]. Through that work, they disclosed how a device using RFID tags could be useful to communicate with a knowledge database to further take decisions for a similar environment. As in ubiquitous learning environment, the number of IoT devices might be huge as well as of many classes, thus for maintaining security and integrity, the author in [MBS+17] tried to separately identify IoT and non−IoT devices by analyzing traffic congestion within the network. Through their method, they distinguished

traffic generated by IoT and non-IoT devices by using HTTP packet property of the user, analyzed traffic and finally applied a machine-learning algorithm to make the decision.

But identifying IoT devices does not solve all problems as data retrieved from the edge device can be invalid or false. To automatically filter such data, a machine learning approach using the artificial neural network is proposed in [CS16] where device id, delay, and sensor value are used as neurons. These three parameters feed to the neural network and it filters inappropriate data. Additionally, as IoT data are increasing with respect to the time period of its extraction, different techniques on deep learning are studied in [VTWA18] and a solution using offline scheduling algorithm is proposed in [LO$^+$18] considering edge node's limited processing capabilities. But they did not implement their application examining real-life IoT environments. Moreover, a mobile crowdsensing IoT application is deployed in [ZLG$^+$18], where they focus on reducing latency and validation of sensor data. But they mentioned that the performance of their solution degrades when it is applied to the real world. Moreover, to handle huge amount of data streams in real-world, Information Flow of Things (IFoT) is propounded in [YYS16], where the open issues and challenges of real-time IoT data management in terms of networking and processing of big data and handling of various type of data in a unified manner are addressed, but they did not show how to make distributed decision after capturing real-time data streams. Several IoT based real-life application is proposed i.e. patient monitoring system [AH17] for observing congestive heart failure patients and sending an automatic alert to observation center, similar kind of centralized cloud-based health care data processing approach is also discussed in [HPS$^+$15]. Furthermore, the author in [AAS19] focused on the needs of sustainability and possible hurdles to build smart cities from the perspective of security and privacy concerns. To enable interoper-

ability amongst different communication network devices, an opportunistic mobile gateway for discovering IoT devices, control, and data management is discussed in [ACF+17]. But all these works can not process data and make the decision in a distributed fashion and hence may encounter communication overhead, high latency, and security issues.

The conventional federated learning resembles a global model which is formed by attaining data from a large number of devices. However, the authors in [LSTS20] aim to propose an approach in which a global model is generated by taking updates from the local device in a periodic manner i.e. the central server will devise a new updated model by learning the aggregated model of each individual devices which are communicating. They also pointed out some core challenges and ways to mitigate those issues, but, they did not verify their model by applying in real-life environments. The author in [CLD+18] showed a distributed meta federated learning technique in which the local devices send the algorithms of their model to the global server instead of sending data or models. But they did not consider periodical updates to the local server to reduce communication overhead and make the minimal number of communication rounds to perform while sending algorithms. The works of [LPH17] considered the issue to reduce communication overhead through making updates in the local devices and performing multitask learning in the central server. But they assumed that the same amount of work will be performed in each distributed device which is evidently not possible because of the system and statistical heterogeneity. To improve communication efficiency, federated learning-based works are studied in [KMY+16] and [KMR15], but they proposed a single global model which is trained by capturing data from the local device within the network which arises processing delay.

3.2.2 Goals and Contributions

In this dissertation chapter, we introduced a novel distributed intelligent sensing leveraging end-user devices. To achieve this, we first develop a platform to identify available sensors in each device, e.g., smartphone. We then explain federated learning that can potentially leverage our sensing platform to enable distributed decision making and learning over a network of IoT devices.

3.3 Distributed Efficient Data Sensing Leveraging Smart Devices

Smartphone device is considered to be an effective tool for information extraction as it is integrated with various type of sensors and has the capability to send data wirelessly which may have a huge impact, particularly for Internet of Things (IoT), enabled applications. Mobile Sensor API helps us to understand the available sensors within the device and gives us the accessibility to fetch those sensor data in real-time. But, if we want to identify each individual device and wish to retrieve sensor data remotely, then we need a mechanism through which each individual device can be handled prudently and devices will be triggered whenever we want to fetch data. To accomplish our goal of sensing distributed data within the smartphone devices, we propose a distributed intelligent sensing approach through which data can be fetched at any time in a distributed way and no extra sensors need to be installed within the devices. Our goal to accomplish distributed intelligent sensing can be divided into a couple of stages: **1)** Discovering the available sensors within a smartphone and **2)** Registering each device to cloud and generate a unique token for each individual device, **3)** Constructing a push notification feature which enables

a device to be triggered on receiving a notification message, **4)** On receiving a notification message, requested device is supposed to pass its available sensor data to the cloud server, and **5)** Storing data in the cloud by maintaining proper structure so that it can generate a prepared dataset based on real-time data.

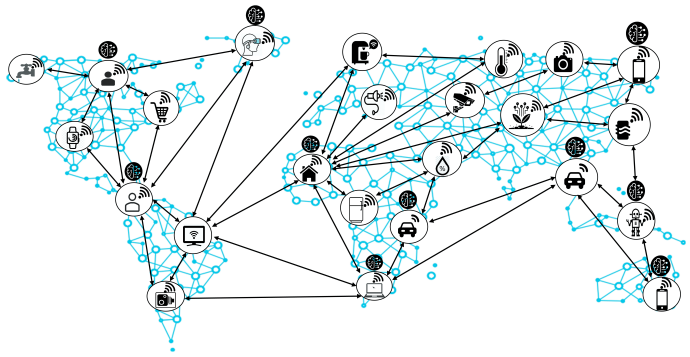
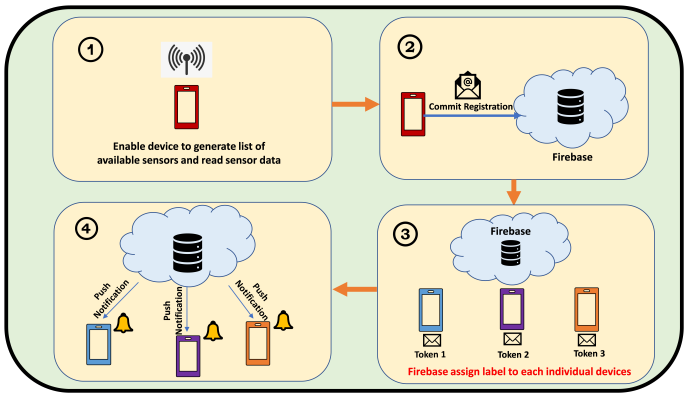


Figure 3.1: Schematic overview of IoT networks.

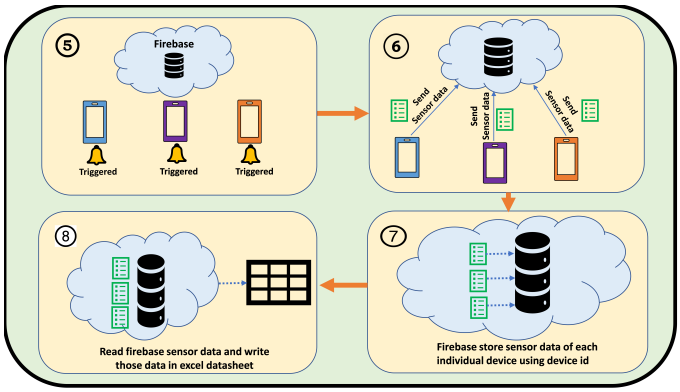
3.3.1 System architecture

To achieve the target of intelligent sensing, first of all, we need to enable our mobile device to compute the list of available sensors and should be able to read those sensor data. To fetch data from distributed devices, therefore, to discover knowledge from the retrieved data, we need to track which device our data is coming from. To incorporate this functionality, we launch a registration process whenever a user going to use the application for the first time after installation. The user needs to provide an email address and password and from this information, we generated a token. All this information we store in the cloud with a device id so that we can easily detect the device from the cloud. After storing the token, we are capable to send push notification to the devices and they will be triggered upon arrival of the notification message. Immediately after being triggered, the devices will start to send the sensor data in real-time to the cloud and it will store those data in a parent-child format of a

tree. Those stored data can play a significant role in understanding each distributed device's current domain and those data can be analyzed in a distributed fashion. In fig. 3.2a, we presented the steps from enabling end-user devices of sensing data until generating push notification and in fig. 3.2b, we displayed the steps from triggering an end-user device until collecting data for applying in the learning model.



(a)



(b)

Figure 3.2: Identification of sensors and leveraging intelligent sensing in end-user devices.

3.3.2 Steps of Intelligent Sensing Leveraging Distributed Devices

Device Registration and token generation

At the starting of our application, the user needs to provide its username and a password to sign in and based on this, we generate a unique token for each individual device. This unique token will further be used to separate a device from others. This token is just like a label which helps us to track a device and forward a push notification without any hassle.

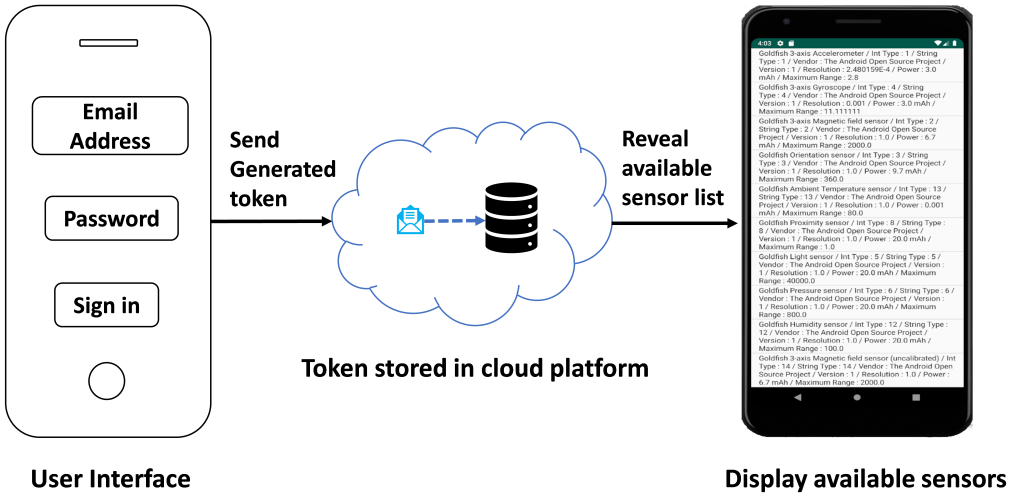


Figure 3.3: Registration and generation of token for each end-user device.

Identification of the list of available sensors within a device

After completing the registration within the application, the user will be able to see the list of available sensors that are identified. The available sensors can be revealed using SensorManager object. We used a method $publicList < Sensor > getSensorList$ and $Sensor.TYPE\_ALL$ [Dev17] to attain all available sensor list from an android device.

Construct a push notification service from server to client

We incorporate a cloud messaging service which helps us to send a push notification to a particular device or a group of devices for collecting real-time data whenever we want. During registration, as we are generating each device's token which is kept in cloud, thus using that unique identity, we will be able to send a push notification through the cloud messaging service which is shown in figure 3.4.

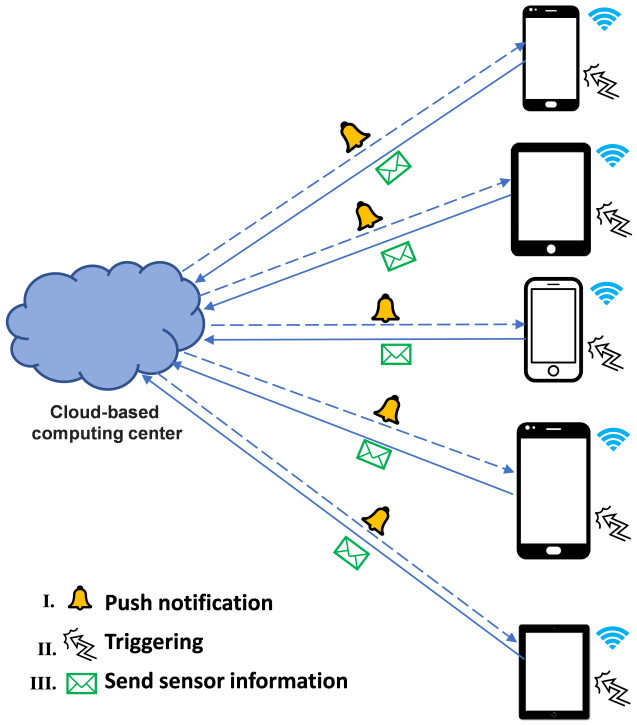


Figure 3.4: Remotely fetch data from a sensing device though our developed distributed sensing mechanism.

Trigger registered device on receiving notification message

After sending a notification, the device needs to understand the arrival of that message. We infused functionalities to each of the device so that it gets triggered upon arrival of a particular push notification from the server. For displaying the

notification on top of the screen, we need to extract the title and body of that notification message and store those. Using *NotificationManagerCompat* instance [Dev17] and *pendingIntent()* method [Dev17] (which provides permission to a foreign application to utilize in the user's application), we able to display notification over the top of our mobile screen.

Generating available sensor list and read current data of each sensor

After being triggered, the device needs to generate all available sensors and sense all the sensor's current values. We used SensorManager to get the context of sensor service. Within getFaultSensor() method [Dev17], we specify the type of sensors we needs to handle and using registerListener() method [Dev17], we activate the sensors to retrieve available sensor data.

Sending data to cloud and store in realtime database

The next step is to send the extracted data in google firebase cloud and store those in real-time. For this, we need to provide permission so that the user can store data whenever they get a notification message from the server. We need *DatabaseReference* instance [Sto16] which helps the application to find the instance as well as the reference to store data in a perfect way. As we are sending sensor data which is continuous and show values whenever sensor data is changed. But, it is not feasible to send data infinite times, we need a predefined number of data that should be sent upon arrival of the notification message. We handled it using *unregisterListerner* method [Dev17] which is supposed to be active after a certain number of data being sent. After we get the data in firebase, we can export it in JSON format which is shown in Fig. 3.7.

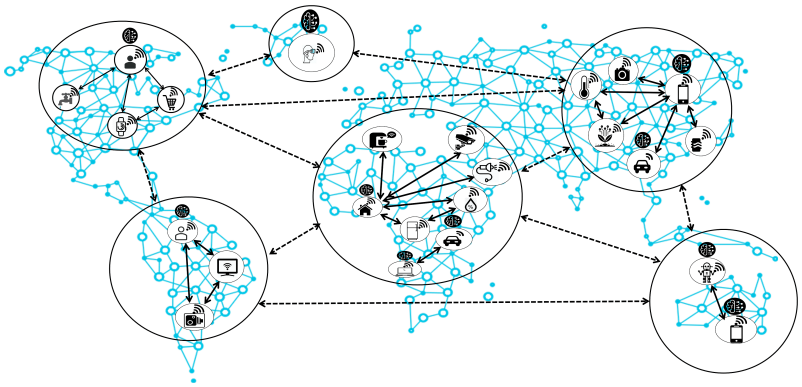


Figure 3.5: Distributed intelligent interaction leveraging end-user devices.

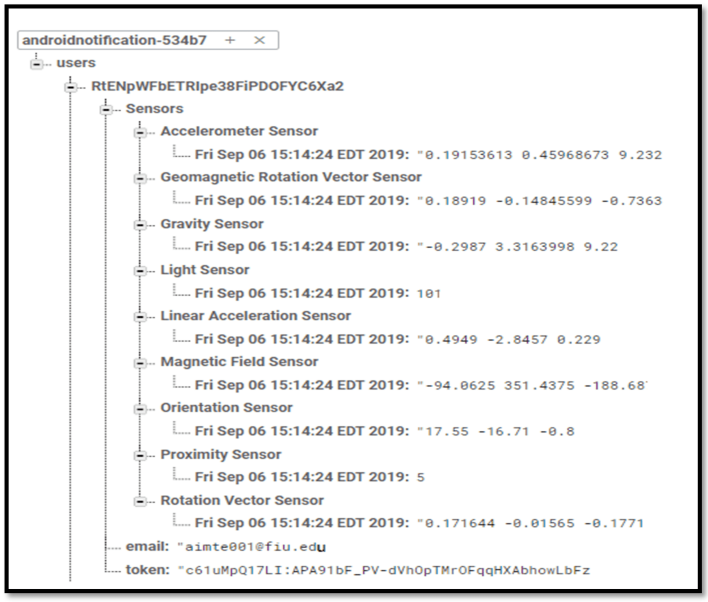


Figure 3.6: Expanded view of fetched sensor readings stored in Firebase Cloud.

3.4 Federated Learning Approach for Remote Sensing

The conventional federated learning resembles a global model that is formed by attaining data from a large number of devices. The author in [LSTS20] aims to propose an approach in which a single, global model is generated by taking updates from the local device in a periodic fashion i.e. the central server devises a new updated model by learning the aggregated model of each individual devices which are communicating. To solve the distributed optimization problem, we can consider four

Figure 3.7: Exported sensor data in JSON format attained from firebase cloud.

major challenges. The first one is, ***expensive communication***, which includes how to achieve our target model by sending a minimal number of updates and how to reduce the size of messages while transmitting to the central server [KMY+16]. The second one is, ***heterogeneity of the systems*** which occurs when we have variance capabilities of nodes with respect to computational power, memory storage, and communication range [SCST17]. Due to this, any device can become straggler or inactive or there may be a possibility that only a few of the existing devices are active to maintain communication with the server. So, it is required that the system can tolerate heterogeneous devices, can generate effective models in spite of having low participation of devices and have fault tolerance in case of having straggler or inactive devices within the network. The third one is, ***Statistical heterogeneity***[SCST17], which means that the samples or data which are collected or stored are not similar. In such a situation, it will be difficult to formulate a global model after getting trained from the local devices. The fourth one is, ***Privacy concerns***

[BIK+17], which is very evident to arise in federated learning as the local devices are sharing their model updates periodically. Though the devices are not supposed to share raw data to the central server, yet it is possible to extract sensitive knowledge by observing the whole communication process. In such a case, differential privacy and multiparty computation can be a solution but there are trade-offs between system overall prediction accuracy and efficiency. In this section of our dissertation chapter, we focus on explaining federated learning that can potentially leverage our sensing platform to enable distributed decision making and learning over an IoT network.

3.4.1 Infusing redundancy amongst the node dataset

Communication overhead is a major challenge that impedes scalability especially in IoT based distributed applications. To mitigate the iteration round during communication, speeding up distributed stochastic gradient descent by infusing redundancy among the worker node's training data could be a good idea [HK+19]. As feeding redundancy to the worker nodes decreases the residual error, which eventually helps to reach the target accuracy by computing fewer number of communication rounds and increases fault tolerance. For this purpose, our real-time dataset (which is formulated in the previous section) can be partitioned into a number of shards or chunks and can be distributed among the computing nodes. Each mini-batch created for a dataset generates individual gradient descent which is obviously faster than batch gradient descent where a single gradient is generated for the whole dataset in each epoch. Though it is faster than batch gradient descent, still in synchronous gradient descent (SGD) each node send their update within the network after every iteration to ensure that each node aware of the updated structure. This incurs

communication overhead in terms of communication round and the number of bits passed during communication. Besides in SGD, as each computational node send the updated model which is only based on their respective mini-batch not on whole data, so it may suffer from residual error if we consider full synchronous SGD. So redundancy infused SGD can be applied where each node can have a chunk of a dataset and can access a partial chunk of other mini-batch but not the whole data from a mini-batch. The author in [HK$^+$19] proposed RI$-$SGD, which works on local update and periodically generating the average of the models. The process of infusing redundancy improves gradient diversity, reduces variance in local update and communication round, and results in faster convergence. Populating the shard of each mini-batch by some partial samples of data that is stored locally does not hampers the overall performance as the main bottleneck of the system is communication rather than computation. Moreover, by reducing the size of mini-batch also outperforms in terms of accuracy compared to conventional SGD according to [HK$^+$19]. Another concern is that, as each worker node exchanges their model with each other at every iteration, then the availability of straggler nodes can encounter large processing delay. We can simply ignore them using RI-SGD algorithm and ensures robustness to node failure. In summary, we can reduce communication round during the interaction of nodes by inundating data redundancy for distributed SGD to attain faster convergence and low residual error. Though the redundancy arises some computational cost, it is worth of; as in return, we can get higher diversity which proceeds to faster convergence.

3.4.2 Quantify the impact of wireless factors on FL

For quantifying the impact of wireless factors on FL, we first focused on the challenges of FL from the perspective of wireless networks. For such a scenario, the nodes within the networks generate a federated learning model or algorithm which is sent to the base stations (BS) and this algorithm is computed based on the data behavior of the user nodes [CYS+19]. After that, the base station will generate a global model that will be transmitted back to the user nodes and all receiver nodes will be trained by learning that model. But, this training process of the nodes can be affected due to the wireless transmission quality and limited bandwidth of the nodes. As the wireless resource and bandwidth limitation are the major concerns, so the base station needs to select a suitable subset of user nodes which should be participated in sending the local model to BS and being trained by the global model. To attain an optimal result for such an environment, we need to formulate an approach through which limited wireless resource can be utilized properly and the subset of user can be chosen prudently which leads us to obtain minimization of the federated learning loss function, hence, increase the overall performance of federated learning algorithm. We need to devise an expression that can be imposed to attain a faster convergence rate considering the influence of wireless networks on federated learning and helps us to figure out the optimal power required to transmit algorithm to BS for each individual device. This optimal power is measured by selecting a subset of user and resource block which is transmitted to BS. In the end, the transmitted resource block and the subset of the user are optimized to obtain minimal FL loss function.

3.5 Conclusion

This chapter explained the foundation of sensing mechanism to trigger and collect information from distributed end-user nodes in a federated learning environment for effective communication among the sensor nodes. To devise this, initially we presented a novel approach of distributed intelligent sensing leveraging smart end-user nodes which can sense environment and communicate with a central server by sending their asset. After that, we explained federated learning that can be incorporated with our distributed intelligent sensing approach to attain optimized outcome while learning over IoT network and making distributed decision. In next chapter, we presented our novel federated learning algorithms that could be effective for resource-constrained heterogeneous IoT environments.

CHAPTER 4

FedAR: ACTIVITY AND RESOURCE-AWARE FEDERATED LEARNING MODEL FOR DISTRIBUTED MOBILE ROBOTS

[1] In the previous chapter, we presented our distributed sensing mechanism that can be effective in remotely triggering any of the edge devices, collecting information, and preparing a federated dataset. This chapter presents our novel activity and resource-aware Federated Learning (FL) algorithm that is suitable for unreliable and resource-constrained learning environments.

4.1 Abstract

Smartphones, autonomous vehicles, and the Internet-of-things (IoT) devices are considered the primary data source for a distributed network. Due to a revolutionary breakthrough in internet availability and continuous improvement of the IoT device's capabilities, it is desirable to store data locally and perform computation at the edge, as opposed to sharing all local information with a centralized computation agent. *Federated Learning (FL)* paves the path toward preserving data privacy, performing distributed learning, and reducing communication overhead in large-scale machine learning (ML) problems. We developed a novel activity-aware and resource-aware FL model that can mitigate the straggler effects and can avoid the untrustworthy agents during FL training. We consider a distributed mobile robot as an FL client with resource limitations either in memory, bandwidth, processor, or battery life. We consider such mobile robots as FL clients to understand their resource-constrained behavior in a real-world setting. We consider an FL client to be untrustworthy if

[1]This chapter is an edited version of the author's previous work published in [IA20]

the client infuses incorrect models or repeatedly gives slow responses during the FL process. After disregarding the ineffective and unreliable clients, we perform local training on the selected FL clients. To further reduce the straggler issue, we enable an asynchronous FL mechanism by performing aggregation on the FL server without waiting for a long period to receive a particular client's response.

4.2 Introduction

4.2.1 Motivation

The ubiquitous nature of IoT devices causes huge data streams due to their widespread applications. Storing and processing such vast amounts of data in a centralized location is costly, highly insecure, and time-consuming. Further, in some cases, the nature of local data is sensitive and requires further security measures to ensure user confidentiality and data privacy while exchanging data for computation and decision-making purposes. Sensitive data such as captured images, browsing history, personal information, and location-based services can be utilized for recommendations, social advertising, prediction, or incurring any potential privacy risks. Such sensitive information is unsafe to share with the server if potential privacy issues are not checked [YLCT19a]. Besides, as privacy preservation awareness is rising, legal laws and restrictions, i.e., General Data Protection Regulation (GDPR), are becoming prominent, which reduce the feasibility of data aggregation [LYY20]. The conventional centralized ML techniques can not be implemented in a distributed fashion due to infrastructure fallibility, including intermittent or weak network connectivity, limited bandwidth, or response delay [LO$^+$18]. To tackle the above-mentioned

issues, an alternative distributed ML paradigm named *Federated Learning (FL)* is proposed in [MMR+17] that performs on-device training based on the client's local data, pushes the client's training parameters at the edge, and learns from the global model. The popularity of FL applications is increasing due to their high acceptability, particularly in improving user privacy. The FL process enables network clients to generate a joint ML model that enhances privacy by not exposing any clients' private data. Another important factor of FL is that it works with nonindependent and identically distributed (non-IID) data samples observed in real-world settings [MMRyA16, BIK+17]. That means the FL algorithm is capable of handling the changes over time in the distribution of collected data samples. However, during an FL process, the client selection is crucial as a straggler, and an unreliable client can retard the learning process and prolong the model convergence time. Any client that is selected for a training phase is called a participant. A participant may turn into a straggler if the participant is underpowered in system requirements for a particular model training. Most of the existing FL approach avoids the straggler issue by assuming all the FL clients as proficient nodes and randomly selecting clients for the training rounds. However, the presence of such straggler clients has a vital impact on the overall performance of the learning model [ITW+20]. Further, there is a possibility of selecting a vulnerable FL client, specifically in an FL-based IoT (FL-IoT) environment, where the devices are more prone to susceptibility. Therefore, we need to monitor the clients' available resources, behaviors, and contributions toward training a learning model.

In our proposed approach, we consider mobile robots integrated with comparatively low processing units, limited memory, bandwidth, and battery power instead of assuming proficient clients with sufficient resource availability. We give instruc-

tions to each registered mobile robot for collecting data samples. We track the robot clients through remote sensing mechanisms, and each client gets triggered upon receiving a particular notification from the server [IA19]. Before executing a training round, we check the resource availability following the system requirement for model training and examine each interested client's trust score, which is updated based on their previous training performance. By enabling asynchronous FL, we further reduce the straggler effect by ignoring the unresponsive clients.

4.2.2 Literature Reviews

The FL research domain is increasingly evolving due to its capability to handle non-IID data, preserve privacy, and reduce communication overhead. Prior works from various disciplines, including distributed systems, ML, databases, cryptography, and data mining, focused on improving the FL method. An overview of the FL system design is presented in [BEG+19a], where they discussed the complexity of FL design by posing challenges to FL implementation and experimental evaluation. In consequence, some open-source frameworks are proposed to handle FL mechanisms such as TensorFlow Federated (TFF) designed by Google [AAB+15], Federated AI Technology Enabler (FATE) from WeBank [WeB18], LEAF [CWL+18], and PySyft [RTD+18].

FL strategy fits best in such applications, where we need to perform model training using sensitive data. Hence, on-device training is preferable rather than passing data to the cloud. The authors in [HY+18] proposed an FL-based model training for predicting heart attack in a patient. By configuring the necessary setup and integrating wearable devices, they observed the patient's health data and performed FL-based on-device model training. Besides, a Federated Transfer Learning (FTL)

framework is presented in [YLCT19a]. In that chapter, they considered each hospital as an FL client and performed collaboration among neighboring hospitals to improve the FL model. In [GWW+20], FL-based privacy-preserving named entity recognition (NER) is implemented that can identify different medicinal attributes (e.g., drug names, their reactions, and symptoms) by analyzing medical texts and performing classification.

Device-centric application data is utilized to build a recommendation system that aims to predict a rating for an item or user preference. For such a recommendation system, usually, one user's data is shared with other users, and privacy is violated in many cases. A federated meta-learning-based recommendation system is proposed in [CLD+18] considering the privacy issues. Each FL client shares its generated algorithm with the server to carry out an effective global model. Similarly, privacy-preserving news recommendation system [QWW+20] and personalized recommendation system [AudIK+19] are constructed based on the FL concept. Another interesting application of FL is user-typing-behavior-based next-word prediction designed by [HRM+18]. Their application can read a user's mind while typing by analyzing the user's messages and browsing history. A similar type of application to predict emoji on a mobile keyboard is presented in [RMRB19]. Moreover, wake-word detector-based applications are prevalent nowadays that consider user voice or speech for training an FL model that can recognize the user's command. The author in [LCL+19] designed such an FL-based wake-word detector application by using a crowd-sourced dataset that does not expose a user's voice during training. The ranking search result is another recent application of FL [BEG+19a, HSK+19], where the user query and preference are not shared with any other entity, and the search result is generated based on FL training. A content suggestion framework is designed on the concept of user activities, i.e., clicking or ignoring content, which

is discussed in [YA+18]. This application tracks and stores the user-click information on content suggestions, and constructs a model accordingly. Both training and inference are performed on-device in their work, and only the model parameters are dispatched to the server. However, most of the existing FL-based applications assumed that the FL clients are resource-boundless and trustworthy, i.e., any FL client can be selected for the training phase. However, if we consider a real-life FL-IoT setting, the FL client is often resource-bounded and prone to vulnerability. This chapter aims to consider both resource-constrained and reliability issues in an FL environment. According to the best of our knowledge, there is no existing work that examines the resource status and scrutinizes the client activity within an FL environment.

4.2.3 Contributions

The main contributions of this chapter can be listed as follows:

- We propose a novel model capable of giving a reward or a punishment to each participating FL client based on their performance during a training period.
- We propose a strategy to choose only capable clients before starting the FL training phase.
- We consider mobile robots to collect federated data in a distributed fashion and perform FL training simulation for a resource-constrained IoT environment.
- We consider an asynchronous FL scheme to mitigate the straggler effect on a resource-constrained FL-IoT environment.

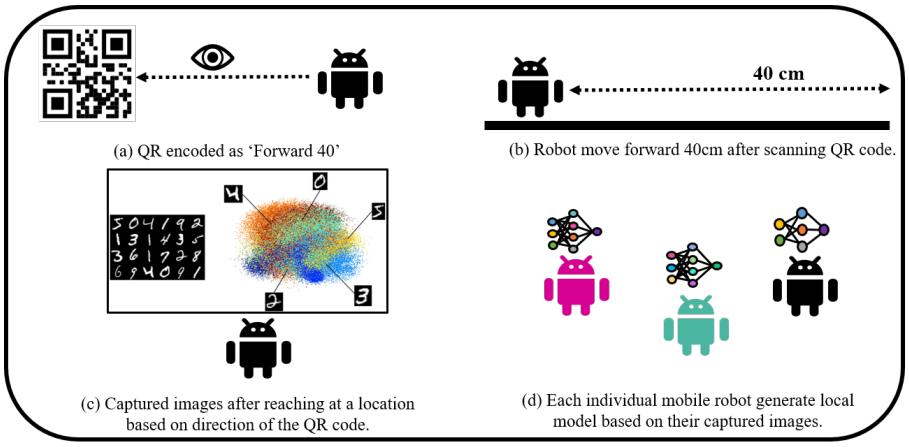


Figure 4.1: Real-time data collection procedure in an Federated Learning environment based on our proposed distributed sensing mechanism.

4.3 System Description

In this chapter, we assume that we have a resource-constrained FL environment, where a client may have a straggler effect, and any interested client may provide inappropriate model information during the training process. The main reason of the straggler effect is the resource shortage issue that leads us to think about a strategy to avoid an unreliable or inefficient client during the learning process. Moreover, the existence of unreliable or inconsistent clients can prolong the convergence time and may create a negative impact on overall model accuracy. To understand the resource-constrained FL behavior, we focus on collecting real-time federated data collection, handling resource heterogeneity, and monitoring client activity during an FL process.

4.3.1 Real-time Federated Dataset Preparation through Distributed Sensing

In a federated dataset, we need to have a column of client id that represents which client possesses the corresponding row information. To collect federated data using distributed mobile robots (clients), we need to send a set of instructions to those clients. One naive strategy is to collect data by controlling the robots remotely. However, such a strategy is not convenient if we have many clients, and we need to keep track of each client's information. We design an approach for controlling mobile robots through which any robot can be tracked and acknowledged about required instructions in a convenient way. In this approach, any new client interested in joining an FL network needs to commit to registration by providing their credentials. We generate a token and store the generated token in our server. A token is a unique identifier that helps to recognize each robot client in a distinguishable way. The token can be used as the client id in our federated dataset. More details can be found in our prior work [IA19]. We infuse functionalities to each robot so that they can identify a push notification when it is sent from the server. The push notification is used to trigger a robot and pass the commands as a Quick Response (QR) code. We use QR codes to hide raw instructions. We enable a QR code scanning mechanism for each mobile robot to read the instructions given as a form of a QR code and understand the meaning of the QR code. After successfully scanning the QR code, a robot client can get the instructions in a form of an array. The first element of the resulting array is the key, and the following elements are the parameters. For instance, an instruction of $Forward$ 40 10 0 means move forward with speed $40ms^{-1}$ for 10 seconds with an angle of 0 $degree$. In case we have multiple instructions within a QR code, we split those by a colon separator. By reaching the desired place, the

robot can capture images or can sense the environment. After that, the collected data are properly labeled. Each distributed robot can train itself by the labeled data, and generate a local model that is shared with the server. In Figure **4.1**, we presented an IoT based data collection overview for an FL environment.

4.3.2 Construction of a Trust and Resource-aware Framework

◇ Step 1: Publish FL Task

An FL process can be considered a monopoly market, where a task publisher acts as a monopolist operator and a set of mobile robots $\mathcal{N} = \{1, \ldots, N\}$ act as the clients. Each client $n \in \mathcal{N}$ uses its local training dataset of size s_n for being a part of the FL task. Each of the training data has an input-output pair, where the input is a vector that contains data sample features, and the output indicates a label for each input vector. The task publisher broadcasts an FL task with minimum resource requirements (e.g., memory, battery, and processing power) and a minimum trust score to qualify for the task participation.

◇ Step 2: Check Resource Availability of Interested Clients

Upon receiving information about interested clients' resource availability, the server filters out the candidates that do not meet the task requirement. Only the robot that satisfies the requirements has a chance to join the training phase. In Algorithm **5**, we have a function *CheckResource* that takes an input of memory ($\mathcal{M}$), bandwidth ($\mathcal{B}$), and battery life ($\mathcal{E}$) in line **14**. We generate resource availability for the three give inputs (line **15**) and compare with the task requirement $\mathcal{L}_{Req}$ (line **16**). If

available resources satisfy the task requirement, then we add that client to a list RA (line **17-18**).

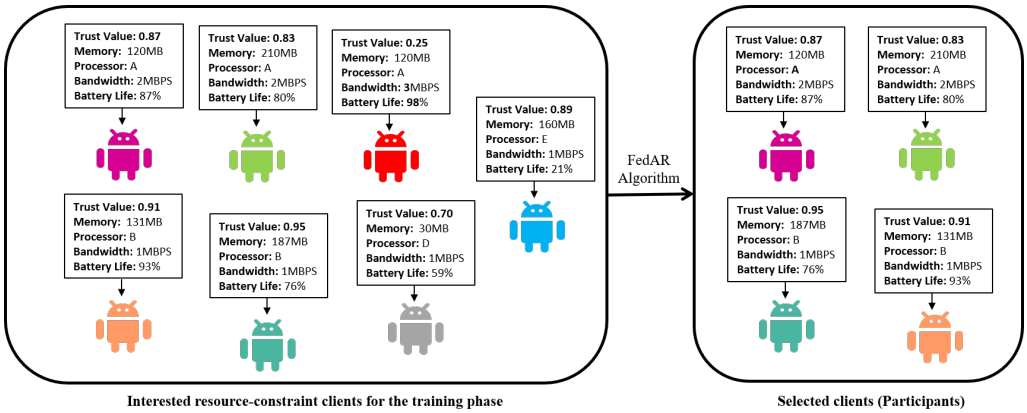


Figure 4.2: Selection of proficient and trustworthy clients for training phase applying Activity and Resource-aware Federated Learning (FedAR) algorithm.

◇ Step 3: Calculate Trust of FL-client

In our proposed FedAR model, we choose a fraction of clients that may change at each training round according to resource availability (e.g., memory availability) and the client's updated trust value. To construct a trust model, we pass the iteration number/communication round (i), client id (m), global model parameters (w_i), threshold time t for sending back local model, and model deviation γ. Each FL participant needs to send back their local model parameter within a given period of t that is set by the task publisher. However, we may encounter various situations (e.g., fast or slow response) during the training process and may receive different FL participants' responses (e.g., inappropriate model information). If a client joins the FL network for the first time, we set its trust score as $C_m = 50$. We add a trust score $C_{Interested}$ with a value of 1 to each of the clients interested in being a part of a training phase, who successfully meets the trust and resource requirement for that task, but could not join the training round. We provide $C_{Interested}$ to encourage the

trustworthy and capable candidate to participate in a future task. After selecting the FL client, we reward each client that accomplishes a task, and if it fails, we decrease its trust value as a punishment. Every participant client in a training round must submit their model within a given time because it is not feasible for the server to wait for a client for an infinite amount of time. The task publisher can set the threshold time for a task. In case an FL participant gives responses within the desired period, we provide a reward to that client (C_{Reward} with a trust value of 8). On the other hand, when an FL client fails to accomplish its task on time, we set a penalty on that client's trust score ($C_{Penalty}$ with a trust value of -2). We check the past performance of that client, and if the client repeatedly failed to respond on time ($20\% - 50\%$ of its participation), then we add a C_{Blame} trust value to that client's trust score (where $C_{Blame} = -8$). In case the client's straggling effect is observed above 50% of its overall participation, or if the client sends a model that has a high deviation to compare to the other client's models, we add a C_{Ban} trust value to that client's trust score (where $C_{Blame} = -16$). In Table **4.1**, we presented the assigned trust score of different events.

Table 4.1: Trust values considering different factors in an FL environment.

Factor	**Value**
$C_{initial}$	50
C_{Reward}	8
$C_{Interested}$	1
$C_{Penalty}$	-2
C_{Blame}	-8
C_{Ban}	-16

The details of our trust and resource-aware model are presented in Algorithm **5**. In line **1**, we receive communication round as i, client id as m, global model parameter as w_i, maximum time t to finish that task, and threshold of model diver-

sity γ. The threshold time to perform a task can be changed in different iterations by the task publisher based on the client's performance. We also do not consider a fixed threshold because, in the initial period of training, a model deviation can be larger for all clients than the global model, while after some iteration, it is supposed to get reduced to some extent. After sending all the function parameters, line **2-4** checks whether a client sends back its optimized local model within a preset time t. If the client sends its optimized local model within time, then we set that client's unsuccessful record (U_m^i) for that training round as 0 and give that client a reward score (C_{Reward}), which is added to the client's trust score. In case a client can not send back its local model within time t, we set that client's unsuccessful record (U_m^i) for that training round as 1 (line **6**). We check how frequently that client encounters unsuccessful or ineffective local model generation. If that unsuccessful record indicates below 20% successful task completion of the overall client participation history, then we add a penalty score ($C_{Penalty}$) to that client as a sign of warning (line **7-8**). In case we get more than 20%, but less than 50% unsuccessful record history for a participated client, then we set a blame score (C_{Blame}) to the existing trust score of that client (line **9-10**). Moreover, if a client is responsible for giving a slow response for more than 50% of its overall task participation history or sending back a model with large diversity compared to other client's local models, we assign a Ban score (C_{Ban}) to the existing trust score of that client (line **11-12**). Finally, in line **13**, we append the updated trust score of each client to a list.

◇ Step 4: Select Client for Training Phase

In a typical FL scenario, we may need to deal with many user nodes; however, only a small portion of them may be available at a specific time to perform training sessions. For instance, if the client nodes are mobile robots, they can be considered

Algorithm 5: Activity and Resource-Aware Model. The global model of i^{th} training round is represented by G^i, D^i_m indicates the local model of a mobile robot m on i^{th} iteration, unsuccessful record of a client m on iteration i is denoted by $(U^i_m.)$, trust score C_m for m^{th} client, t represents timeout, and γ indicates deviation.

UpdateTrustScore (i, m, w_i, t, γ)**:**
if m *sends model* w^i *within* t **then**
    set $U^i_{ID(m)} = 0$
    set $C_m = C_m + C_{Reward}$
else
    set $U^i_{ID(m)} = 1$
    if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{m}}^{\mathrm{p}} < 0.2$ **then**
        set $T^i_{ID(m)} = C_{Penalty}$
    if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{m}}^{\mathrm{p}} < 0.5$ *and* $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{m}}^{\mathrm{p}} \geq 0.2$ **then**
        set $C_m = C_m + C_{Blame}$
    else if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{m}}^{\mathrm{p}} \geq 0.5$ *or* G^i-$D^i_m > \gamma$ **then**
        set $C_m = C_m + C_{Ban}$
Append C_m to $TrustList$
CheckResource $(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$**:**
Resource availability, $\mathcal{R}_m = f(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$
Compare $\mathcal{R}_m$ with $\mathcal{L}_{Req}$
if $\mathcal{R}_m$ *satisfied* $\mathcal{L}_{Req}$ **then**
    Add $\mathcal{R}_m$ to RA list
Return C and RA

for training rounds only when charged and active. In our FL-IoT environment, all client data are locally available, and we need to choose a subset of clients to participate in the training process. This subset of clients is changed in each round, considering the trust score and resource availability. After calculating the trust value and checking out resource availability, we select the eligible candidates as participants. Each participant generates their local optimal decisions after giving consent to participate in a task according to its resource conditions and local dataset content. The effectiveness of the local model update directly depends on the local dataset accuracy [SFYB18]. After selecting the clients, we can reuse them again and again until our target model reaches convergence. In Figure **4.2**, we presented the client selection process of our proposed FedAR method.

◇ Step 5: Creating Model with Forward Pass Method

Initially, we prepared input samples of client images to be grouped as batches. We identify the TensorFlow variables that are required in constructing our model. To represent the entire dataset, we consider a data structure that holds variables such as model weights, bias, and various counters and cumulative statistics updated during training. We define a forward pass method to compute loss with these variables and model parameters, make predictions, and update the statistics of sample batches of input. After that, we define a function that is responsible for returning a set of local metrics. Each client devices send its local metrics to the central server. The server performs aggregation upon receiving the values in a federated learning evaluation process. To the end, we build a model representation to use with TensorFlow-Federated (TFF) and apply the Keras optimizer to the model. Finally, the constructed model and optimizer instances are fed to the FedAR algorithm, and

the process is initialized with federated train data to generate output metrics loss and accuracy. The overall working process is depicted in Figure **4.3**.

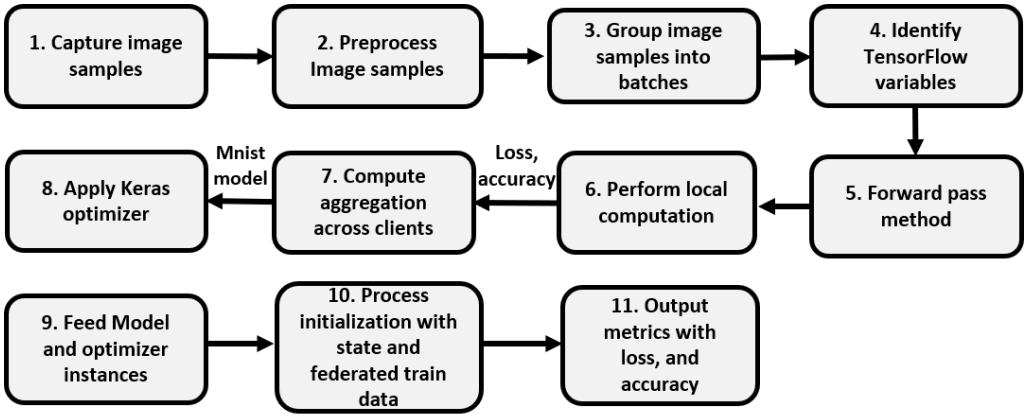


Figure 4.3: Federated Learning-based image classification approach considering TensorFlow framework.

◇ Step 6: Evaluate Local Model Quality

After client selection, each client can be trained with the FL optimization technique (e.g., FL-SGD). The task publisher disseminates the initial shared global model parameters to all selected clients. Each mobile robot trains its model using local data and uploads its model parameters to the server. To ensure the reliability of the model update, we leverage model quality evaluation by applying the FoolsGold scheme [FYB18] that identifies unreliable participants by observing their local model updates' gradient diversity and helps to avoid poisoning attacks. If a local client's model update performance is lower than a specified threshold or a client repeatedly sent similar gradient updates, then the task publisher rejects the client update and does not update the global model. With resource checking, unreliable client handle strategy, and malicious attacker detection approach, weak clients, and unreliable model updates are not considered during the learning process. The task publisher

only considers the reliable local model update and performs a federated averaging strategy [MMR+17] to generate an updated global model.

◇ Step 7: Leveraging Asynchronous FL to Accelerate Convergence

This chapter assumes that we have resource-constrained clients within the learning environment; therefore, performing synchronous federated learning can lead the server to wait for a long time to update the global model. In a synchronous FL, the server performs aggregation only after receiving a response from all the available clients. However, if a client has a straggler effect, then the client's slow response time can harm the overall model convergence. For such a scenario, synchronous FL does not guarantee convergence [BEG+19a, ITW+20]. On the other hand, in asynchronous FL, the server performs aggregation every time it receives a model from a client (See Figure **4.4**), i.e., the task publisher does not have to wait for a particular client. Hence, the straggler effect does not hamper the overall model effectiveness, and it guarantees convergence [LZZL17, XKG19]. We applied the asynchronous FL strategy in our proposed FedAR algorithm due to the uncertain response time of the heterogeneous resource clients. Each time the server updates its global model, the server sends the model back to all the available participants for the next iteration until the updated global model satisfies a convergence condition set by the task publisher.

◇ Step 8: Update Trust Models based on Performance

After successfully executing an FL task, the server assigns the trust t value to each participant based on their performance. If we update the trust value after reaching model convergence, then there remains a possibility of repeatedly selecting stragglers and unreliable clients for the training process. Rather than, we update

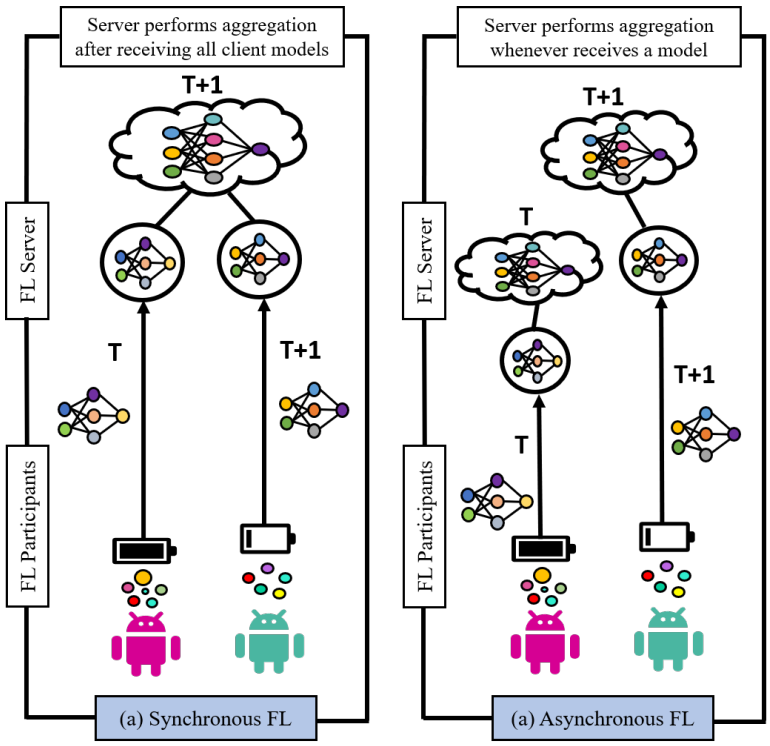


Figure 4.4: Comparing Synchronous and Asynchronous Federated Learning structures based on global model update period.

each FL participant's trust score after completing every iteration, which eventually helps to avoid straggler or inconsistent clients in the further iteration.

4.3.3 Proposed FedAR Algorithm

We presented our proposed *FedAR* scheme in Algorithm **6**. Initially, each client needs to register themselves to join a network (line **1**). We initialize each newly joined FL client with a trust value (line **2**). The server disseminates a task with a system requirement to each available client and initializes the task parameters (line **3-4**). In (line **5**), the server receives available resource information from all clients.

In each communication round of the training phase, the available resource of each client is compared with the system requirement, and eligible clients are included in

Algorithm 6: FedAR: Activity and Resource-aware Federated learning. The $\mathcal{S}$ eligible clients are indexed by $u; \mathcal{B}$ is the local mini-batch size, $\mathcal{F}$ is the client fraction, E is the number of local epochs, η is the learning rate, and t represents timeout.

Registration: Each client commits registration
Initialize Trust score to newly registered clients
Disseminate system requirements to all clients
Server executes: initialize w_0
Reveal resource availability by each interested client.
for *each round* $m = 1, 2, \ldots$ **do**
    $RA_m =$ **CheckResource** $(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$
    Sort available client based on $TrustList$ and RA, and store in $\mathcal{S}$
    $\mathcal{C} \leftarrow$ Top $\mathcal{S} \cdot \mathcal{F}$ clients
    $M_m \leftarrow$ (random set of $\mathcal{C}$ clients)
    for *each client* $u \in M_m$ *in parallel* **do**
        $w_{m+1}^u \leftarrow$ ClientUpdate (u, w_m)
        if *Model received from* u *within time* t **then**
            $w_{m+1} \leftarrow w_{m+1} + \frac{n_m}{n} w_{m+1}^u$
        UpdateTrustScore (m, u, w_m, t, γ)
ClientUpdate(k, w) : // Run on client k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $\mathcal{B}$)
for *each local epoch* i *from 1 to* E **do**
    for *batch* $b \in \mathcal{B}$ **do**
        $w \leftarrow w - \eta \nabla \ell(w; b)$
    return w to server

a list RA (line **6-7**). We sort the interested clients according to their trust score and resource availability, and store the eligible candidates into a list $\mathcal{S}$ (line **8**). We select a fraction of the eligible candidate in line **9**, and randomly select a subset of eligible clients (line **10**). For each selected client, we pass the latest global model (line **11-12**). We assume, there are M clients, each client have n_u local data, and the overall data n is partitioned among the M clients with a set of indexes $\mathcal{P}_u$ on client u, where $n_u = |\mathcal{P}_u|$. Each client splits their local data into batches, performs SGD for each of the batches, and sends back the local model parameter to the server (line **16-21**). If the server receives the client's model parameter, it immediately updates the global model by performing aggregation (line **13-14**). Finally, in line **15**, the trust score is updated based on a client's response within a threshold time of t.

4.4 Experimented Results

4.4.1 Simulation Settings

To create a resource-bounded real-time FL environment, we considered twelve distributed mobile robots that can follow a set of given instructions. We integrated variant sizes of memory, processor, and battery to each mobile robot to simulate the heterogeneous environment in terms of hardware configurations. We use a combination of a popular digit classification dataset called MNIST [oST] and our captured digit images using distributed mobile robots. We consider eight reliable and consistent robots and four unreliable robots. Among the four unreliable robots, two of them have issues regarding resource scarcity, while the other two generate low-quality models that can be considered a poisoning attack. We provided ten classes of digits for the unreliable workers and deliberately modified some of the training

samples to mislead our FL model training process. The strength of the poisoning attack depends on the modification percentage of the labels. The robot clients use a batch size of twenty and compute five local iterations to accomplish a local SGD update. We set the same transmission rates for all the existing robot clients during the local model update to attain simplicity during execution. Therefore, we maintain the same energy consumption and transmission rates for all robot clients.

4.4.2 Performance Evaluation

We evaluate an FL setting's performance by considering different chunks of data samples for the distributed mobile robots (see Table **4.2**). As we mentioned before, we consider four unreliable mobile robots with resource limitations and assign fewer image samples and classes to those clients (i.e., Robot 3, Robot 5, Robot 6, Robot 9 in Table **4.2**). We randomly apply either Softmax or Relu activation and set instructions for each robot to collect image samples from the environment. The data label and image sample number assigned to each of the robots are presented in Table **4.2**. Each image is simply a matrix of pixels where each pixel indicates color density. We flatten the 28x28 image samples into 784-element arrays. After that, we shuffle the samples and group them into batches. We shuffle our image samples to avoid the risk of creating batches that are not representative of the overall dataset. We group the images into batches to consider a mini-batch of data samples while applying local SGD. We utilize an FL framework TensorFlow 1.12.0 to evaluate the digit classification task. We apply the forward pass method to perform local computation, apply a *keras* optimizer function to obtain an optimized result, and finally, generate the model loss using the $SparseCategoricalCrossentropy$ loss function.

Table 4.2: Model architectures of FL mobile robots.

FL Client	**Emnist Labels**	**Activation Function**	**Number of Image Samples**
Robot 1	0-9	Softmax	1000
Robot 2	0-9	ReLu	1000
Robot 3	0,1,2,3	Softmax	400
Robot 4	0-9	Softmax	1000
Robot 5	4,5,6	ReLu	300
Robot 6	7,8,9	ReLu	300
Robot 7	0-9	Softmax	1000
Robot 8	0-9	ReLu	1000
Robot 9	5,6,8	Softmax	300
Robot 10	0-9	Softmax	1000
Robot 11	0-9	ReLu	1000
Robot 12	0-9	Softmax	1000

We simulated our model's behavior with the variance of batch size and local epoch of each robot client (see Figure **4.5**). In Figure **4.5**, we use E to represent the local epoch of each client and B to indicate the data samples' batch size. We can see that the accuracy of FL increases (i.e., prediction loss decreases) as the communication round goes up. We observe that when we have a batch size of 10 and a local epoch of 20, we obtain better model accuracy for our data sample. The learning period continues until we reach a target convergence.

To observe the clients' trust score update, we considered different events, e.g., successful task completion, improper model infusion, response delay, interested to be a part of a training phase, and not able to participate due to unavailable resources. In Figure **4.6**, we visualized the trust score update of three different mobile robots in different periods. Besides, we simulated the straggler effect on the FL process by considering different straggler robots that can not send back their local model update within a given time and eventually reduce the global model's overall accuracy. Figure **4.7** depicted that less number of stragglers accelerates the FL accuracy.

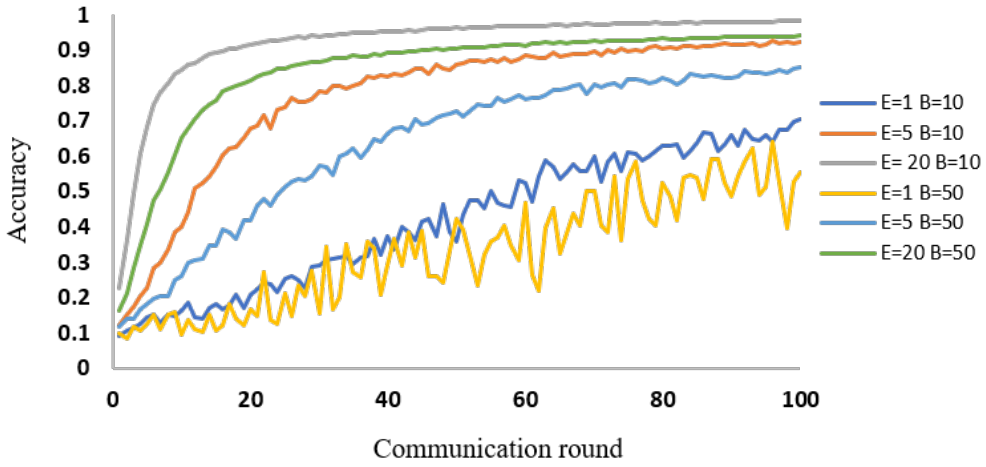


Figure 4.5: Federated Learning accuracy of mobile robots with variance in batch size and local epoch.

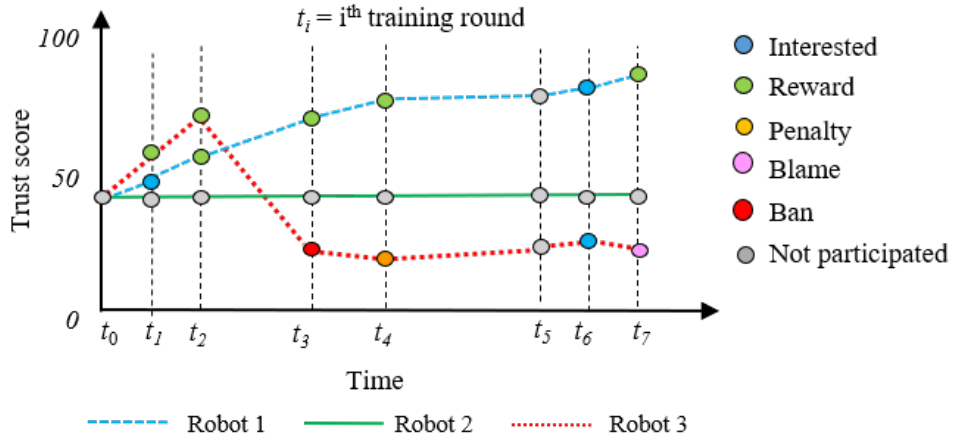


Figure 4.6: Trust score update of three mobile robots applying our proposed our activity and resource-aware Federated Learning (FedAR) algorithm.

4.5 Conclusion

This chapter proposes a trust and resource-aware FL framework to deal with the untrustworthy and resource-constrained FL environment. For our simulation settings, we consider distributed mobile robots that have reliability and resource limitation issues. Instead of assuming all the FL clients are consistent and resource-efficient,

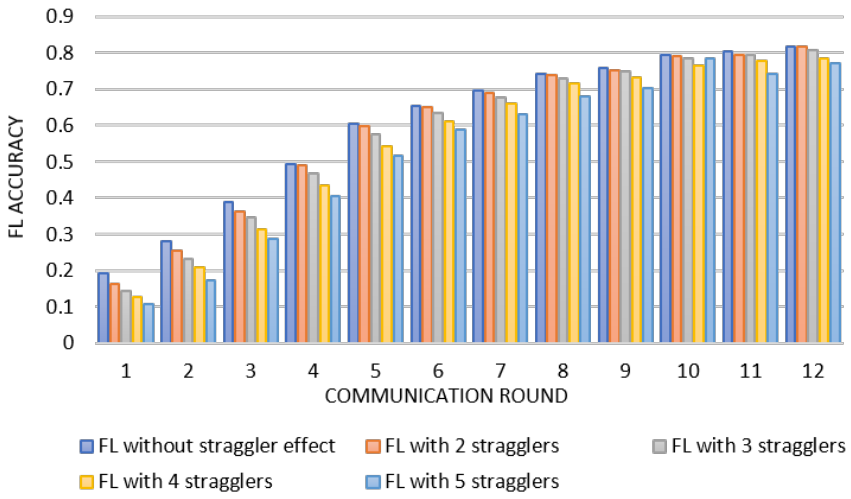


Figure 4.7: Performance of our proposed Federated Learning model in presence of straggler effect.

we consider additional steps to check each client's resources and previous training performance. We enable a feature of assigning a trust score to each robot client and avoiding the inconsistent and inadequate resources occupied by clients during the training process. To further handle the straggler effect, we selected the most proficient and reliable clients for an FL task and applied asynchronous FL to reduce the convergence time. In the next chapter, we build on FedAR to develop a tailored algorithm for resource-constrained heterogeneous IoT environments that can handle both systems and statistical heterogeneity of trust in the FL agents.

CHAPTER 5

FedPARL: CLIENT ACTIVITY AND RESOURCE-ORIENTED LIGHTWEIGHT FEDERATED LEARNING MODEL FOR RESOURCE-CONSTRAINED HETEROGENEOUS IOT ENVIRONMENT

[1] In this chapter, we presented our tri-layer Federated Learning (FL) model, which is an extended version of our FedAR model, and we additionally try to answer two things: first, How our system will respond if we have large data across the network, and second how we can tackle a situation if the majority of the available clients possess low resources and if we have no other way without selecting the resource-scarce clients for the training. So, in a nutshell, we focus on how can we carry-out lightweight model training and count every little contribution from the resource-constrained clients.

5.1 Abstract

Federated Learning (FL) is a recently invented distributed machine learning technique that allows available network clients to perform model training at the edge, rather than sharing it with a centralized server. Unlike conventional distributed machine learning approaches, the hallmark feature of FL is to allow performing local computation and model generation on the client-side, ultimately protecting sensitive information. Most of the existing FL approaches assume that each FL client has sufficient computational resources, and can accomplish a given task without facing any resource-related issues. However, if we consider FL for a heterogeneous IoT environment, a major portion of the FL clients may face low resource-availability (e.g.,

[1]This chapter is an edited version of the author's previous work published in [IA21]

lower computational power, limited bandwidth, and battery-life). Consequently, the resource-constrained FL clients may give a very slow response, or, may be unable to execute the expected number of local iterations. Further, any FL client can inject inappropriate model during a training phase that can prolong convergence time and waste resources of all the network clients. In this dissertation chapter, we propose a novel tri-layer FL scheme, ***FedPARL***, that reduces model size by performing sample-based pruning, avoids misbehaved clients by examining their trust score and allows partial amount of work by considering their resource-availability. The pruning mechanism is particularly useful while dealing with resource-constrained FL-based IoT (FL-IoT) clients. In this scenario, the lightweight training model will consume less amount of resources to accomplish a target convergence. We evaluate each interested client's resource availability before assigning a task, monitor their activities, and update their trust scores based on their previous performance. To tackle system and statistical heterogeneities, we adapt a re-parameterization and generalization of the current state-of-the-art Federated Averaging (FedAvg) algorithm. The modification of the FedAvg algorithm allows clients to perform variable or partial amounts of work considering their resource-constraints. We demonstrate that simultaneously adapting the coupling of pruning, resource and activity awareness, and re-parameterization of the FedAvg algorithm leads a to more robust convergence of FL in IoT environment.

5.2 Introduction

We first discuss the motivations for introducing the Federated Proximal, Activity and Resource-Aware Lightweight model (FedPARL) that can handle system and statistical heterogeneity of the clients and is particularly effective for a resource-

constrained FL-IoT environment. We analyze the existing works in the FL domain and mention how FedPARL can be effective in filling up the gap of prior research considering the FL-IoT setting. Further, we describe our research contribution and justify the necessity of conducting this research work.

5.2.1 Motivation

Federated Learning (FL) has come to the light because of its promising paradigm as distributed machine learning training over a network of available devices. Prior works focused on distributed optimizations and learning [SSZ14, CS12, TLR12]. However, FL has a unique way of generating a cumulative global model by learning from the client's model parameters, and it has two distinctive challenges from conventional distributed optimization: system heterogeneity and statistical heterogeneity [MMR+17, LSZ+20, YLCT19a, ZLL+18]. The details description of the FL challenges (i.e., handling heterogeneity by performing on-device training and considering low participation of network clients, tackling high communication costs) are discussed in [MMR+17, LSZ+20, ITW+20, SCST17]. The earlier invented FL algorithm, FedAvg [MMR+17] is an iterative process and optimization approach that generates a global model by learning from the client's local update. Though the FedAvg algorithm has a significant contribution in FL settings, it has missed some underlying challenges that can be observed in a heterogeneous FL-IoT setting. First, the FedAvg assumes all the available clients as uniform capabilities and randomly selects a fraction of local clients for the training phase. However, in a real-world FL setting, we may observe a marginal difference in various clients in terms of their system configurations. Second, FedAvg does not entitle to perform variable or partial amounts of work by the participated clients; rather, it simply drops

the participants that fail to perform a given task within a specified time window [BEG+19a]. Third, the performance of FedAvg diverges significantly when the client has non-identically distributed data across their devices, i.e., there remains statistical heterogeneity within the FL network [MMR+17, LSZ+18]. Forth, the FedAvg algorithm does not guarantee convergence in case most of the clients are dropped, or if the majority of the clients send back divergent model updates compared to the actual target.

In this dissertation chapter, we propose a novel FL model, referred to as FedPARL, that can be effective for resource-constrained and highly heterogeneous FL settings. Our developed FL model, FedPARL, is a tri-layer FL model that reduces the model size by applying sample-based pruning, supports the effective clients through a trust and resource checking scheme, and allows partial amounts of computational tasks by examining their resource-availabilities. We bridge the gap between systems and statistical heterogeneity by reparameterization of the FedAvg algorithm [MMR+17]. Instead of dropping the underperformed clients and naively considering the partial amounts of work from the participated clients (that may prolong convergence), we added a proximal term by considering the resource-availability of the selected clients. By checking the trust score and resource-availability of the clients, our proposed approach shows more stability than the existing FedProx framework [LSZ+18] in a highly resource-constrained FL-IoT environment.

5.2.2 Background and Related Works

The invention of new distributed optimization and learning techniques has recently been popular due to the extensive growth of data that opens the door to rethink the design of ML and data center settings [RT16, SSZ14, ZDW13, WRXM18, DGBSX12,

BPC11, A+15]. On one side, the improvements in internet availability, speed, and architecture bring more convenience for Internet-of-things (IoT) services. On the other hand, the ever-growing development of modern edge devices (e.g., smartphones, wearable devices, drones, and sensors) enables performing computation at the edge without passing local sensitive data to the server. The *Federated Learning (FL)* technique is invented after being motivated by the same theme [MMR+17]. Though FL faces many challenges in terms of systems and statistical heterogeneity, privacy, communication overhead, and massively distributed federated network [ITW+20, YLCT19a], the wide popularity of the FL approach motivates researchers to develop new optimization techniques suitable for a federated setting. Such novel federated optimization technique outperforms the conventional distributed methods, e.g., mini-batch gradient descent [DGBSX12], or Alternating Direction Method of Multipliers (ADMM) [BPC11]. The distributed optimization technique, e.g., [MMR+17, SCST17, ZLL+18, KMY+16, MSS19, SWMS19] allows for inexact local model updating that would help to balance between computation and communication in large-scale networks, and permit to active a small subset of devices at any iteration period [MMR+17, SCST17, LSZ+20, IA19]. For instance, a multi-task learning framework is proposed in [SCST17] to assist FL clients in learning separate but close models through a primal-dual optimization strategy. Although their proposed method guarantees convergence, the approach is not generalizable for the non-convex problem. For the non-convex settings, the FedAvg algorithm [MMR+17] considers averaging client local SGD update and outperforms existing models. Besides, to avoid the issues regarding active clients and statistical heterogeneity of the FedAvg algorithm, a couple of works [Sti18, HKMC19, BDKD19, WPS+20, KMR19a, MKG+20] have shown efforts to analyze FedAvg algorithm considering the non-federated setting, i.e., they assume the data to be identical and uniformly

distributed. However, in a heterogeneous setting, it is not proper to assume that each local solver can perform the same stochastic process using their local data. Further, the authors in [CYS+20] proposed a joint learning framework by considering the effect of wireless quality during model training such as packet errors, and limited bandwidth. By considering joint learning, resource factors, and client selection, they formulate objective functions of the optimization problem. Besides, the authors in [YCS+20] investigated the issues regarding effective energy utilization during model computation and transmission for FL over wireless networks. Though wireless quality and optimal energy utilization are two important factors for a resource-constrained IoT environment, these two factors are out-of-scope of this research.

One of the main challenges in federated networks is systems heterogeneity, i.e., the clients within the network may possess variant memory, processing capability, battery-life, or bandwidth. Such heterogeneity exacerbates straggler issues and degrades system performance. If the number of stragglers becomes high, then it may take a long time or even fail to reach the target convergence. One solution could be to avoid the resource-constrained clients or not select them during a training phase [BEG+19a, ITW+20, Imt20b]. However, dropping the stragglers could limit the number of active clients and it could bring bias during training or even some dropping clients may have important data with higher volume [LSZ+18]. Beyond systems heterogeneity of the FL clients, statistical heterogeneity or divergence of client model update is also a concern in federated networks. Some recent FL works [WTS+19, HM19, NSH+20, GLL20, DTN+19, CPSC20] analyze how to guarantee convergence both in theoretically and empirically for an FL setting. The major problem is that they assume all FL clients are resources capable to perform a predefined uniform number of iterations while considering all the devices to participate

in the training round. However, such assumptions are not feasible if we consider a realistic FL network [MMR+17, LSZ+20]. To handle statistical heterogeneity, some works proposed the idea of sharing either client's local data or the server's proxy data [ZLL+18, JOK+18, HY+18]. However, the assumption of passing client data to the server or disseminating proxy data to all the clients could violate the privacy [JOK+18, HY+18]. The authors in [LSZ+18] proposed a framework that can handle both systems and statistical heterogeneity. Through generalization of the FedAvg algorithm and adding a proximal term, they handle statistical diversity and allow partial amounts of work. However, they randomly select a subset of clients like the FedAvg algorithm [MMR+17], which would not be effective in an FL-IoT environment as most of the participants would be inactive or out-of-resources. In the worst case, the random selection of the participants may lead them to choose all the straggler devices that could hardly perform an iteration. Besides, in their simulation, they consider that the straggler or inactive client would take a random local iteration between 1 to E, where E is the local epoch defined by the task publisher for the overall task. In the worst case, it is possible that most of the stragglers need to perform local epochs close to the E. That means, instead of considering the resource-availability or previous history, they randomly assign a local epoch for the straggler or inactive clients. Particularly, in a real-life FL setting, such random assigning of local epoch to the stragglers would result in an ineffective model update.

In this work, inspired by FedAvg [MMR+17] and FedProx [LSZ+18], we design a tri-layer FL model, *FedPARL* that can be effective, specially in an FL-IoT settings. In the initial layer, we perform a sample-based model pruning on the server so that the server and the client can deal with smaller model size. In the second layer, we examine the resource-availability (CPU, memory, battery-life, data volume) as well as previous activities, and select the proficient and trustworthy clients

for the training phase. In the third layer, we perform a generalization of the FedAvg algorithm to allow partial works by assigning local epochs according to the client's resource-availability. Our tri-layer FL framework accelerates convergence and improves robustness in a resource-constrained FL-IoT environment.

5.2.3 Contribution

The main contributions of this dissertation chapter can be listed as follows:

- We propose a tri-layer FL scheme that helps resource-constrained FL clients consume less resources during training, avoid untrustworthy and out-of-resource clients (e.g., low battery-life) during client selection for training, and perform variable local epochs based on the client's resource availability.
- We perform model pruning to reduce the size of the client model that will be more efficient in an FL-IoT setting.
- We integrate a reward-punishment scheme to incentivize effective clients to participate in future training rounds and to punish the malicious and under-performed clients.
- We allow partial amounts of computational tasks to be performed by the participating FL clients and our proposed approach is robust even in a resource-constrained FL-IoT environment.

5.3 Federated Optimization Techniques

In this section, we highlight the widely popular FedAvg and FedProx algorithms and present the outline of our proposed FedPARL framework. In the FedAvg [MMR+17]

method, the central server initializes a global model which is updated based on the client's local model parameters. The main aim of the FedAvg algorithm is to minimize an objective function (loss) which can be expressed as follows:

$$\min_{w} F(w) := \sum_{i=1}^{N} P_i F_i(w), \tag{5.1}$$

where N is the number of devices, $P_i \geq 0$ refers to the impact of each device on overall FL model, satisfying $\sum_i P_i = 1$, and F_i denotes the objective function of local device i. Here, we assume that n_i samples are available at each device and $n = \sum_i n_i$ is the total data points, hence, $P_i = \frac{n_i}{n}$.

In the FedAvg procedure, the central server selects a fraction of clients for the training round and a local objective function is used as a replacement of the global objective function considering the device's local data. At first, the server initializes a global model that is disseminated to a fraction of local clients which are randomly selected. The clients that are selected for the training phase are called participants. After that, each client trains themself locally with E number of local epochs by applying stochastic gradient descent (SGD) using their local data as well as global model information and sending back the model information to the server. Further, the server performs aggregation based on all the received model parameters and updates the global model. The iteration process is continued until a specific iteration round or until the global model reaches convergence. Each iteration process is called a federation [JWK+19]. However, instead of enforcing all the clients to perform an exact local epoch, we can allow a flexible or inexact local objective function to solve by each client. The authors in [MMR+17] discussed that tuning up the number of local epochs plays an important role in reaching convergence. On one side, a higher number of local epochs leads to more local computation to be performed by the FL clients and reduces the communication overhead with the server resulting in faster

convergence. On the other side, if the heterogeneous FL clients possess dissimilar local objectives and perform a higher number of local epochs, then model convergence could be negatively affected which may even cause model divergence. Besides, in a heterogeneous FL-IoT environment, setting up higher local epochs may increase the possibility that the FL clients fail to perform assigned computational tasks. Further, if the FL clients perform a lower number of local epochs, it may reduce local computations, but may prolong the communication overhead and convergence time. Therefore, it is vital to set local epochs as sufficiently high while also ensuring robust convergence. As the suitable number of local epochs may change at each training round and depends on device resources, thus, determining the number of local epochs can be considered as a function of on-device data and available system resources. For tuning the local computation and client-server interaction, we adapt an inexact solution that allows flexible local epochs to be performed by each client which is stated below [LSZ+18]:

Definition 1(φ **-inexact solution**). Let us consider a function $\mathcal{G}(w; w_0) = F(w) + \frac{\beta}{2}\|w - w_0\|^2$, and $\varphi \in [0, 1]$, we can say w^* is a φ -inexact solution of $\min_\theta \mathcal{G}(w; w_0)$ if $\|\nabla\mathcal{G}(w^*; w_0)\| \leq \varphi\|\nabla\mathcal{G}(w_0; w_0)\|$, where $\nabla\mathcal{G}(w; w_0) = \nabla F(w) + \beta(w - w_0)$.

Here, a smaller φ resembles a higher accuracy. The advantage of φ-inexactness is that it measures the variable local computation to be performed by the selected local client at each training round. As we mentioned earlier, the system' of the clients leads to heterogeneous progress towards solving local problems, and therefore, it is necessary to allow a variant of φ considering clients' resource-availability and training round.

Another federated optimization technique is FedProx [LSZ+18], that tolerates partial works of the FL participants. By enabling fractional works of the clients and

considering a regularization term, they handle systems and statistical heterogeneity. However, the FedProx framework does not consider any pruning mechanism to reduce the model size that could be effective for resource-constrained FL devices and generates higher loss while most of the selected participants have very low resources, i.e., the majority of the selected devices can hardly perform local iterations (see Figure **5.1**). Few other prior works on federated optimization [KMY+16, SLS+18, RCZ+20, XKG19, LR20, PW20] tries to leverage federated optimization for heterogeneous networks, but none of these works are designed by considering all the features of our proposed *FedPARL* framework, i.e., pruning, checking model quality as well as client activity, and accepting partial works from the stragglers.

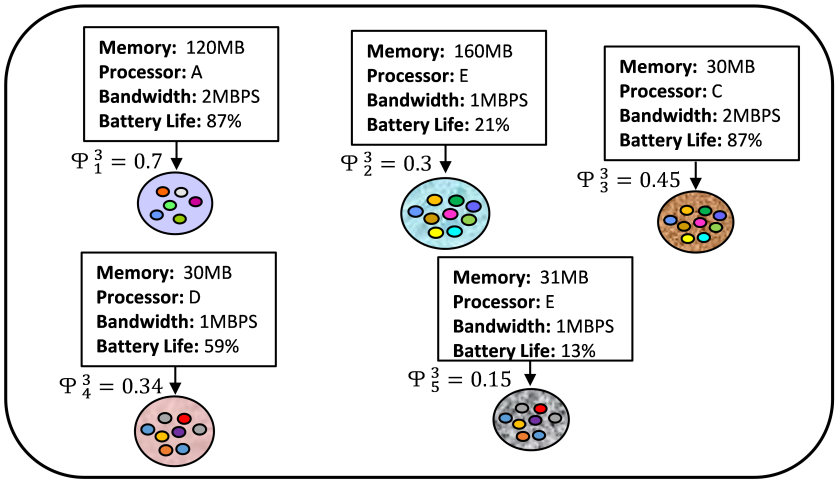


Figure 5.1: Majority of the clients performing low-partial works that leads to slower convergence.

5.3.1 Proposed Framework: FedPARL

In this segment, we discuss our FedPARL framework that consists of three layers: **1)** sample-based pruning for lightweight model training, **2)** activity and resource-aware

FL client selection strategy, and **3)** generalization of client's local objective function to perform local training epochs according to their available resources.

Sample-based Pruning

In an FL-IoT environment, as the clients may have constrained resources and limited communication bandwidth, therefore, the typical FL process may face significant challenges to perform training on a large-size model. To handle such challenges, we deploy a model pruning mechanism for reducing model size that would eventually reduce computation overhead on the client-side. The authors in [HPTD15] proposed the pruning approach for centralized ML settings, where they initially train an ML model using SGD for a particular number of iterations. After that, a model pruning is performed considering a certain level, i.e., a percentage of model weights are removed that have comparatively smallest absolute layer-wise values. The model training with the pruning process is repeated until the model reaches to the desired model size. As the training and pruning occur at the same time, we obtain a reduced model size at the end of the training process. However, the centralized pruning techniques [HPTD15, ZG17, SMBJ09, LAT18] require all the data samples for training at a central location which is not applicable for an FL process as the main theme of FL is that the clients would not share their all data samples with an external entity.

To apply the model pruning mechanism to the FL process, we aim to perform model pruning on the server with the concept of sample-based pruning and further, carry out local training on the edge clients by sharing that pruned global model. The authors in [JWK+19] discussed applying FL considering sample-based and sample-less pruning strategies. In this dissertation chapter, we apply sample-based model pruning due to its high probability of reaching convergence [JWK+19]. In sample-

based pruning, we consider a small-subset of data samples on the server that are requested from the available clients. The samples may be collected by requesting the clients to share a small portion of their available data that they wish to share, or the server can collect small samples on its own. Besides, as a device within an FL-IoT environment can act as both server and a client, therefore, that device can use its data to perform sample-based model pruning. One would expect that the quality of the pruned model would be poor compared to the existing ML-based pruned mechanism [HPTD15]. However, while applying the pruning mechanism to the FL-IoT environment, we observe that the model quality is marginally reduced with the high deduction of the model size.

After the sample-based initial pruning, further training and pruning actions can be performed on both the server and client-side. The process can be done in one or more federations. Particularly, we carry out initial pruning so that only a small size of the initial global model is shared with the FL clients and it does not consume excessive time for the edge device to perform on-device training. When pruning is performed only on the initial global model to a certain pruning level, we call it one-shot pruning. We can reduce the model size by performing repeated model pruning in every iteration of the FL process, and we call it sample-based federated pruning. The benefit of federated pruning over one-shot pruning is that it reflects the removal of insignificant parameters from the local model, i.e., it incorporates the local data impact available on the client-side. The overall pruning process is discussed below:

1. The server collects a small portion of data samples from the environment or requests the available clients within the FL network to send a small subset of data they wish to share.
2. If data is requested from the clients, then the available devices share that with the FL server.

3. For the first iteration, a global model is initialized by the server, and in case of further iteration, the global model is updated based on the feedback of the local model of the clients.
4. The server performs a sample-based model pruning until a target pruning level.
5. The pruned model is shared with the FL clients that participated in the training process. Each client updates their model utilizing their updated local data and learning from the pruned global model.
6. Each participated client can share a partial amount of work in case they have resource scarcity and the shared local model is aggregated by the server.

If we apply federated pruning, then the server again performs pruning on the updated global model by removing the parameters having small magnitudes, and the iterative process is continued until we obtain a desired pruning level. After reaching the desired pruning level, the usual FL process is executed.

Activity and Resource-Aware Model

In an FL environment, we may observe clients that have heterogeneous resources, and therefore, it is challenging to assign a task that could be performed by all the selected participants. If the majority of selected participants become stragglers, then the target convergence could never be obtained. A client can become a straggler due to being underpowered in terms of systems requirements for the assigned task completion, or, due to network connectivity. The typical FL models assume all the available clients as resource-sufficient and randomly select clients for the training phase. Besides, in an FL-IoT environment, there is a huge risk of receiving vulnerable local model updates by the clients as the IoT devices are comparatively more prone to attack [IA20]. Therefore, it is required to monitor client activities, available

resources, and their contributions to the FL process. By understanding the necessity of examining client activities and observing their resources, we integrate trust and resource-awareness into our proposed FL model. Initially, the FL server publishes a task with minimum system requirements. All the interested clients acknowledge server with sending their resource-availability information, e.g., memory, battery-life, bandwidth, and data volume. The server applies the client's information into a function and filters out the ineligible candidates. To handle inappropriate model information, we leverage a trust score mechanism to understand the records of the client's activities. In order to assign trust scores to the clients, we consider several events, e.g., infusion of the improper model, task completion or contributions towards model training, response delay, interested in joining the training phase, and unable to participate in the training round due to lack of resources.

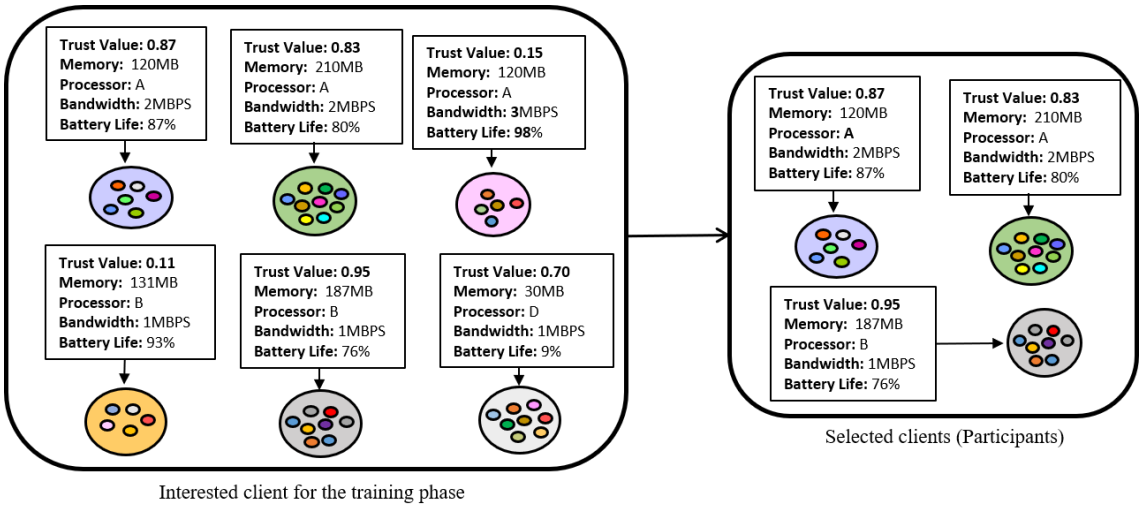


Figure 5.2: Client selection applying our novel activity and resource-aware Federated Learning algorithm.

Initially, we assign a trust score $T_{Initial} = 50$ to all the network clients. Any client who is interested to be a part of the training phase, and meets resource requirements for the model training but is not selected for the training round, we

assign a trust score $T_{Interested} = 1$ for that client. We assign this score to motivate interested and resource-proficient clients to participate in future tasks. Besides, we provide a reward score $T_{Reward} = 8$ to a client if it accomplishes the given task within a predefined time period. In case, an FL client becomes a straggler on less than 20% of its overall participation, we set a penalty to that client's trust score $T_{Penalty} = -2$. If the client becomes a straggler in equal or greater than 20% but not more than 50% of its participation, then we assign a blame score to that client's trust score, i.e., $T_{Blame} = -8$. Further, if any client becomes a straggler in equal or more than 50% of its overall participation, or, sends back an improper model, we assign a ban score ($T_{Ban} = -16$) to that client's trust score. Finally, the trust score is scaled up by dividing by 100 and stored as a trust value (with a range of 0 to 1). In Figure **5.2**, we illustrated a high-level overview of client selection process by checking resource-availability and trust score, and in Table **5.1**, we presented the chart of different factors with their associated trust score that we considered for our simulations. The trust score is assigned according to the significance of the events and we got inspired by designing such scoring of event reputation factors for our simulation from [MDB17].

Table 5.1: List of trust score corresponding to different factors for an FL-IoT environment.

Factor	**Trust score**
$T_{Initial}$	50
$T_{Interested}$	1
T_{Reward}	8
$T_{Penalty}$	-2
T_{Blame}	-8
T_{Ban}	-16

We present the details of the integration of trust and resource-awareness strategy in the FL model in Algorithm **7**. In line **1**, function receives parameters of training

round i, client id k, global model parameter w_i, maximum time t to finish task, and model diversity threshold γ. The threshold time can be set by the task publisher based on the task difficulty. We also do not fix the model diversity threshold as in the initial training round, the model diversity could be higher compared to the further training rounds. If an FL participant sends back its local model within time t, then we set the unsuccessful record of that client, U_k^i as 0 and add a reward score to that client's existing trust score (line **2-4**). On the other hand, if a client can not send back its local model within time t, we set the unsuccessful record of that client, U_k^i as 1 (line **5-6**). We examine the previous task record of that client and check whether the $U_k^i = 1$ event occurs less than 20% of that client's overall participation. If so, we add a penalty score to that client's existing trust score (line **7-8**). Particularly, for our simulation setting, we consider that each FL client which shows interest to be a part of FL training may, unfortunately, fail to accomplish a task at any time and to track their activity, we set this condition of penalty as less than 20%. Likewise, if the event $U_k^i = 1$ event occurs greater or equal to 2% but less than 50%, then we add a blame score to that client's existing trust score (line **9-10**). Finally, if the client's unsuccessful event occurs greater or equal to 50%, then we add a ban score to that client's existing trust score (line **11-12**). After assigning the trust value, the updated trust score of the client is appended into a list (line **13**). In **CheckResource** function, we take resources of the clients, i.e., bandwidth ($\mathcal{B}$), memory ($\mathcal{M}$), battery life ($\mathcal{E}$), data volume ($\mathcal{V}$) and store the resource availability status within a list, $\mathcal{R}_k$ (line **14-15**). After that, we compare the client's resource-availability with the task system requirements, and if it satisfies, then we add that client's resource availability information into another list, RA (line **16-18**). Finally, the algorithm returns the trust score and resource availability list of the clients (line **19**).

Algorithm 7: Activity and Resource Checking. Training round i^{th}, global model G^i, local model L_k^i, trust score C_k for k^{th} client, γ indicates deviation, task requirement $\mathcal{L}_{Req}$, and t represents timeout.

UpdateTrustScore (i, k, w_i, t, γ)**:**
if *k sends model to FL server within t* **then**
    set $U_k^i = 0$
    set $T_k = T_k + T_{Reward}$
else
    set $U_k^i = 1$
    if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{k}}^{\mathrm{p}} < 0.2$ **then**
        set $T_k = T_k + T_{Penalty}$
    if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{k}}^{\mathrm{p}} < 0.5$ *and* $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{k}}^{\mathrm{p}} \geq 0.2$ **then**
        set $T_k = T_k + T_{Blame}$
    else if $\frac{1}{\mathrm{i}} \sum_{\mathrm{p}=1}^{\mathrm{i}} \mathrm{U}_{\mathrm{k}}^{\mathrm{p}} \geq 0.5$ *or* G^i-$L_k^i > \gamma$ **then**
        set $T_k = T_k + T_{Ban}$
Append T_k to trustlist $\mathcal{T}$
CheckResource $(\mathcal{B}_k, \mathcal{M}_k, \mathcal{E}_k, \mathcal{V}_k)$**:**
Store $(\mathcal{B}_k, \mathcal{M}_k, \mathcal{E}_k, \mathcal{V}_k)$ into a list $\mathcal{R}_k$
Compare $\mathcal{R}_k$ with $\mathcal{L}_{Req}$
if $\mathcal{R}_k$ *satisfies* $\mathcal{L}_{Req}$ **then**
    Add $\mathcal{R}_k$ to RA list
Return $\mathcal{T}$ and RA

Allowing Partial Works from FL Participants

In this segment, we explain how the generalization of the FedAvg algorithm allows us to accept partial amounts of work from the FL participants. In FedPARL, unlike the FedAvg algorithm, we select a subset of clients (called participants) that are comparatively resource-proficient and trustworthy using the concept discussed in Section **5.3.1**. The server collects some sample data to perform sample-based model pruning to reduce the global model size. After that, the pruned global model is disseminated to all the participants. As we discussed before, the federated clients may have heterogeneous resource limitations in terms of memory, bandwidth, battery levels, processing ability, or network connectivity. It may occur that we select a proficient client that has available system configurations but somehow the device lost the network connection. Besides, it is possible that almost all the interested and available clients have limited resources, and we may have no other choices without considering those devices for the training phase. It is to be noted each device needs to use its resources to perform each local epoch. Therefore, it is not feasible to force all selected participants (i.e., IoT devices) to perform uniform local iterations. Rather than, we consider allowing partial amounts of work from the devices to tackle such challenges (see Figure **5.3**). Based on the resource-availability including the available data volume, we assign a local epoch to each participant, and perform aggregation on the server on receiving model update from any of the participants. That means, unlike FedAvg model we do not drop any stragglers, instead, we let the stragglers to compute a fewer number of iterations according to their available resources. In FedProx [LSZ+18], they consider a random network clients for the training phase and allow partial work. However, in the worst case, they may end up selecting all the devices with very low resource-availability which may lead their algorithm to perform a very low number of local epochs. Therefore, the global model

accuracy would be lower because of the deviate local model updates and it may greatly impacted when the number of local samples are few. Our resource and trust aware feature of our FedPARL framework tackles the consideration of all straggler client issue by avoiding random selection of participants and further, allows partial amounts of work of the clients. Using the idea of [LSZ+18], we can allow partial works for our federated clients that are selected through trust and resource-aware strategy and we can define the ϕ_k^c -inexactness for federated client k at training round c:

Definition 2 (φ_k^c **-inexact solution**). Let us consider a function $\mathcal{G}_k(w; w_c) = \mathcal{F}_k(w) + \frac{\beta}{2}\|w - w_c\|^2$, and $\varphi \in [0, 1]$, we call w^* is a φ_k^c -inexact solution of $\min_w \mathcal{G}_k(w; w_c)$ if $\|\nabla \mathcal{G}_k(w^*; w_c)\| \leq \varphi_k^c \|\nabla \mathcal{G}_k(w_c; w_c)\|$, where $\nabla \mathcal{G}_k(w; w_c) = \nabla \mathcal{F}_k(w) + \beta(w - w_c)$.

Here, φ_k^c determines how much local computation is needed to perform by the device k in communication round c to solve local problems. That means, φ_k^c is the representation of the variable local iterations of the clients. The systems heterogeneity can be handled by relaxing the φ_k^c -inexactness. In Figure 5.3, we present a conceptual visualization of allowing partial amounts of work to be performed by 10 heterogeneous clients in their third training round. From the figure, we can see that, client 1 and 4 are performing 70% and 30% of the overall tasks due to resource-limitations while the second client is performing the whole task because of its available resources. If we explain it in more simplified way, then let consider that the task publisher expects 200 local epochs to be performed by all the selected FL clients. However, due to resource-constraint issues, some of the clients may not be able to perform 200 local epoch for generating their local models. For such a case, considering the resource status, the weak clients are allowed to perform lower number of local epochs, e.g., the first and forth clients need to perform only 140

and 60 local epoch if the overall computational task is 200 local epochs for the third training round. For the convenience of our simulations, we assign an approximate number of local epochs to different clients considering their heterogeneous resources.

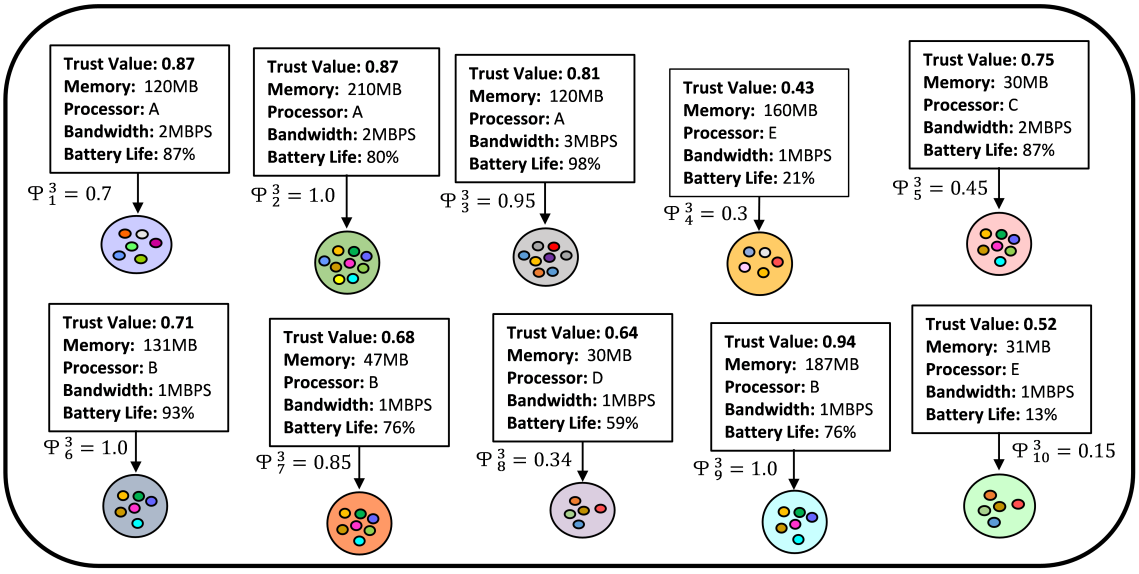


Figure 5.3: Partial amounts of work to be performed by the selected clients.

From the above discussion, we understand that variant local works can help us to deal with systems heterogeneity, however, too many local epochs, or, local update through false model injection could generate a diverge local model. The divergence local model update could be handled by adding a proximal term. In FedAvg, each device solves their corresponding local function, while the authors in [LSZ+18] considers an extra proximal term for each participant while solving local problem which is given below:

$$\min_{w} \mathcal{G}_k\left(w; w^c\right) = F_k(w) + \frac{\beta}{2}\left\|w - w^c\right\|^2 \tag{5.2}$$

The modified local function helps to restrict the local update closer to the global model which is particularly beneficial while dealing with statistical heterogeneity and

also allows partial amounts of work to be performed by heterogeneous clients. The overall process of sample-based pruning mechanism and performing partial amounts of work by the FL participants is presented in Figure **5.4**.

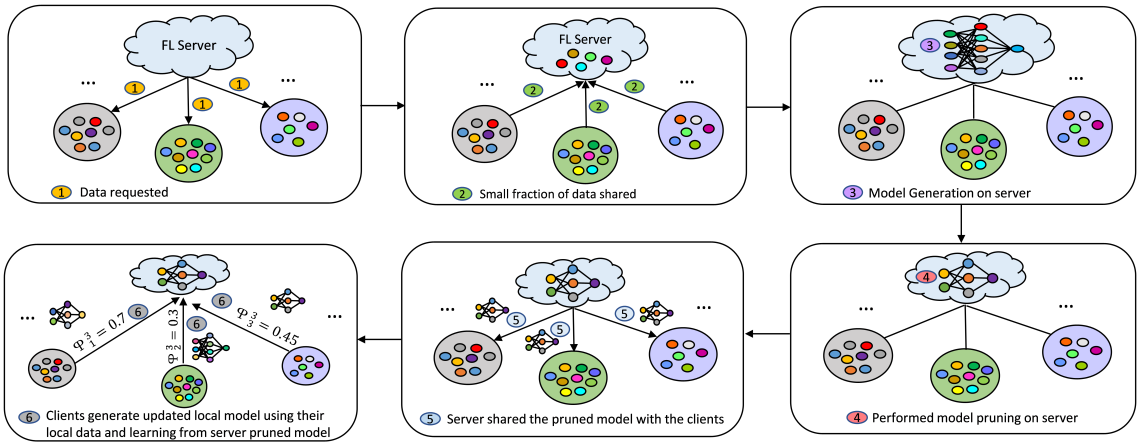


Figure 5.4: Performing sample-based model pruning on the server and allowing partial amounts of work considering client resource-availability and previous activities.

5.3.2 Proposed FedPARL Framework

We presented our proposed FedPARL in Algorithm **8**. Initially, FL server collects a small samples by sensing environment, or request fractions of samples from the available clients and performs model pruning (line **1**). After performing the model pruning, a compressed size of model w_0 is obtained, which is disseminated to all the available clients along with the task requirements (line **2-3**). Each client that are interested to perform the task shares their available resource information with the FL server (line **4**). For each training round, the FL server check available resources of each client by calling **CheckResource()** function of Algorithm **7**, and extract the trust score and available resource information of each interested clients (line **5-6**). The interested clients are sorted based on their trust $\mathcal{T}$ and available resources

$\mathcal{R}$, which are stored within a list r (line **7**). A fraction of clients from the eligible candidates are chosen and further, only a few of them are randomly selected for the training phase (line **8-9**). The FL server calls each chosen client to perform local training through **ClientLocalUpdate()** function and passes the latest global model (line **10-11**). We assume the cumulative number of data samples within the FL network is n, which are partitioned among the available clients having a set of indexes $\mathcal{P}_k$ on client k, where $n_k = |\mathcal{P}_k|$. Besides, each client's available local data during a communication round c is indicated by n_c. During training, each chosen client utilizes its local solver to figure out inexact minimizer φ_k^c to solve its local objective function (line **16-17**). After that, each client splits their data into batches, obtains optimal local solution by performing SGD, and sends back model parameters to the FL server (line **18-22**). The FL server performs aggregation upon receiving models from the chosen clients and updates their trust score based on their performance (line **12-15**).

5.4 Convergence Analysis

For the convergence analysis of our FedPARL framework, we first discussed a measure of dissimilarity called $\mathcal{B}$-local dissimilarity that can lead us to prove convergence of our proposed framework. According to [LSZ+18], the local functions of FL clients are B-locally dissimilar at w if $\mathbb{E}_k\left[\|\nabla F_k(w)\|^2\right] \leq \|\nabla f(w)\|^2\mathcal{B}^2$. From here, we can define the value of $\mathcal{B}(w)$, i.e., $\mathcal{B}(w) = 1$ when $\mathbb{E}_k\left[\|\nabla F_k(w)\|^2\right] = \|\nabla f(w)\|^2$, ($w$ is fixed solution that all the local functions of the clients agree on or all the clients holds the same local functions), and $\mathcal{B}(w) = \sqrt{\frac{\mathbb{E}_k\left[\|\nabla F_k(w)\|^2\right]}{\|\nabla f(w)\|^2}}$ when $\|\nabla f(w)\| \neq 0$. The $\mathbb{E}_k[\cdot]$ expresses the expectation over FL clients with masses $p_k = n_k/n$ and $\sum_{k=1}^{N} p_k = 1$, where n_k indicates the number of local data samples on each client

Algorithm 8: FedPARL Framework. The $\mathcal{S}$ eligible clients are indexed by k; $\mathcal{B}$ = local minibatch size, $\mathcal{F}$ = client fraction, E = number of local epochs, η = learning rate, and t = timeout.

Model pruning: FL server collects a fraction of samples and applies sample-based model pruning
Server executes: initialize pruned global model w_0
Disseminate task requirements to all clients
Collect resource information of interested clients
for *each round* $c = 1, 2, \ldots$ **do**
    $\mathcal{T}_c$, $\mathcal{R}_c$ = **CheckResource** $(\mathcal{B}_c, \mathcal{M}_c, \mathcal{E}_c, \mathcal{V}_c)$ for all interested clients
    Sort available clients based on $\mathcal{T}$ and $\mathcal{R}$, and store in a list r
    $\mathcal{S} \leftarrow$ Top $r \cdot \mathcal{F}$ clients
    $P_c \leftarrow$ (random set of $\mathcal{S}$ clients)
    for *each client* $k \in P_c$ *in parallel* **do**
        $w_{c+1}^k \leftarrow$ **ClientLocalUpdate** (k, w_c)
    for *each client* $k \in P_c$ **do**
        if *model is received from client* k *within time* t **then**
            $w_{c+1} \leftarrow w_{c+1} + \frac{n_c}{n} w_{c+1}^k$
        UpdateTrustScore (c, k, w_c, t, φ)
ClientLocalUpdate(k, w) : // Run on client k
    Each client k finds a w_k^{c+1} which is a φ_k^c -inexact minimizer of: $w_k^{c+1} = F_k(w) + \frac{\beta}{2} \|w - w^c\|^2$ and determines maximum feasible number of local epochs E
    $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
    for *each local epoch* e *from 1 to* E **do**
        for *batch* $b \in \mathcal{B}$ **do**
            $w \leftarrow w - \eta \nabla \ell(w; b)$
    return w to server

k and n denotes the total number of data samples over the whole network. In particular, B-dissimilarity definition represents a bounded dissimilarity with a privilege of allowing statistical heterogeneity in an IID scenario. According to bounded dissimilarity, if we have $\epsilon > 0$, there exists a $\mathcal{B}_\epsilon$, i.e., for all the data points $w \in \mathcal{S}_\epsilon^c = \{w \mid \|\nabla f(w)\|^2 > \epsilon\}, \mathcal{B}(w) \leq \mathcal{B}_\epsilon$. However, in FL setting, there is a high chance of observing $\mathcal{B}(w) > 1$ due to the heterogeneous data distributions within the network and larger value of $\mathcal{B}(w)$ indicates larger dissimilarity among the client's local functions.

5.4.1 Convergence Analysis: Non-convex Case and Variable ϕ's

Let us assume that the local functions F_k are non-convex, and $\mathcal{L}$ -Lipschitz smooth. Besides, we assume that there exists $\mathcal{L}_- > 0$, i.e., $\nabla^2 F_k \succeq -\mathcal{L}_- \mathbf{I}$, $\bar{\beta} := \beta - \mathcal{L}_- > 0$ and $B(w^c) \leq B$. Now, in Algorithm **8**, if we choose β, K, and ϕ following the analysis from [LSZ+18], then we obtain

$$\lambda = ((\frac{1}{\beta} - \frac{\phi\mathcal{B}}{\beta} - \frac{\mathcal{B}(1+\phi)\sqrt{2}}{\bar{\beta}\sqrt{K}} - \frac{\mathcal{L}\mathcal{B}(1+\phi)}{\bar{\beta}\beta} - \frac{\mathcal{L}(1+\phi)^2\mathcal{B}^2}{2\bar{\beta}^2} - \frac{\mathcal{L}B^2(1+\phi)^2}{\bar{\beta}^2 K}(2\sqrt{2K}+2)) > 0 \quad (5.3)$$

For iteration c of our algorithm, we can observe an expected decrease of the global objective function that can be expressed as: $\mathbb{E}_{S_c}[f(w^{c+1})] \leq f(w^c) - \lambda \|\nabla f(w^c)\|^2$, where S_c is the set of K devices chosen at iteration c.

In a similar fashion, for variable ϕ's, we have exact similar assumptions, and if we choose β, K, and ϕ of Algorithm **8** following the analysis from [LSZ+18], then we obtain

$$\lambda^c = (\frac{1}{\beta} - \frac{\phi^c \mathcal{B}}{\beta} - \frac{\mathcal{B}\left(1+\phi^c\right)\sqrt{2}}{\bar{\beta}\sqrt{K}} - \frac{\mathcal{L}\mathcal{B}\left(1+\phi^c\right)}{\bar{\beta}\beta} - \frac{\mathcal{L}\left(1+\phi^c\right)^2\mathcal{B}^2}{2\bar{\beta}^2} - \frac{\mathcal{L}\mathcal{B}^2\left(1+\phi^c\right)^2}{\bar{\beta}^2 K}(2\sqrt{2K}+2)) > 0 \quad (5.4)$$

Here, also for iteration c, we can observe an expected decrease of the global objective function that can be expressed as: $\mathbb{E}_{S_c}\left[f\left(w^{c+1}\right)\right] \leq f\left(w^c\right) - \lambda^c \left\|\nabla f\left(w^c\right)\right\|^2$, where S_c represents the set of K clients chosen at iteration c and $\phi_c = \max_{k \in S_c} \phi_k^c$

5.4.2 Convergence Analysis: Convex Case

Let us assume that $F_k(\cdot)$'s be convex and $\phi_k^c = 0$ for any k, c, such that all the clients solve the local problems exactly. Besides, we consider that the assertions that we mentioned in Non-convex case (see Section **3.1**) are satisfied. If $1 \ll \mathcal{B} \leq 0.5\sqrt{K}$, then it is viable to choose $\beta \approx 6\mathcal{L}\mathcal{B}^2$ which derives that $\lambda \approx \frac{1}{24\mathcal{L}\mathcal{B}^2}$.

5.5 Experiments

In this section, we present empirical results of our proposed FedPARL framework. We provided the experimental details, i.e., our simulation settings with detail description about the dataset we considered, demonstrated the outperformance of our proposed FedPARL framework compare to the conventional FedAvg [MMR+17] and FedProx [LSZ+18] algorithm considering systems heterogeneity, and presented the effectiveness of our FedPARL framework in presence of statistical heterogeneity within an FL-IoT setting.

5.5.1 Experimental Details

We perform experimental simulation of our proposed FedPARL framework on different datasets, tasks, and models. We implement sample-based model pruning to generate a lightweight FL model that would be effective for FL-IoT settings. To create an FL-IoT setting, we consider twelve distributed mobile robots that are capable to follow a given set of instructions. Each robot is integrated with variant sizes of memory, battery-life, and processor that brings systems heterogeneity. We carry out similar transmission rates to all considered robots for maintaining the simplicity of the FL training process. Among the twelve robots, we assume that eights are unreliable, and two of them have resource-shortage issues, and rest of the two generates low-quality models that can be regarded as poisoning attack. We consider the weak clients having low amounts of resources to simulate systems heterogeneity. Besides, we deliberately changed some of the robot client on-device samples to mislead the FL process. The strength of the poisoning attack is dependent on the degree of sample modification. For a better understanding of the effects of statistical heterogeneity, we also evaluate our FedPARL framework by considering synthetic and federated datasets.

To simulate FedPARL framework on synthetic dataset, we follow the synthetic data generation process provided in [SSZ14, LSZ$^+$18]. As discussed in [LSZ$^+$18], we generated samples $(\mathcal{X}_k, \mathcal{Y}_k)$ for each device k, considering model $y = \text{argmax}(\text{softmax}(\mathcal{W}x + b))$, where, model weights $\mathcal{W} \in \mathbb{R}^{10\times 60}$, samples $x \in \mathbb{R}^{60}$, and bias $b \in \mathbb{R}^{10}$. At first, we generate an IID dataset by keeping the similar value of $\mathcal{W}$ and b on all the available devices and set $\mathcal{X}_k$ to ensure the same distribution. After that, we define $(\alpha, \beta) = (0, 0), (0.5, 0.5)$ and $(1, 1)$ to prepare three non-IID datasets. Particularly, α helps us to control variance among the local models while β controls bringing variation among the local data located at a

device that differs from the other available devices. By controlling α and β, we prepare three other heterogeneous and distributed synthetic datasets. For all the four synthetic datasets, we considered 30 different devices in order to generate a global model with optimized model weights $\mathcal{W}$ and bias b. For the simulation of FedPARL on federated dataset, we consider two different datasets MNIST [LeC98] and Sent140 [GBH09]. For MNIST which is a popular dataset of handwriting digits, we split the overall MNIST dataset among 1000 clients such that each client has only sample of two digits. We consider Sent140 dataset for non-convex setting which is basically a sentiment analysis of tweets, and we consider 772 clients to distribute the overall datasets.

To better understand the performance and simulate the comparison of FedPARL with existing similar approaches (i.e., FedAvg and FedProx), we implement all the three approaches with same simulation settings. As FedAvg and FedProx algorithm use SGD as a local solver, hence, to bring fairness, we also apply SGD as a local solver of FedPARL. We maintain the same hyperparameters for all the experiments of a particular dataset that is obtained after proper tuning (e.g., learning rate). For each training phase, we select 10 clients as participants and we define the number of stragglers, batch size, number of iteration rounds, and learning rate. Our proposed framework is applicable to any sort of heterogeneous FL-IoT environment and the convergence time of the model training depends on the available FL client's local data and resources.

5.5.2 Simulation of Trust Score Update

As we discussed in Section **5.3.1**, the trust score is updated based on the client activities. We consider various events, i.e., interested to perform a task, successful

task completion, response delay in sending model, incorrect model infusion, or unable to accomplish a task. In Figure **5.5**, we present the trust score update of four distributed mobile robots with respect to time in various training rounds.

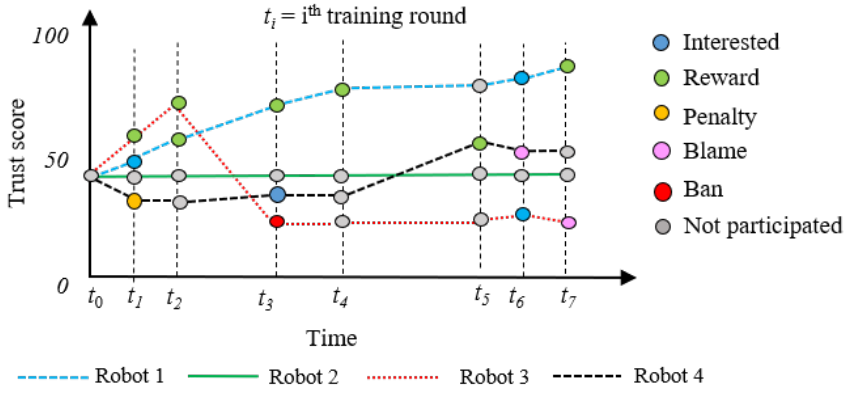


Figure 5.5: Activity dependent trust score update of four distributed mobile robots.

However, if we marginally change the threshold values of agents' participation, then we found that it negatively affects the overall model accuracy. For instance, if set the T_{Penalty} within a threshold value of 70%, then a client would repeatedly give a slow response and would still be selected for the training process. From Figure **5.6**, we can see that though Robot 3 repeatedly gives slow response multiple times, it is still selected for the training process. If we have a highly malicious FL environment, then changing the threshold values significantly could open up the door of selecting vulnerable clients for the training process.

5.5.3 Simulation of Sample-based Model Pruning

In this dissertation chapter, we particularly interested to apply our proposed framework on a resource-constraint FL-IoT environment, therefore, producing a lightweight FL model by eliminating the less important features could

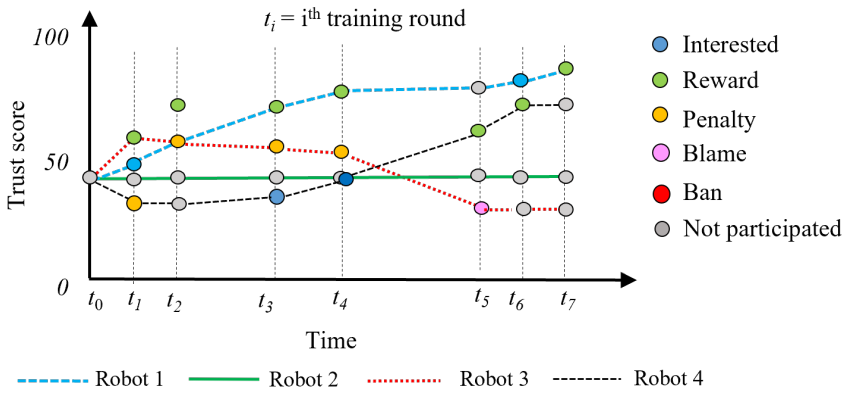


Figure 5.6: Activity dependent trust score update of four distributed mobile robots.

be effective in accelerating the training period. To perform pruning, we import *tensor_model_optimization* class from TensorFlow and extract *prune_low_magnitude* class from Keras. We define a model for pruning by setting up the epoch, batch size,and validation split. We also set up initial and final sparsity, with begin and end step for pruning. After that, we feed the model in *PolynomialDecay* class, store the parameters, and feed into the *prune_low_magnitude* class. Finally, we compile, fit and evaluate the pruned model. In Figure **5.7**, we show the steps of our applied pruning process and in Figure **5.8**, we present the effect of performing model pruning on different sizes of samples considering MNIST [LeC98] dataset, where each image has a size of 28*28 =784 pixels. After leveraging a sample-based pruning mechanism with a sparsity of 80%, the number of features is reduced to 160. Using the sample-based pruning mechanism, we found that the number of trainable parameters is 20,410 out of 40, 805 parameters while considering a Convolutional Neural Network model architecture. We can observe hat, we obtain a significantly lower pruned model size compared to the unpruned model (see **5.8**). Besides, in Figure **5.9**, we show how the accuracy varies while performing initial sample-based pruning on different

sample sizes. We can see from the Figure **5.9**, on some cases, e.g., sample 25 has less accuracy than sample 10. It is because as we randomly select small samples for training, therefore, there is a possibility of choosing the same type of sample class while missing other ones. When we consider comparatively large samples for training, we do not observe any such cases because there is a high probability of holding all the available classes. Further, we compare our pruned model accuracy with baseline model accuracy for different sample sizes that demonstrating that we loss very small accuracy while performing the model pruning (see Figure **5.10**).

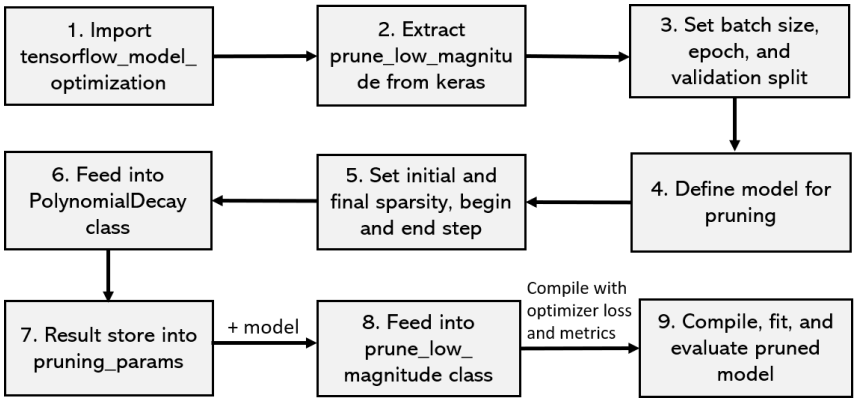


Figure 5.7: Steps of performing model pruning.

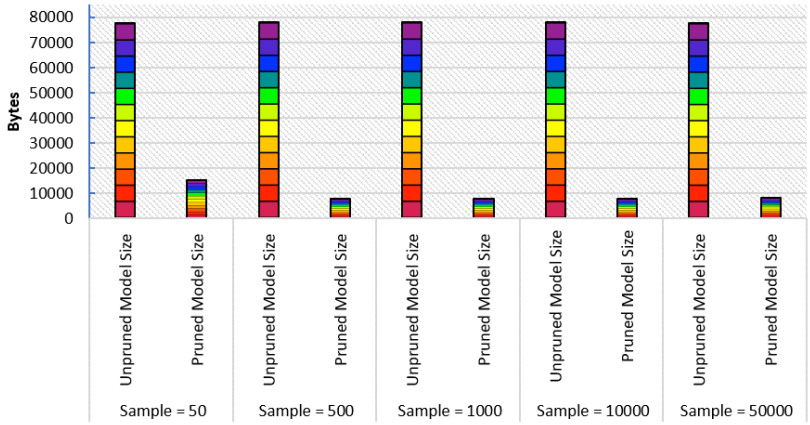


Figure 5.8: Performing model pruning and achieving compressed model sizes.

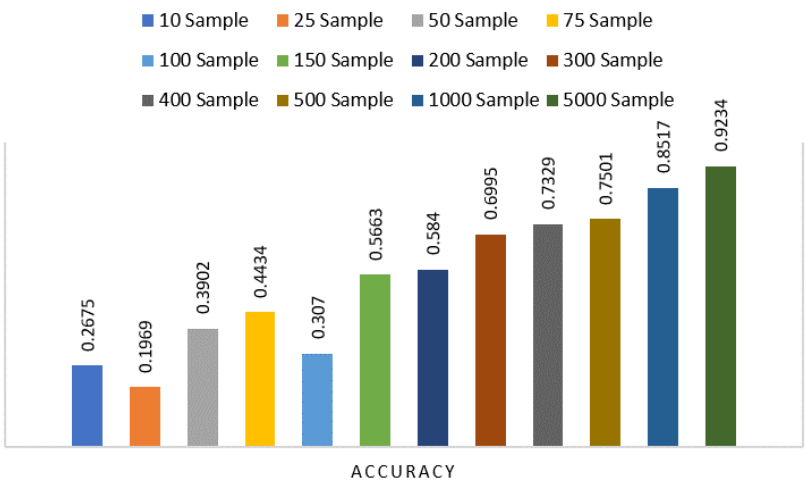


Figure 5.9: Accuracy after performing initial sample-based pruning for different sample sizes.

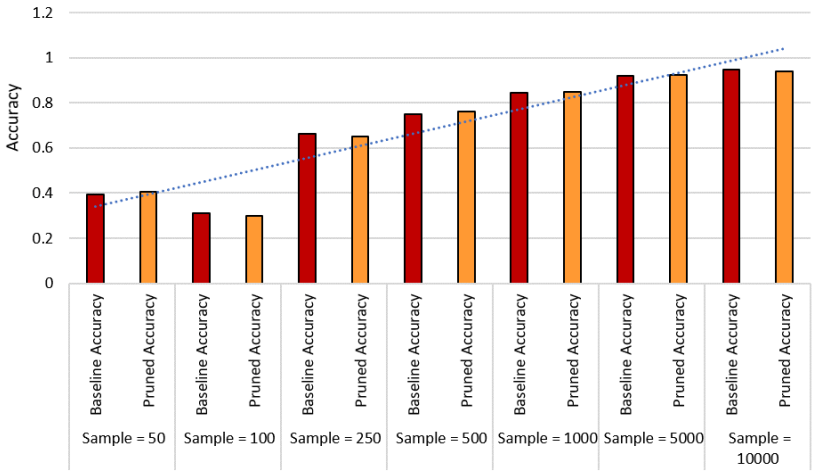


Figure 5.10: Comparison between baseline accuracy and pruned model accuracy considering different sample sizes.

5.5.4 Simulation of Handling Systems Heterogeneity

To measure the impact of allowing partial works from the clients, we simulate our federated settings by considering systems heterogeneity. We assume that, for each task, there is a global clock cycle and each participated client k measures

the amounts of work it needs to perform on iteration c (φ_k^c) as a function of its available resource constraints and clock cycle. We define a global epoch E to be performed by all the clients and if any client has resource limitations issues to perform E epochs, then that client performs fewer updates considering their resource constraints. For each task, we set the number of clients that could be stragglers, e.g., $0\%, 10\%, 25\%, 50\%, 95\%$, where 0% straggler means all the participated clients can perform the defined number of global epoch E (i.e., there is no systems heterogeneity), and 95% straggler means only 5% of the clients can perform the defined number of global epoch E. The conventional FedAvg algorithm simply drops the clients that could not perform the local epoch E, i.e., does not allow any partial solutions. In Figure **5.11**, we simulate the training loss by testing with various numbers of stragglers $(0\%, 50\%, 90\%)$ and can see that the FedPARL achieves higher training loss compared to the FedAvg and FedProx approaches. We also present the testing accuracy of our proposed FedPARL framework after accepting partial works from the stragglers and can observe that FedPARL outperforms the FedAvg and FedProx model, particularly when the majority of the clients are stragglers (see Figure **5.12**). From Figure **5.11** and **5.12**, it is evident that systems heterogeneity has negative effect on the convergence of all the datasets and higher heterogeneity leads to worse convergence. It is also clear that simply dropping the stragglers from the training rounds degrades the overall performance, and allowing partial solutions helps to ensure robustness and improves convergence. We also see that while $\beta > 0$, we achieve faster convergence, and also, in that case, FedPARL obtains higher accuracy and lower loss than FedProx. We also investigate two other FL settings with less system heterogeneity. In our first investigation, we limit the local epoch of each device to be exactly 1, i.e., each client can perform only a single local epoch. In such a case, still FedPARL performs better than FedAvg model by loosing higher training

loss (see Figure **5.13**) and by attaining higher testing accuracy (see Figure **5.14**). In our second investigation, we consider a synthetic IID dataset that does not have any statistical heterogeneity, and for such a setting, FedAvg is more robust than our proposed FedPARL framework. That means, allowing partial works from the clients does not have much effect on the overall performance while considering a synthetic dataset. The simulation results show that though we lose some accuracy while performing pruning, we can still achieve faster convergence with higher accuracy and lower loss if we select FL client effectively.

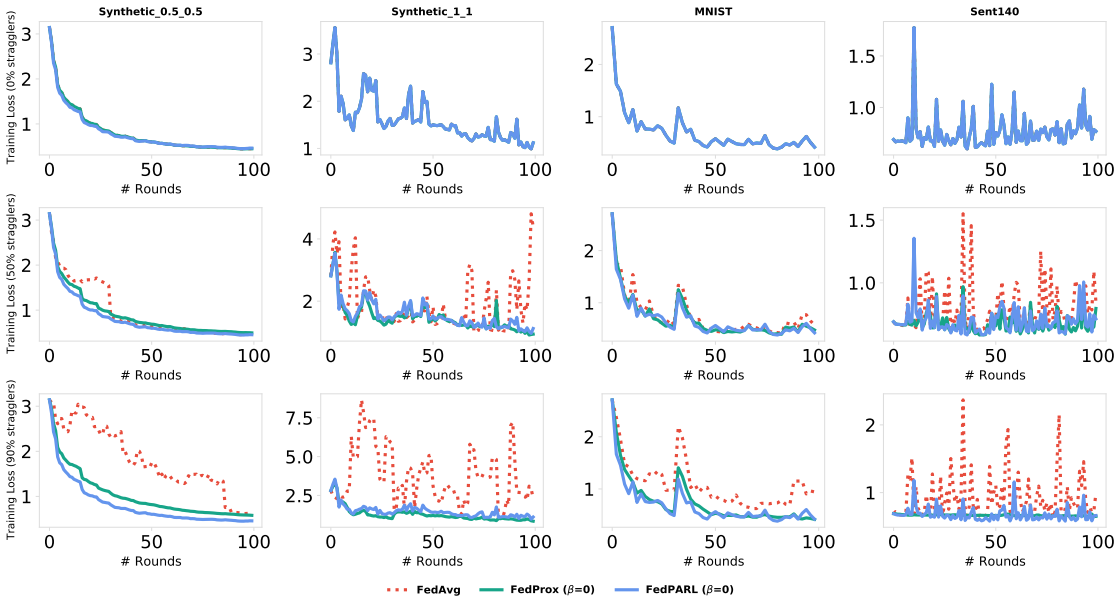


Figure 5.11: Comparison of training loss of our proposed FedPARL framework (considering no proximal term, i.e., $\beta = 0$) with FedAvg and FedProx in presence of different percentage of stragglers (i.e., $0\%, 50\%$, and 90%).

5.5.5 Simulation of Controlling Statistical Heterogeneity

To understand how our proposed FedPARL framework can handle statistical heterogeneity, we simulate the convergence behavior by eliminating the proximal term from

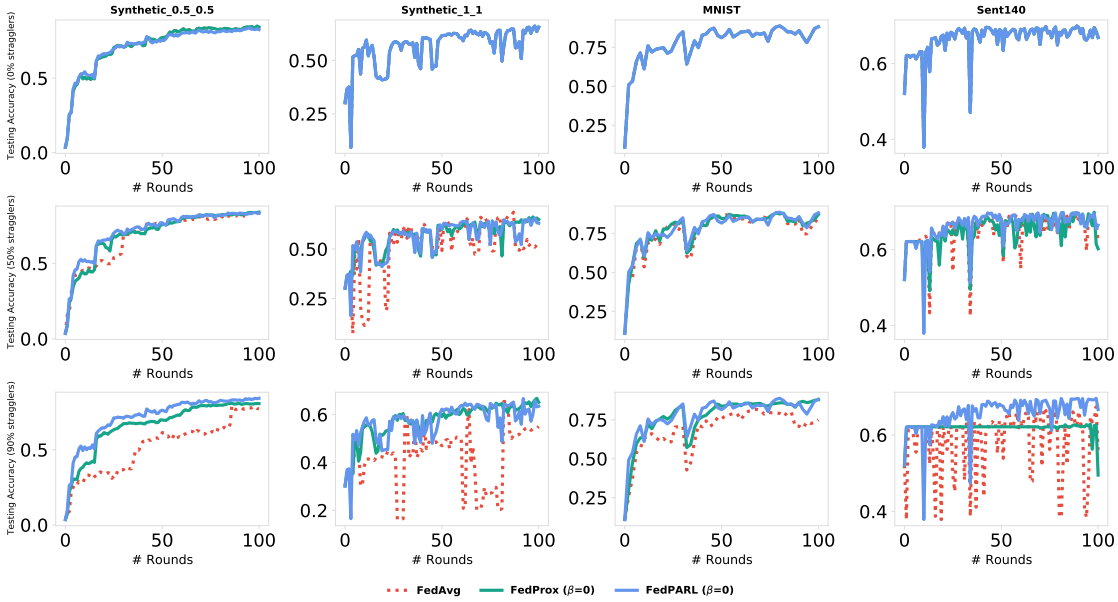


Figure 5.12: Comparison of testing accuracy of our proposed FedPARL framework (considering no proximal term, i.e., $\beta = 0$) with FedAvg and FedProx in presence of different percentage of stragglers (i.e., $0\%, 50\%$, and 90%).

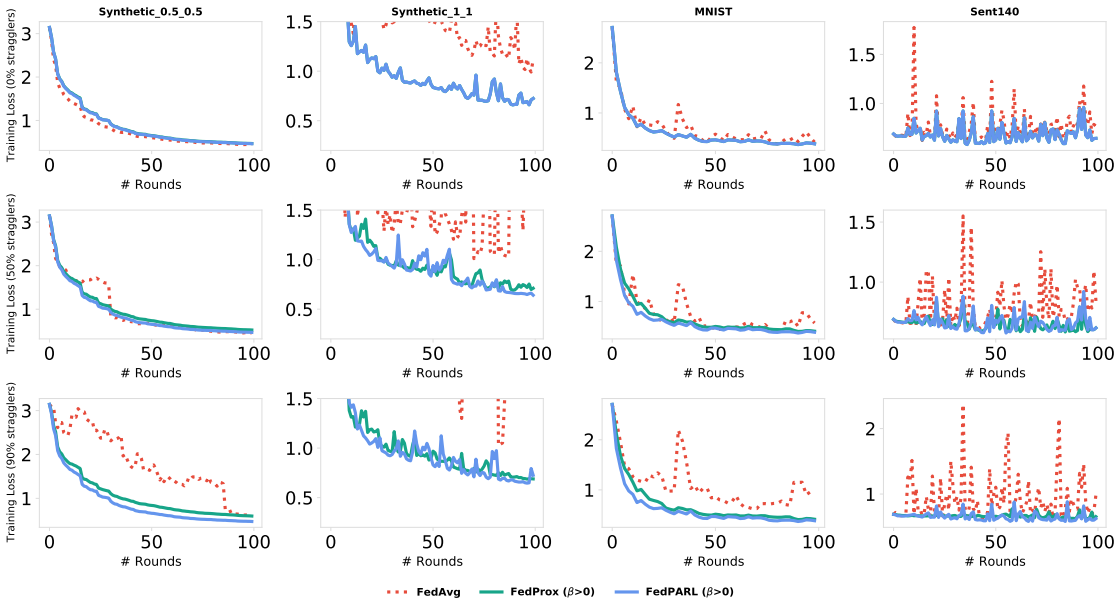


Figure 5.13: Comparison of training loss of our proposed FedPARL framework (considering $\beta > 1$) with FedAvg and FedProx in presence of stragglers.

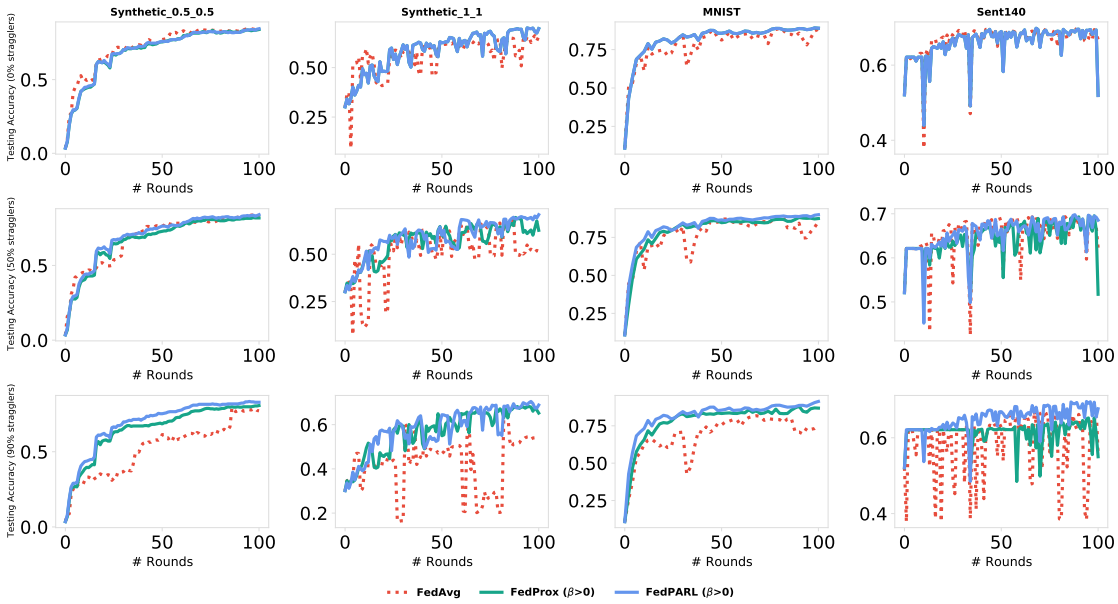


Figure 5.14: Comparison of testing accuracy of our proposed FedPARL framework (considering $\beta > 1$) with FedAvg and FedProx in presence of stragglers.

the client's local objective function. We observe that, when we bring heterogeneity within the dataset, the training performance of the clients starts to degrade. In Figure **5.15** and **5.16**, we show how statistical heterogeneity affects the convergence behavior of four different datasets. For this simulation, we do not consider any systems heterogeneity, i.e., we assume each client is resource-proficient and can perform E local epochs. The authors in [MMR+17] discussed that tuning up the number of local epochs plays an important role in reaching convergence. On one side, a higher number of local epochs leads to more local computation to be performed by the FL clients and reduces the communication overhead with the server which results in faster convergence. On the other side, if the heterogeneous FL clients possess dissimilar local objectives and perform a higher number of local epochs, then model convergence could be negatively affected which may even cause model divergence. Besides, in a heterogeneous FL-IoT environment, setting up higher local epochs may

increase the possibility that the FL clients fail to perform assigned computational tasks. Further, if the FL clients perform a lower number of local epochs, it may reduce local computations, but may prolong the communication overhead and convergence time. Therefore, it is vital to set local epochs as sufficiently high while also ensuring robust convergence. As the suitable number of local epochs may change at each training round and depends on device resources, thus, the 'best' number of local epochs can be considered as a function of on-device data and available system resources.

We also demonstrate how statistical heterogeneity degrades the performance of FedAvg ($\beta = 0$) and how the proximal term ($\beta > 0$) helps to improve convergence. In a synthetic dataset, where statistical heterogeneity does not have any influence, we can see that FedAvg performs better than FedPARL in terms of training loss (see Figure **5.15**) and testing accuracy (see Figure **5.16**). As statistical heterogeneity increases, we can see that the training loss of FedAvg decreases and testing accuracy becomes inconsistent or unstable. On the other hand, FedPARL handles the situations effectively and obtains higher training loss with a consistent as well as higher training accuracy compare to FedAvg and FedProx (see Figure **5.15** and **5.16**). We also simulate the variance of local gradients of FedPARL, FedAvg, and FedProx framework (where the lower variance of local gradients indicates better convergence), and FedPARL performs better than FedAvg and FedProx (see Figure **5.17**). We also test our system with a federated dataset and obtains similar results.

In a heterogeneity FL setting, the activities of the local clients (i.e., amounts of local work) and their model quality directly influence the overall model convergence. Defining a suitable number of local epochs for the clients is essential to utilize the client's resources effectively. Besides, a higher number of local epochs can also cause model overfitting issues. To solve the issue, we perform fine-tuning for local epoch

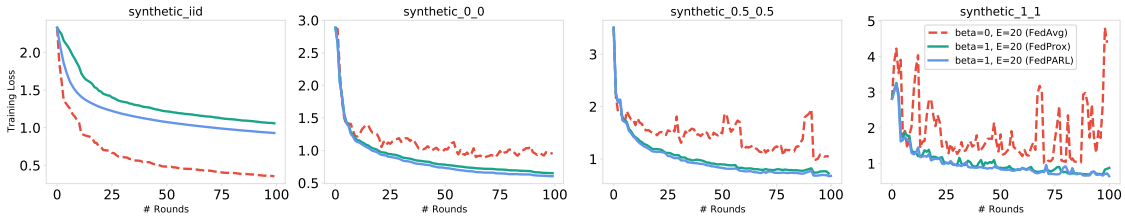


Figure 5.15: Simulation of data heterogeneity effects on training loss by considering four synthetic datasets. From left to right, the statistical heterogeneity increases.

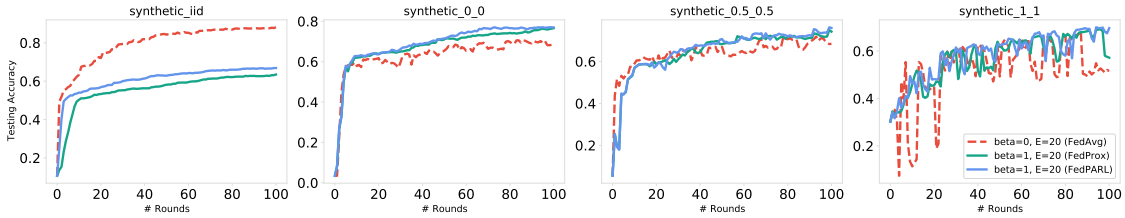


Figure 5.16: Simulation of data heterogeneity effects on testing accuracy by considering four synthetic datasets. From left to right, the statistical heterogeneity increases.

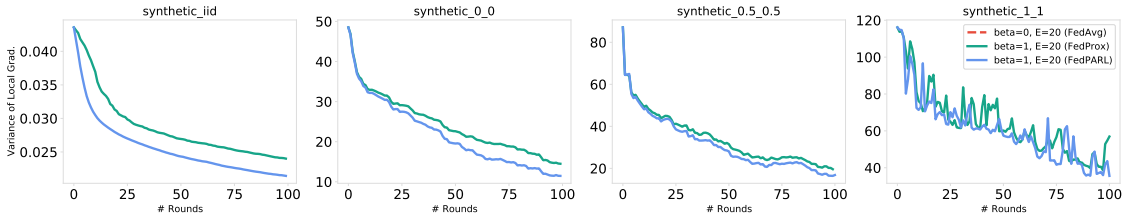


Figure 5.17: Simulation of data heterogeneity effects on variance of local gradients by considering four synthetic datasets. From left to right, the statistical heterogeneity increases.

E and after that, allow each resource-constraint device to find out the appropriate number of local epochs to perform locally. Further, if any client still sends back a diverge model update to the server, the overall model quality may degrade. To prevent that, applying a proximal term β helps to limit the local model update [LSZ+18]. Therefore, we allow the clients to perform their device-specific local epochs φ and handle the divergence of model update (if any) by adding a proximal

term β. In this way, we handle the issues of overfitting and divergence of the model update.

One of the challenges of evaluating the best model performance is to properly choose the value of proximal term β. While a large β can slow down the overall convergence time, a small value of β may not have any impact on the overall performance. The authors in [LSZ+18] figure out the best value of proximal term β for the considered different datasets. For the Synthetic_0_0, Synthetic_1_1, MNIST, Sent140, the best value of β are $1, 1, 1$, and 0.01, respectively. From Figure **5.11 - 5.17**, we visualize the effects of considering proximal term and show how our proposed FedPARL framework consisting of pruning, activity, and resource-awareness with re-parameterization of FedAvg model performs better than FedAvg and FedProx model. We consider proximal term $\beta = 0$ and $\beta > 1$, and show how the value of β can increase the stability of a heterogeneous FL-IoT setting. We simulate systems heterogeneity of our FL-IoT environment by forcing $0\%, 50\%$, and 90% of the participated clients to be stragglers without adding any proximal term and observe the improved convergence in terms of model training loss and testing accuracy while allowing partial amounts of work with model pruning, activity, and resource-awareness of FedPARL framework in a heterogeneous network. We also simulate the convergence behavior of our FedPARL framework by considering the proximal term and observing robust and stable performance, particularly, in a heterogeneous setting in presence of $0\%, 50\%$, and 90% of stragglers. To the end, we simulate our proposed framework in presence of data heterogeneity. We obtain higher training loss during model training and achieve improved stable accuracy compare to the FedAvg and FedProx approaches.

5.6 Conclusion

In this dissertation chapter, we propose an FL model that can be effectively applied in a resource-constrained IoT environment. The generalization of FL objective functions coupling with the pruning mechanism, and activity and resource-awareness help us to generate a lightweight FL model that can handle systems and statistical heterogeneity. By selecting trustworthy and proficient clients, performing local training with the lightweight model, and allowing variable amounts of work from FL clients, we achieve a robust, stable, and consistent FL model that has remarkable performance within unreliable heterogeneous networks. We have tested our FedPARL framework with various datasets and obtained an improved convergence behavior compared to the existing FL techniques that are implemented with the concept of realistic heterogeneous settings. The core idea of FedAR and FedPARL has been explained in Chapters 4 and 5. There is still a gap when the distributed agents hold heterogeneous model architectures. Chapter 6 will further improve our proposed algorithms by handling the model heterogeneity issues through the knowledge distillation technique.

CHAPTER 6

FedMDP: A FEDERATED LEARNING FRAMEWORK TO HANDLE MODEL AND SYSTEMS HETEROGENEITY VIA KNOWLEDGE DISTILLATION AND DYNAMIC LOCAL TASK ALLOCATION

6.1 Abstract

The advancement of technology, improvement of network infrastructures, and wide availability of internet open up the door of new opportunities to perform on-device inference. Realizing the potential of such advancement, an advanced machine learning (ML) technique was invented called *Federated Learning (FL)* that facilitates the formation of a powerful model without exposing user data. While successful, it does not consider the case where the selected FL agents independently crafts their local model with heterogeneous architecture and performs computational tasks based on their available resources. In the original FL model, all agents need to agree on a uniform model architecture and are assigned a uniform computational task. However, in a real-life FL setting, situation could be different as some agents may not be interested to share their local model architecture details because of privacy and security concerns, while any FL agent might find the assigned task computationally challenging. Further, the heterogeneous local model architectures cannot be aggregated together on the FL server following the traditional approaches. Further, during the training process, we may observe a large number of straggler agents due to resource-limitation issues and the stragglers may not be able to accomplish the given computational task within a preset time window. To address the above-mentioned challenges regarding agent's local model and resource heterogeneity, we propose an FL framework, *FedMDP* that can effectively handle federated agents possessing nonidentical local model structure as well as variant local resources using

knowledge distillation and dynamic local task allocation techniques. We tested our framework on MNIST and CIFAR100 dataset and observed significant improvement in accuracy in a highly heterogeneous environment. By considering 10 uniquely designed model of the agents, we achieved 15% gain on average comparing to the accuracy of the traditional learning methods and observe a few percent lower accuracy comparing to the case if the agents' local datasets were pooled and made available for all the network agents.

6.2 Introduction

As we already mentioned in our previous chapter $2-5$, Federated Learning (FL) is a privacy preserving distributed machine learning (ML) scheme that does not require to obtain any user's data in a centralized location, instead, a global predictor model is constructed that is learned through the aggregated knowledge of participating users. On top of preserving privacy, FL provides several other benefits such as autonomy [NDR19], security [MLD$^+$20] and efficiency [SWS20] because of its on-device training and distributed decision-making. The authors in [HDJ21] discussed the decentralization mechanism of FL and analyzed how it is effective than gossip learning. Despite the benefits, the FL process faces many challenges among which heterogeneity is one the major concerns and appears throughout the learning process. The participated agents in the FL process may have heterogeneous resources in terms of their computational power and bandwidth that was partially solved by the prior works using asynchronous FL technique and further refined through active sampling [NY19, LLH$^+$20]. However, when we have a majority of the agents as stragglers, then asynchronous FL or active sampling failed to work effectively. Besides, statistical heterogeneity problem, i.e., nonuniform distribution of data among

the agents can also be observed in an FL environment [ZXB+21, SSCE21]. As a result, the agent's local update can be dispersed that may prolong the model convergence. Moreover, the participated FL agents may also possess heterogeneous local models that may cause issues while performing aggregation of the FL server following the state-of-the-art FL process. Some of the FL agents may have simple model while some agents contain complicated and large model architecture. It is because a agent may have sufficient resources to generate a large model while another agent may not be capable to process its local data and generate such a large model. The original FL work assume that the FL agents agree on a particular model architecture and all local models as well as global model follow that design. However, if we consider that from real-world setting, then any agent may have desire to construct their own unique model. Such situations can be arrived in areas like supply chain, health care, AI services, and finance. For instance, when several department of a supply chain company collaborate without sharing private data, they can craft their own model as per distinct specification. Such supply-chain company would not like to reveal their models because of privacy, and security concerns. Besides, we can consider AI-enabled chat bots that are used for customer service and different companies may have dozens of such service bots. Each service bots may have to deal with different customers and solves variant tasks. It would be beneficial if knowledge of one bot can be shared with others without compromising independency and privacy. Here comes the motivation of FL that preserves privacy though on-device model training and enables each device to learn global knowledge independently. However, one of the core challenges that FL faces is statistical heterogeneity and one unsophisticated way to tackle the statistical heterogeneity is to allow individual model for each agent. Different frameworks, e.g., meta-learning [KFBT19], transfer learning [ZXLJ21], multi-task learning [SCST17], Bayesian [C+19] proposed differ-

ent approaches to handle statistical heterogeneity for non-IID data and achieved acceptable performance; however, their approaches needs to perform model customization of the agents up to a certain level. So, instead of centralizing control over agents' model, i.e., customizing agents' model, a mechanism that can enable full model independency can completely tackle the issues of statistical heterogeneity. The authors in [MMZG21] proposed an FL model, FedMDR based on weighted geometric median, which is resilient to the corrupted model injection. The authors in [JOK+18, ASK19, LW19, LKSJ20] also proposed different FL knowledge distillation technique to handle heterogeneous agent model but none of the approaches are effective while applying on a resource-constrained environment. Another challenge of FL method is systems heterogeneity and it becomes critical when we apply FL in a resource-constrained IoT environment [ITW+22]. According to the traditional FL method, the task publisher assigns a uniform task to all the selected agents. However, the agents' resources may be limited or heterogeneous and all the selected agents would not be able to perform the assigned task. As a consequence, the FL server may need to wait for a long time for getting update from the straggler agents and it may prolong the overall model convergence. According to the authors of [MMR+17, BEG+19a], one solution is to drop the straggler agents or not selecting them during model training. For instance, the authors in [AKP21] designed a federated edge learning framework that can select a subset of edge devices as participants by analyzing the downlink channel conditions. However, if the majority of the selected agents are stragglers, then we may have only a few active agents that can significantly reduce the model performance, or some straggler agents may have higher volume of data [LSZ+18]. Beyond model and systems heterogeneity, diverge local model update from the agents is also an issue of federated networks that can be mainly occurred due to false model injection [KXN+20]. The authors in [CBM20]

proposed an FL model, FedMax that can mitigate communication overhead by applying a technique of restricting activation-divergence of the participated agents. However, they did not consider the straggler effects due to resource-constrained agents. The core research question that we try to address in this chapter is how we can carry out FL process when the agents have heterogeneous model architectures which is blackbox to others, and the federated networks have a large number of resource-constrained agents that could become stragglers or can infuse false model during global model update.

6.2.1 Contributions

On the whole, the main objectives of this chapter can be stated as follows:

- We developed an FL framework, FedMDP that can be effectively applied in a highly heterogeneous resource-constrained environment. Our proposed framework can be applied in such a FL setting, where the network agents possess unknown model architectures and the participated agents are resource-constrained devices.
- We propose to use transfer learning and knowledge distillation to develop a universal framework that enables federated learning when each agent owns not only their private data, but also uniquely designed models and heterogeneous resources. We develop a module that translates knowledge between participants.
- We enable variable computational tasks for the participated agents that can significantly mitigate the straggler effects by considering every little computational tasks performed by the agents.

6.3 System Description

6.3.1 Problem Definition

We assume that there are n number of agents in an FL environment. Each agent p owns a small labeled dataset $\mathcal{D}_p := \{(x_i^p, y_i)\}_{i=1}^{N_p}$ and the dataset does not need to be drawn from the similar type of distribution. We considered a large public dataset $\mathcal{D}_0 := \{(x_i^0, y_i^0)\}_{i=1}^{N_0}$ that is resided on the server and all FL agents can access that. We enable each user to independently design its own model f_p that is used to perform an assigned computational task. Therefore, each agent may have heterogeneous model architectures. Unlike traditional FL methods,, we aim to preserve agent privacy moving one-step forward by not sharing even the hyper-parameters of the agents. Therefore, a agent model information is not be known to anyone and on top of that, the hyper-parameters of the agent models are not exposed. It would protect the FL process from leaking sensitive information due to void knowledge about agent model architectures. Now, the resource-constrained FL agents may struggle to carry-out learning process if the assigned computational task is too overwhelming. That could result in slow learning process and a majority of such slow learning agents can prolong the model convergence time. We can eliminate that issue by assigning computational tasks among the agents based on their resource-availability. On the whole, the main goal of this chapter is to develop an FL model that improves the performance of each agent's model f_p using publicly accessible data $\mathcal{D}_0$ as well as on-device data $\mathcal{D}_p$, and remove the straggler effects by assigning computational tasks based on the agent resources.

6.3.2 Proposed Approach

Our proposed FedMDP framework has three phases. In the first phase, we leverage the federated distillation technique by translating the local knowledge. In the second phase, we enable variable local computational tasks based on the agents' resources. Finally, on the third phase, we integrate the distillation technique and dynamic task allocation technique to handle model and system heterogeneity of the network agents.

Leveraging Federated Distillation Technique

The key to handle model heterogeneity of the FL agents is communication. In particular, we need to design a translation protocol that enables interpreting the knowledge of a local model that is understandable to the server and to its peers. We consider that each agent has a private dataset with a uniquely designed local model. To carry-out FL process, each agent's local model needs to translate to a standard format. To translate the knowledge of the local models, we propose to employ knowledge distillation technique, where the smaller local models are trained based on the large-sized public dataset residing on the server. That means the translator will be developed using knowledge distillation technique. The central server is mainly responsible for collecting the translated knowledge and spreading a consensus across the FL network. Further, the agents share their output class scores which is evaluated considering the agent's performance on the public dataset.

At first, each participated federated agents generates a model using its local data. Then each agent computes class scores on the public dataset and shares with the server. The server generates an aggregated class scores that are received from all the agents. That aggregated class score is the latest consensus for the public dataset. For the next iteration round, each agent downloads the latest consensus and trains

its model based on the public dataset. After that, each agent retrain its local model utilizing its private data and computes an updated class score for the public dataset. Each agent again shares its updated class score and the server generates a latest consensus. The server-agent interaction continues until the consensus class score reach to a target. The working procedure of federated distillation technique is explained below:

1. At the starting of each federated distillation round, the FL server selects a subset of interested agent for the training process and each selected agent can synchronize with the server by downloading the latest soft-labels Y^{Pub} (aggregation of all previously participated agents soft-labels) on public dataset.
2. Each selected agents update their models through model distillation technique. The publicly available dataset and downloaded latest soft-labels are used to generate a distilled model on each agent side.
3. Each agent train the distilled model by applying its own-device local data and generate an updated local model, i.e., each agent learns from the global knowledge.
4. After that, the local model's class scores on the public dataset are generated.
5. Each agent shares the latest soft-label or class score with the FL server.
6. The server performs aggregation on all the collected soft-labels from the agents and computes an aggregated soft-labels Y^{Pub} for the next communication round.

Enabling Partial Computational Tasks

In a resource-constrained FL environment, each agent may not be able to perform a given computational task. If we consider the conventional FedAvg [MMR+17]

algorithm, then an uniform task is assigned to all the selected agents. However, the selected agents may not be homogeneous in terms of their resources (e.g., model architecture, processing power, memory, bandwidth). As a consequence, the majority of the agents would become stragglers during a training process and model convergence would be prolonged. The straggler issue can be observed in the distillation approach, particularly, when a agent applies a distilled model on its private data and generates a new local model (Step 3 in Section 6.3.2). Therefore, it is not reasonable to assign a same computational tasks to all the selected agents. Instead of that, enabling dynamic assignment of the local computational tasks by analyzing the agent's resources can be effective for a resource-constrained FL environment. Allowing a flexible amount of work helps to solve local objectives inexactly and assists to tune up the number of communication vs. local computations. While too many local epochs can overfit the model, a smaller number of local epochs increases communication overhead as well as convergence time [IA20]. Therefore, it is required to set local epoch through proper tuning to ensure robust convergence. We incorporate a generalization of FedAvg algorithm that entitles the straggler agent to perform partial amount of works instead of the whole task. We can allow partial works for our federated agents that are selected through trust and resource-aware strategy and we can define the ϕ_p^i -inexactness for federated agent p at training round i:

Definition 2 $\left(\varphi_p^i\right.$ **-inexact solution** $\left.\right)$. Let us consider a function $\mathcal{G}_p\left(w; w_i\right) = \mathcal{F}_p(w) + \frac{\beta}{2}\left\|w - w_i\right\|^2$, and $\varphi \in [0, 1]$, we call w^* is a φ_k^c -inexact solution of $\min_w \mathcal{G}_p\left(w; w_i\right)$ if $\left\|\nabla \mathcal{G}_p\left(w^*; w_i\right)\right\| \leq \varphi_k^i \left\|\nabla \mathcal{G}_p\left(w_i; w_i\right)\right\|$, where $\nabla \mathcal{G}_p\left(w; w_i\right) = \nabla \mathcal{F}_p(w) + \beta\left(w - w_i\right)$.

Here, φ_p^i determines how much local computation is needed to perform by the device p in communication round i to solve local problems. That means, φ_p^i is the

representation of the variable local iterations of the agents. The systems heterogeneity can be handled by relaxing the φ_p^i -inexactness.

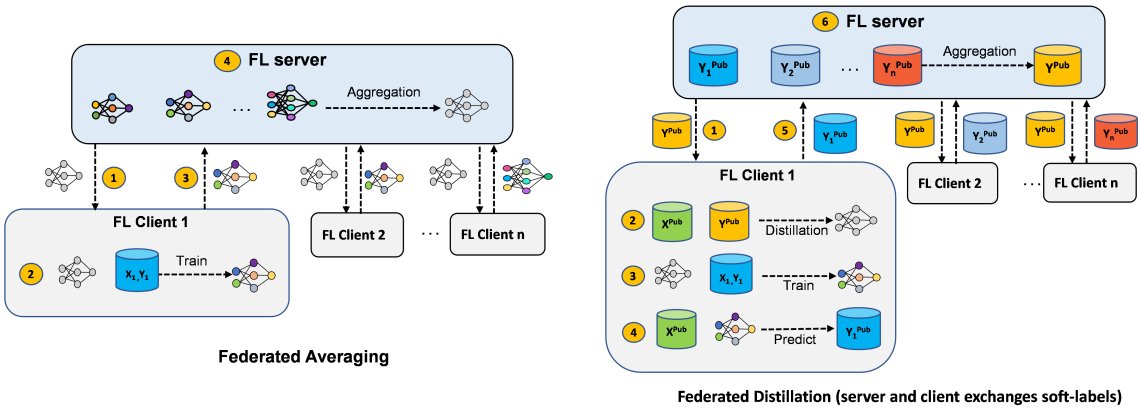


Figure 6.1: The working process of Federated Averaging and Federated Distillation. In Federated Averaging, training information are transferred between server and agents via model parameters. In Federated Distillation approach, the similar information is shared via soft-label predictions Y^{pub} by the agents on a publicly shared data set X^{pub}.

In Figure **6.1**, we present a conceptual visualization of allowing partial amounts of work to be performed by 3 heterogeneous agents. From the figure, we can see that, agent 1 and n are performing 30% and 48% of the overall tasks due to resource-limitations while the second agent is performing the whole task because of its available resources. If we explain it in more simplified way, then let consider that the task publisher expects 100 local epochs to be performed by all the selected FL agents. However, due to resource-constraint issues, some of the agents may not be able to perform 100 local epoch for generating their local models. For such a case, the weak agents are allowed to perform lower number of local epochs, e.g., the first and forth agents need to perform only 30 and 48 local epoch if the overall computational task is 100 local epochs for that training round.

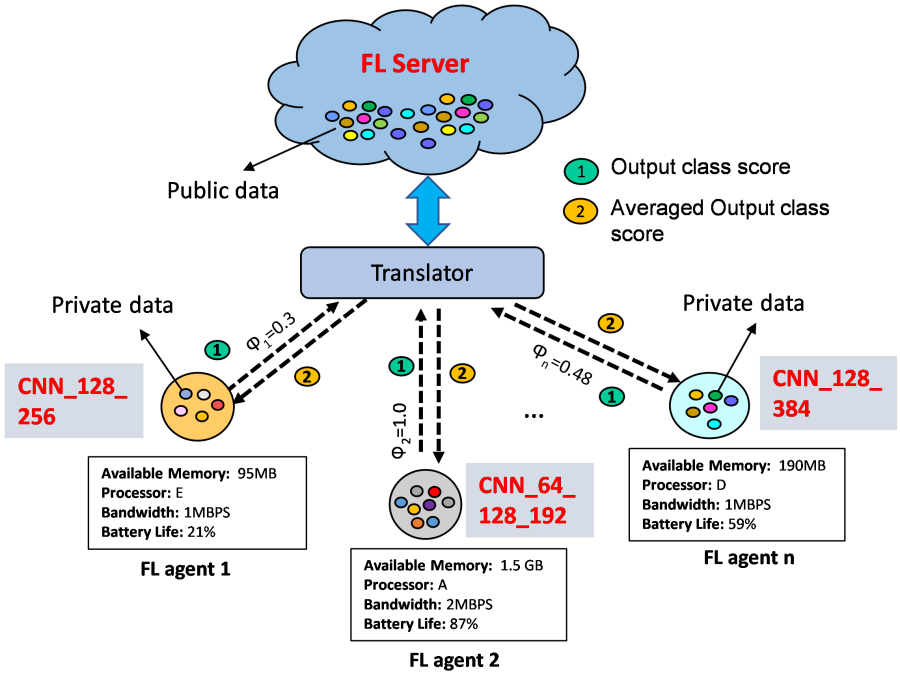


Figure 6.2: Handling model heterogeneity and systems heterogeneity of federated agents through our proposed approach.

Integrating Federated Distillation and Dynamic Local Task Allocation

After leveraging federated distillation for translating local knowledge and enabling dynamic local task allocation to handle straggler agents, we propose an FL framework by integrating both of them which is capable to handle model and system heterogeneity. We graphically represented our proposed *FedMDP* framework in Figure **6.2**. From that figure, we can see the agents hold variant resources in terms of their available memory, processor, bandwidth and battery life, and possess heterogeneous model architectures (e.g., CNN_128_256, CNN_64_128_192, CNN_128_384, etc.). Each agent performs variable computational task as per their resources (e.g., 0.3, 1.0, 0.48) and generates a local model, After that, each local model compute a class score on the available dataset and share that with the FL server. On the server, we have a translator that aggregates the generated class score that holds

the overall feedback of the participated network agents. To reveal more technical insight, we presented the details of our *FedMDP* framework in Algorithm **9**.

Algorithm 9: FedMDP Framework for enabling FL for heterogeneous systems and heterogeneous models. The global model of i^{th} training round is represented by G^i and $\mathcal{P}$ denotes the selected agents for FL process.

1 **Input:** Public dataset $\mathcal{D}_0$, private datasets $\mathcal{D}_\mathcal{P}$, independently designed model $f_\mathcal{P}, k = 1 \dots m$.
2 **Output:** Trained model $f_\mathcal{P}$.
3 **Registration:** Each interested agent, $\mathcal{I}_a$ commits registration.
4 **Initialize** G^i, set timeout t^i, and select a set of agent, $\mathcal{P}$ for training
5 Transfer learning: Each $\mathcal{P}$ trains $f_\mathcal{P}$ to convergence on the $\mathcal{D}_0$ and then on $\mathcal{D}_\mathcal{P}$.
6 **for** i $= 1, 2 \dots$ **do**
7 Communicate: Each $\mathcal{P}$ computes the class scores $f_\mathcal{P}(x_i^0)$ on the public dataset, and transmits the result to a central server.
8 Aggregate: The server computes an updated consensus, which is an average $\tilde{f}(x_i^0) = \frac{1}{m} \sum_\mathcal{P} f_\mathcal{P}(x_i^0)$.
9 Distribute: Each $\mathcal{P}$ downloads the updated consensus $\tilde{f}(x_i^0)$.
10 Each $\mathcal{P}$ finds a $w_\mathcal{P}^{i+1}$ which is a φ_k^i -inexact minimizer of: $w_\mathcal{P}^{i+1} = F_\mathcal{P}(w) + \frac{\beta}{2} \|w - w^i\|^2$ and determines maximum feasible number of local epochs, $E_{par}^\mathcal{P}$.
11 Digest: Each $\mathcal{P}$ trains its model $f_\mathcal{P}$ to approach the consensus $\tilde{f}$ on the public dataset $\mathcal{D}_0$.
12 Revisit: Each $\mathcal{P}$ trains $f_\mathcal{P}$ on $\mathcal{D}_\mathcal{P}$ for the determined feasible local epochs, $E_{par}^\mathcal{P}$.

Here, we consider a public dataset $\mathcal{D}_0$ (which is available to all federated agents), private datasets (which is possessed by each agent and may vary from agent to agent), and individual agent's model architecture as input (line **1**). The output that we expect for each agent is a trained model $f_\mathcal{P}$ that learns from the global knowledge and on-device data (line **2**). Initially, each agent needs to register itself to join in a network (line **3**). After that, the global model, FL participants, training time window, number of participated agents as well as communication round are

initialized (line **4**). Each agent trains its local model $f_{\mathcal{P}}$ to convergence on the public dataset $\mathcal{D}_0$ and then on its private dataset $\mathcal{D}_{\mathcal{P}}$ (line **5**). On each communication round i, each agent generates a score by classifying all the sample of the public dataset $\mathcal{D}_0$ and shares the class score with the FL server (line **6-7**). Upon receiving class scores from all the participated agents, the server performs aggregation on the class scores and generates a consensus that reflects the cumulative knowledge of all the participated agents (line **8**). The latest consensus is shared with all the participated agents and each agent download that updated consensus $\tilde{f}\left(x_i^0\right)$ (line **9**). After that, each participated agent exploits its local solver to determine inexact minimizer $\varphi_{\mathcal{P}}^i$ for solving its objective function, i.e., the number of local epochs that is viable to perform locally in order to evaluate the class score (line **10**). Through on-device model training, each agent approach towards consensus $\tilde{f}$ for public dataset $\mathcal{D}_0$ (line **10**). Finally, each agent $\mathcal{P}$ train its local model $f_{\mathcal{P}}$ on its private dataset $\mathcal{D}_{\mathcal{P}}$ for the determined feasible local epoch, $E_{par}^{\mathcal{P}}$.

6.4 Experimental Evaluation

We perform the evaluation of our proposed FL framework in two different settings. In the first test experiment, we consider the MNIST as the public data and a subset of FEMNIST as the private data. We prepare i.i.d. simulation setting by randomly selecting samples from FEMNIST as the private dataset of the agents. For non-i.i.d. case, each participated agent is given only the letters written by a writer (instead of giving handwriting of all writers), and the agent need to classify letters of all writers during testing period. In the second test experiment, we consider CIFAR10 as the public dataset and CIFAR100 as the private dataset, which possesses 100 subclasses with a 20 superclasses, e.g., leopard, tiger, wolf, bear and lion. The main prediction

task for the second experiment is to classify image samples into correct subclasses while in the non i.i.d. case, each participated agent holds image samples of one subclass from every superclass, and that agent needs to classify the data that are from other subclassess of every superclass. For instance, an FL agent who has seen leopard during on-device model training is assigned task to classify tiger, wolf, bear, or, lion. The knowledge sharing mechanism of the FL process makes it possible to predict unforeseen events or objects. We presented the dataset details in Table **1**.

Table 6.1: Dataset details.

Dataset	**Class**	**Training sample**	**Testing sample**	**Total**
CIFAR100	100	50000	10000	60000
MNIST	9	60000	10000	70000

In our FL simulation setting, we consider 10 FL participants that has unique convolutional neural networks (CNNs). The CNNs differ in terms of number of layers and number of channels. In Figure **6.3**, we presented the details of the model architectures of the FL agents including their layer, output shape, and number of parameters. From the figure, it is clear that each agent has uniquely designed model architecture. Now, if we want to aggregate the heterogeneous model architectures, then the traditional FL methods (which performs weighted average of the local model) would fail to generate a global model. To tackle such issues, initially, the participants are trained based on the public dataset until they reach a target convergence. Then, each participant carries-out on-device training on the private dataset. After that, the participants shares the output class scores on the public data and the server generates an aggregated consensus on the labels of the publicly available data. The aggregated consensus is shared with the participants and they tune-up their model accordingly. We evaluated the pre-trained accuracy of the agents considering MNIST and CIFAR100 dataset. For MNIST dataset,we set up

three convolutional layer filters and a dropout rate, and observed an accuracy of above 95% for all 10 participated agents (Table **2**). In turn, For CIFAR100 dataset, we set up four convolutional layer filters and a dropout rate, that gives us an accuracy of above 72% for all 10 participated agents (Table **3**). We simulated the model accuracy for MNIST IID dataset by not considering any partial works and we can observe how some of the agents' performance degrades due to straggler effects (see Figure **6.4(a)**). However, leveraging our proposed FedMDP model (which selects effective agents and also allows partial works) generates an effective global model that helps every agent including the stragglers to improve their local model (see Figure **6.4(b)**). Further, we simulated the model accuracy for MNIST non-IID dataset by again not considering any partial works and we can observe how some of the agents' performance becomes even worse due to straggler effects in non-IID setting (see Figure **6.5(a)**). However, such performance degradation are effectively handled by our FedMDP model which can deal with statistical heterogeneity and can scale up any diverse local model update (see Figure **6.5(b)**).

Table 6.2: Agent pre-trained accuracy for MNIST dataset.

Model	Three Conv. Layer Filters n_1 —n_2 —n_3	Dropout rate	Pre-trained Accuracy
1	128—256 — None	0.2	96.4%
2	128 — 384 — None	0.2	96.6%
3	128 — 512 — None	0.2	96.0%
4	256 — 256 — None	0.3	98.2%
5	256 — 512 — None	0.4	95.3%
6	64 — 128 — 256	0.2	98.3%
7	64 — 128 — 192	0.2	98.4%
8	128 — 192 — 256	0.2	97.9%
9	128 — 128 — 128	0.3	98.9%
10	128 — 128 — 192	0.3	97.4%

However, we realize that assigning an uniform computational task could be overwhelming for any of the participated agents and thus, we infused our dynamic

Layer	Model 1 (CNN 128_256)		Model 2 (CNN 128_384)		Model 3 (CNN 128_512)		Model 4 (CNN 256_512)		Model 5 (CNN 64_128_256)	
	Output Shape	P	Output Shape	P	Output Shape	P	Output Shape	P	Output Shape	P
Input Layer	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0
Reshape	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0
Conv2D	(None, 32, 32, 128)	3584	(None, 32, 32, 128)	3584	(None, 32, 32, 128)	3584	(None, 32, 32, 256)	7168	(None, 32, 32, 256)	7168
Batch Normalization	(None, 32, 32, 128)	512	(None, 32, 32, 128)	512	(None, 32, 32, 128)	512	(None, 32, 32, 256)	1024	(None, 32, 32, 256)	1024
Activation	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 256)	0	(None, 32, 32, 256)	0
Dropout	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 256)	0	(None, 32, 32, 256)	0
Average Pooling2D	(None, 32, 32, 128)	0	(None, 32, 32, 384)	0	(None, 32, 32, 128)	0	(None, 32, 32, 256)	0	(None, 32, 32, 256)	0
Conv2D_10	(None, 15, 15, 256)	295168	(None, 15, 15, 384)	442752	(None, 15, 15, 512)	590336	(None, 15, 15, 256)	590080	(None, 15, 15, 512)	1180160
Batch Normalization	–	–	(None, 15, 15, 384)	1536	(None, 32, 32, 512)	2048	(None, 15, 15, 256)	1024	(None, 15, 15, 512)	2048
Activation	(None, 32, 32, 256)	0	(None, 15, 15, 384)	0	(None, 32, 32, 512)	0	(None, 15, 15, 256)	0	(None, 15, 15, 512)	0
Dropout	(None, 32, 32, 256)	0	(None, 15, 15, 384)	0	(None, 32, 32, 512)	0	(None, 15, 15, 256)	0	(None, 15, 15, 512)	0
Average Pooling2D	(None, 32, 32, 128)	0	(None, 32, 32, 384)	0	(None, 32, 32, 128)	0	(None, 32, 32, 256)	0	(None, 32, 32, 256)	0
Conv2D_12	(None, 15, 15, 256)	295168	(None, 15, 15, 384)	442752	(None, 15, 15, 512)	590336	(None, 15, 15, 256)	590080	(None, 15, 15, 512)	1180160
Batch Normalization	–	–	(None, 15, 15, 384)	1536	(None, 32, 32, 512)	2048	(None, 15, 15, 256)	1024	(None, 15, 15, 512)	2048
Activation	(None, 32, 32, 256)	0	(None, 15, 15, 384)	0	(None, 32, 32, 512)	0	(None, 15, 15, 256)	0	(None, 15, 15, 512)	0
Dropout	(None, 32, 32, 256)	0	(None, 15, 15, 384)	0	(None, 32, 32, 512)	0	(None, 15, 15, 256)	0	(None, 15, 15, 512)	0
Flatten	(None, 57600)	0	(None, 86400)	0	(None, 115200)	0	(None, 57600)	0	(None, 115200)	0
Dense	(None, 16)	921600	(None, 16)	1382400	(None, 16)	1843200	(None, 16)	921600	(None, 16)	1843200
Activation	(None, 16)	0	(None, 16)	0	(None, 16)	0	(None, 16)	0	(None, 16)	0

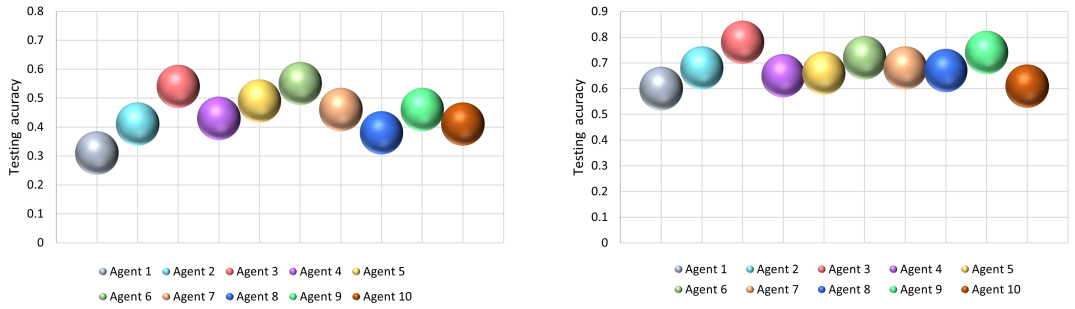
Layer	Model 6 (CNN 64_128_256)		Model 7 (CNN 64_128_192)		Model 8 (CNN 128_192_256)		Model 9 (CNN 128_128_128)		Model 10 (CNN 128_128_192)	
	Output Shape	P	Output Shape	P	Output Shape	P	Output Shape	P	Output Shape	P
Input Layer	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0
Reshape	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0	(None, 32, 32, 3)	0
Conv2D	(None, 32, 32, 64)	1792	(None, 32, 32, 64)	1792	(None, 32, 32, 128)	3584	(None, 32, 32, 128)	3584	(None, 32, 32, 128)	3584
Batch Normalization	(None, 32, 32, 64)	256	(None, 32, 32, 64)	256	(None, 32, 32, 128)	512	(None, 32, 32, 128)	512	(None, 32, 32, 128)	512
Activation	(None, 32, 32, 64)	0	(None, 32, 32, 64)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0
Dropout	(None, 32, 32, 64)	0	(None, 32, 32, 64)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0
Average Pooling2D	(None, 32, 32, 64)	0	(None, 32, 32, 64)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0	(None, 32, 32, 128)	0
Conv2D	(None, 16, 16, 128)	32896	(None, 16, 16, 128)	32896	(None, 16, 16, 192)	98496	(None, 16, 16, 128)	65664	(None, 16, 16, 128)	65664
Batch Normalization	(None, 16, 16, 128)	512	(None, 16, 16, 128)	512	(None, 16, 16, 192)	768	(None, 16, 16, 128)	512	(None, 16, 16, 128)	512
Activation	(None, 16, 16, 128)	0	(None, 16, 16, 128)	0	(None, 16, 16, 192)	0	(None, 16, 16, 128)	0	(None, 16, 16, 128)	0
Dropout	(None, 16, 16, 128)	0	(None, 16, 16, 128)	0	(None, 16, 16, 192)	0	(None, 16, 16, 128)	0	(None, 16, 16, 128)	0
Average Pooling2D	(None, 8, 8, 128)	0	(None, 8, 8, 128)	0	(None, 8, 8, 192)	0	(None, 8, 8, 128)	0	(None, 8, 8, 128)	0
Conv2D	(None, 3, 3, 256)	295168	(None, 3, 3, 192)	221376	(None, 3, 3, 256)	442624	(None, 3, 3, 128)	147584	(None, 3, 3, 192)	221376
Batch Normalization	(None, 3, 3, 256)	1024	(None, 3, 3, 192)	768	(None, 3, 3, 256)	1024	(None, 3, 3, 128)	512	(None, 3, 3, 192)	768
Activation	(None, 3, 3, 256)	0	(None, 3, 3, 192)	0	(None, 3, 3, 256)	0	(None, 3, 3, 128)	0	(None, 3, 3, 192)	0
Dropout	(None, 3, 3, 256)	0	(None, 3, 3, 192)	0	(None, 3, 3, 256)	0	(None, 3, 3, 128)	0	(None, 3, 3, 192)	0
Flatten	(None, 2304)	0	(None, 1728)	0	(None, 2304)	0	(None, 1152)	0	(None, 1728)	0
Dense	(None, 16)	36864	(None, 16)	1382400	(None, 16)	36864	(None, 16)	18432	(None, 16)	27648
Activation	(None, 16)	0	(None, 16)	0	(None, 16)	0	(None, 16)	0	(None, 16)	0

Figure 6.3: Model architecture of 10 different federated agents [P = parameters].

Table 6.3: Agent pre-trained accuracy for CIFAR100 dataset.

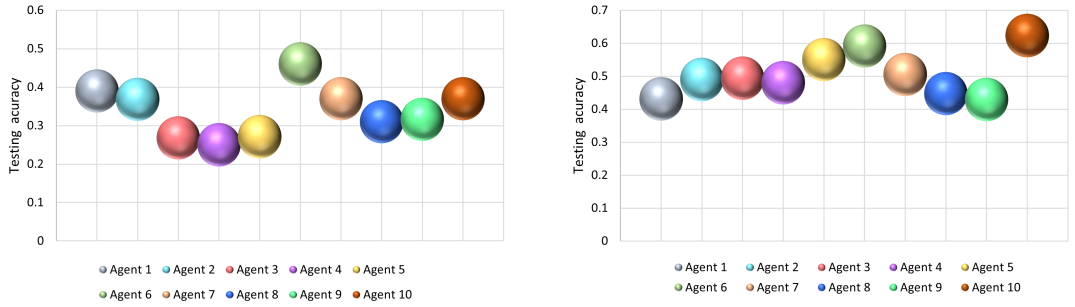
Model	Four Conv. Layer Filters n_1 —n_2 —n_3 — n_4	Dropout rate	Pre-trained Accuracy
1	128 — 256 — None — None	0.2	72.3%
2	128 — 128 — 192 — None	0.2	79.9%
3	64 — 64 — 64 — None	0.2	73.5%
4	128 — 64 — 64 — None	0.3	76.9%
5	64 — 64 — 128 — None	0.4	75.8%
6	64 — 128 — 256 — None	0.2	77.4%
7	64 — 128 — 192 — None	0.2	78.7%
8	128 — 192 — 256 — None	0.2	74.6%
9	128 — 128 — 128 — None	0.3	78.7%
10	64 — 64 — 64 — 64	0.2	75.9%

resource allocation strategy (i.e., allowing partial works) for assigning local computational tasks according to the agent's resource availability. To measure the effects



(a) (b)

Figure 6.4: Model accuracy (a) without allowing partial works and (b) enabling partial works (our proposed approach) considering federated agents including stragglers in presence of model heterogeneity and systems heterogeneity using MNIST IID dataset.



(a) (b)

Figure 6.5: Model accuracy (a) without allowing partial works and (b) enabling partial works (our proposed approach) considering federated agents including stragglers in presence of model heterogeneity and systems heterogeneity using MNIST Non-IID dataset.

of enabling partial works of the agents, we perform a simulation of our federated setting considering system heterogeneity. We assume that, a global clock cycle is remained during the FL process which can specify the time window for executing an assigned computational task. We also assume that, a task publisher will define a global epoch E which needs to be executed by all resource-sufficient participants. In case, a selected agent k has resource-constraints, then that agent can perform fewer epochs by evaluating the feasible amount of computational works it can perform on communication round i (φ_k^c) using a function consisting of a global cycle

and its available resources. We evaluate the learning progress of our considered 10 heterogeneous agents to how those agents collaborate with each other despite having model and system heterogeneity. In Figure **6.6(a)** and Figure **6.6(b)**, we present the simulation results of the accuracy of those 10 agents considering both IID and non-IID setting for the MNIST dataset, which depicts that the model quality of each agent is improving as the communication round increases. We also considered CIFAR100 dataset to check the performance of our FedMDP framework. In Figure **6.6(c)** and Figure **6.6(d)**, we show the the accuracy of the agents considering both CIFAR100 IID and non-IID dataset, which demonstrate the effectiveness of our proposed FedMDP framework. From the simulation, we can observe that the agents achieve higher accuracy in both of the IID settings as similar type of data are distributed among the agents in such a case. As a result, if an agent fail to perform the whole computational task, other agents that capture the similar knowledge can cover up the loss. In turn, in non-IID settings, we achieve a little less accuracy than IID setting which is obvious, still the agents achieve a notable accuracy within a very few communication round despite having high heterogeneity in their local data.

To evaluate the performance of our proposed model in presence of straggler effects, we defined various number of agents that could be stragglers, e.g., $0\%, 10\%, 20\%, 50\%, 90\%$, where 0% straggler indicates that all the participated agents are able to perform the assigned computational tasks (i.e., global epoch E), and 90% agents are stragglers means only 10% agents could perform the whole task. In such a situation, the state-of-the-art FedAvg algorithm simply drops the stragglers from the training round which could prolong the convergence time, or the global model could never reach to the target convergence of there is a high number of stragglers. Instead of that, if we could count every little computational tasks that are performed by the resource-constrained agents, then we would

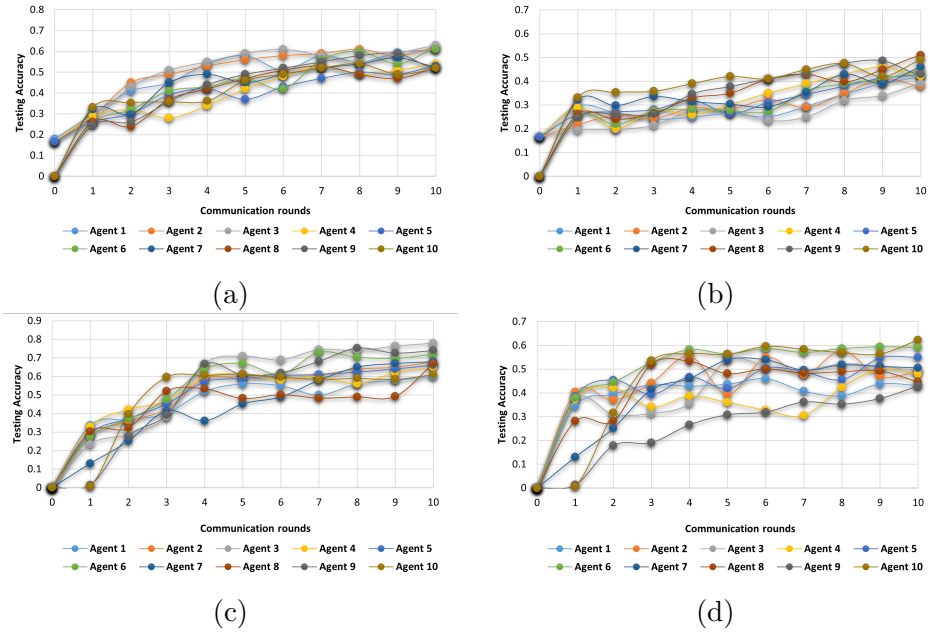


Figure 6.6: Agent's learning progress in different communication rounds in spite of systems and model heterogeneity through transfer learning-based knowledge distillation and leveraging partial computational tasks on (a) CIFAR100 IID dataset, (b) CIFAR100 Non-IID dataset, (c) MNIST IID dataset and (d) MNIST Non-IID dataset.

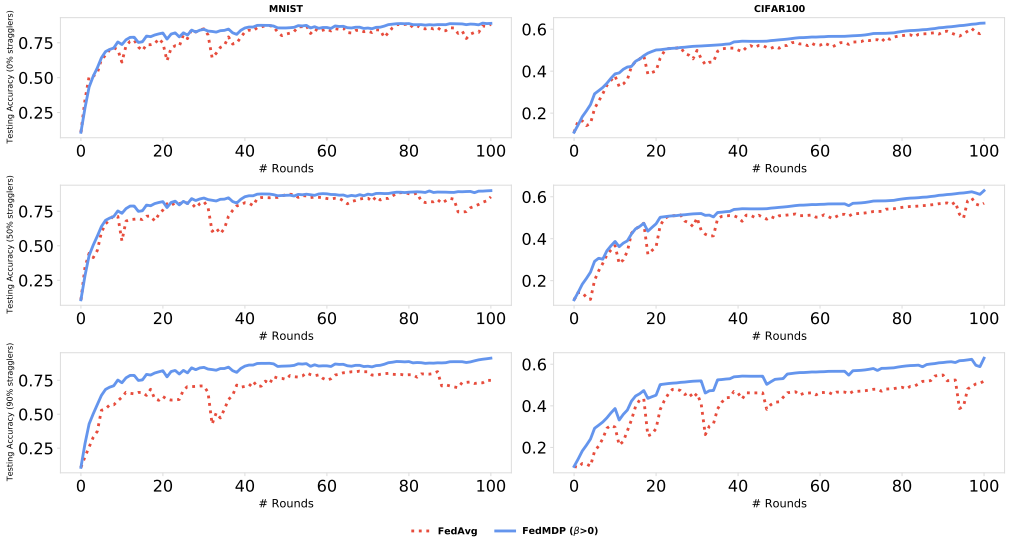


Figure 6.7: Testing accuracy of FedAvg model and our proposed model in presence of different percentages of stragglers (i.e., 0%, 50%, and 90%)

mitigate straggler effects and also the model convergence would be accelerated. In Figure **6.7**, we demonstrated the validation of our hypothesis by simulating the

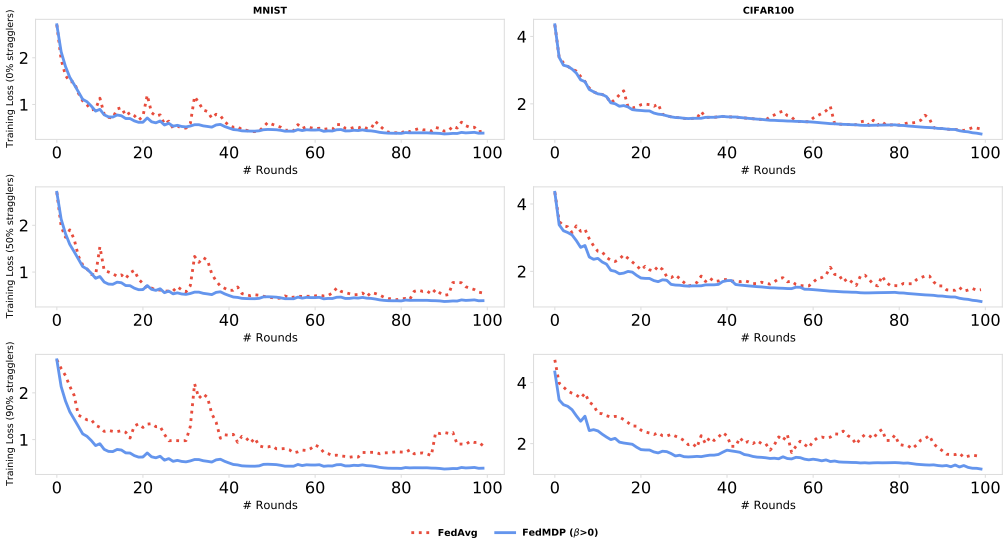


Figure 6.8: Training loss of FedAvg model and our proposed model in presence of different percentages of stragglers (i.e., 0%, 50%, and 90%)

training loss considering different number of stragglers $(0\%, 50\%, 90\%)$ and we achieved higher training loss of the global model compare to the FedAvg method. We also simulated the testing accuracy of FedMDP and FedAvg, and it is clear that the FedMDP outperforms the FedAvg, particularly when a high number of network agents are stragglers (see Figure **6.8**). From these simulations, it is clear that heterogeneity has a negative impact on the agent's learning process and dropping the straggler agent can significantly degrade the model performance. Instead, leveraging our proposed FedMDP model handles model heterogeneity through knowledge distillation technique and also helps to achieve higher model accuracy by considering every little computational tasks performed by the agents.

We also investigate our proposed FedMDP model by considering two different settings. In the first inspection, we limit the agent's local computational task as exactly 1 epoch, i.e., each selected agent needs to perform only a single local iteration on its local data using its local resources and learning from global model. In that investigation, we found that our proposed FedMDP model performs better than the

FedAvg model. In our next investigation, we checked the performance of FedMDP and FedAvg on a synthetic Independent and Identically Distributed Data (IID) dataset that does not have any statistical heterogeneity across different network agents. For such a dataset, the FedAvg method performs better because it simply drops the stragglers, and consider the contributions of other agents that already holds the similar type of knowledge. However, in non-IID dataset, if we simply drop the stragglers, then we may lose valuable information that are possessed by that agent. That is why, in non-IID dataset, our proposed method obtains Superior performance than the FedAvg method.

We analyze several valuable aspects of our simulation results:

1. We evaluate the accuracy an agent could have achieved considering the same simulations settings with the presence of straggler agents. All local data of the agents are pooled together and shared with all the agents (see Table **V**). Our proposed model accelerates the performance of all participated agents only a few percent less than the performance of the pooled data in presence of stragglers.

2. Our proposed model can handle extreme level of model heterogeneity. We considered several models that have low prediction performance, i.e., fully connected neural networks with two layers. If such models contribute equally as the advanced models, then the overall model performance is hindered. If we suppress the contribution of the resource-limited agents that can not perform the whole computational tasks and possess a low-quality model, then our model performs better.

6.5 Conclusion

In this chapter, we proposed an FL framework, *FedMDP* that can deal with agents with heterogeneous local model architectures as well as heterogeneous locally available resources. Through the generalization of FedAvg algorithm and knowledge translation of the federated agents, our developed framework can accelerate the model convergence and upgrade the knowledge of the uniquely designed models of the agents. We tested our framework on various datasets and tasks and we demonstrated the effectiveness of FedMDP even in a highly resource-constrained environment. Our simulations results prove that the infusion of knowledge distillation and allowing partial works significantly improve the model quality of the network agents and also cut-off the negative impacts due to slow agents towards model convergence. In future, we will investigate more sophisticated privacy mechanism while sharing class score in the distillation process and will explore the optimal hyper-parameter tuning for our proposed framework. In next chapter, we will discuss how we leverage our developed FL algorithms in various applications such as improving resilience of critical infrastructures, human activity recognition, and customer's financial distress prediction.

CHAPTER 7

FEDERATED LEARNING APPLICATIONS

[1] Given that we have proposed and evaluated the theoretical algorithms and foundations of our novel FL for resource-constrained IoT environments, i.e., FedAR, FedPARL, and FedMDP, we will explain three major applications of developed algorithms in real-world scenarios. Particularly, this chapter leverages our proposed FL algorithms in Chapters 4, 5, and 6 to deploy in real-life applications. These applications include FL for resilient operation of critical infrastructures (7.1) [IKKA21], human activity recognition (7.2) [IAA21], and financial systems (7.3) [IA22].

7.1 FedResilience: A Federated Learning Application to Improve Resilience of Resource-Constrained Critical Infrastructures

7.1.1 Abstract

Critical infrastructures (e.g., energy and transportation systems) are essential lifelines for most modern sectors and have utmost significance in our daily lives. However, these important domains can fail to operate due to system failures or natural disasters. Though the major disturbances in such critical infrastructures are rare, the severity of such events calls for the development of effective resilience assessment strategies to mitigate relative losses. Traditional critical infrastructure resilience approaches consider that the available critical infrastructure agents are resource-sufficient and agree to exchange local data with the server and other agents. Such

[1]This chapter is an edited version of the author's previous works published in [IKKA21] ©2021 MDPI, [IAA21] ©2021 Elsevier, and [IA22] ©2022 Springer.

assumptions create two issues: (1) uncertainty in reaching convergence while applying learning strategies on resource-constrained critical infrastructure agents, and (2) a huge risk of privacy leakage. By understanding the pressing need to construct an effective resilience model for resource-constrained critical infrastructure, we aim at leveraging a distributed machine learning technique called Federated Learning (FL) to tackle an agent's resource limitations effectively and at the same time keep the agent's information private. Particularly, we focused on predicting the probable outage and resource status of critical infrastructure agents without sharing any local data and carrying out the learning process even when most of the agents are incapable of accomplishing a given computational task. To that end, an FL algorithm is designed specifically for a resource-constrained critical infrastructure environment that could facilitate the training of each agent in a distributed fashion, restrict them from sharing their raw data with any other external entities (e.g., server, neighbor agents), choose proficient clients by analyzing their resources, and allow a partial amount of computation tasks to be performed by the resource-constrained agents. We considered a different number of agents with various stragglers and checked the performance of FedAvg and our proposed FedResilience algorithm with prediction tasks for a probable outage, as well as checking the agents' resource-sharing scope. Our simulation results show that if the majority of the FL agents are stragglers and we drop them from the training process, then the agents learn very slowly and the overall model performance is negatively affected. We also demonstrate that the selection of proficient agents and allowing them to complete only parts of their tasks can significantly improve the knowledge of each agent by eliminating the straggler effects, and the global model convergence is accelerated.

Motivation

Critical infrastructures such as power systems, transportation, fuel, water, and gas are interconnected and are all parts of distinct or interdependent networks that operate cooperatively to produce and distribute essential services [GCU+16]. When a sudden interruption occurs in any of the infrastructures, the resilience technique can assist in the continuation of the operations through its ability to resist, avoid, adapt, and recover swiftly from a disastrous situation. However, it is vital to ensure the resilience of the critical infrastructures as the sectors are interdependent. Using an inter-network communication scheme, entities within the same network can exchange resources, and in an intra-network communication infrastructure, clients residing in different network domains can provide support to each other to enhance the resilience of the system [AIP20]. For instance, most of the critical infrastructures—e.g., transportation networks, telecommunication, and finance and banking sectors—rely on a continuous and stable power supply. However, the prolonged disruption of the operations of critical infrastructures can incur a significant economic loss. One of the recent surveys found that power outages occur for a minimum of one out of four companies in every month [SEC13]. Specifically, in large companies, power outage loss costs over a million dollars an hour and around 150 USD annually [Hus13]. Several works have already been conducted with the aim of improving the resilience of critical infrastructures by applying machine learning (ML) [SVM20], deep learning (DL) [DRSD+19], distributed edge computing [AATM18, AIM20], and transfer learning [ISW19]. All these works constructed their prediction model either by collecting data from the agents (e.g., ML, DL) or receiving an update for some data from the distributed agents (e.g., distributed ML). However, sharing such sensitive data could be privacy-intrusive. An attacker can expose or tamper with data, which may cause the failure of the whole resilience system, and a company may thus face

a huge loss. Besides, in a distributed system, we may observe straggler clients that learn very slowly due to resource-limitation issues and degrade the overall performance of the prediction model [ITW+22].

The conventional resilience approaches that are constructed on the theme of the learning and forecasting of probable outages consider that all critical infrastructure agents (CIAs) have available resources and can perform an assigned computational task. However, in a real-world scenario, any agent may possess low system configurations or may run out of resources. In consequence, some agents may not be able to complete the assigned computational tasks due to the shortage of resources. Therefore, our main motivation for this research is to mitigate the outage loss of the CIAs by developing a novel FL-based prediction model that can preserve privacy and handle the straggler issues in the case of resource-constrained network agents. We propose a novel FL-based strategy that consists of a local CIA, which acts as an intelligent decision-making entity or an FL agent; e.g., a smart factory or an autonomous micro-grid can act as a CIA. A CIA can generate a model based on its available local data (i.e., power demand and resource availability) and share the model with a central fusion center that acts as an FL server. Similarly, the neighboring CIAs share their model to pursue a common goal, and the coordinator generates a global model that learns the outage information and resource sharing scope of all the CIA agents. In case any CIA agent has limited computational resources (e.g., low processing capability, bandwidth) and cannot generate a learning model, the central coordinator is enabled to select proficient clients for the training rounds, allowing partial amounts of work from the resource-constrained CIAs considering their available resources. Therefore, the resource-constrained issues of the grids related to model training would be resolved, and distributed resources could be supplied from the neighbors in case any CIA fails to continue its operation.

Literature Review

The concept of resilience can be generalized for any discipline as a system's capacity to predict and withstand forthcoming shocks, restore the system's normal state swiftly, and adapt with an improved action for handling future catastrophic events. Managing and improving the infrastructure resilience of critical infrastructures has recently attracted the attention of several researchers and, in consequence, several studies related to the modeling and upgrading of systems and networks resilience have been proposed [CLQ+18, Pur18, ATT+20]. The authors in [ANH13] focused on reducing the peak load of the critical infrastructure of power systems by considering multi-agent-based power generation, network grids, and relative demand response status. In their proposed approach, the agents could share their local information only with a central fusion center and were unable to interact with neighboring agents to exchange local resources. Besides, a comprehensive study on modeling the resilience of large-scale critical infrastructure was presented in [WZRB+19]. However, centralized resilience systems become overly complex when a large amount of data is stored, processed, analyzed, and shared from a central fusion center [LCK19, WRS20]. The drawbacks of centralized resilience systems (e.g., scalability, computational power, storage) can be handled with distributed systems and learning resilience schemes [ABDR+14]. The authors of [SVK18] presented a detailed analysis of multi-agent systems (MAS), leveraging distributed intelligence among the network agents through peer-to-peer communication and sharing demand and load status to achieve a common goal. Besides, the authors of [CXL+18] proposed an adaptive synchronization approach for heterogeneous MAS against actuator fault by developing a multi-objective optimization technique to measure the installation capacity of network agents considering power-resilience against disasters [UYO+19, HRM+21]. The usual oper-

ations of smart grids can also be hampered due to cyber attacks that may result in power outage. The authors in [FKAE21] proposed a sequential supervised ML technique to predict cyber attacks in smart grids. Moreover, several works have adapted the strategy of utilizing infrastructure resources to improve the resilience of relative operations [HBK19, GCM+17]. Further, some recent works developed resilience management systems for power systems [IAM20, AIM20, MGA+21, GDN+21, BAKB20], transport [CHR21, BBC+21, AMWK19, PYHH21], urban areas [NM21, EAF+19, CVD19], healthcare [RSB+21, HCGW21, SWJ+21], and production systems [JMC+21, Kus20, BGIWG21] by adding intelligence to the CIAs so that the agents could make autonomous decisions by analyzing the demand–response state. In summary, all the prior works proposed the improvement of resilience either by passing local sensitive data of infrastructure agents to a central fusion center or by sharing such local sensitive data with neighboring agents. However, sharing the sensitive data that reside in the CIAs leads to the risk of privacy violation and can also interrupt the infrastructure operation through data falsification. To prevent that, a recently invented distributed ML technique called Federated Learning (FL) was proposed that can generate a smart model by utilizing edge resources and keeping an agent's information private. As the FL process is completely dependent on the agent's local model update, one of the challenges that the FL process presents is the straggler issues that arise due to the heterogeneity of the systems. System heterogeneity can be referred to as the heterogeneous nature of the agents in terms of their computational power, memory, battery life, or bandwidth. If we apply the FL process considering Internet of Things (IoT) devices, then there is a high chance of observing straggler agents during a training process [KSH+21]. This is because IoT devices are resource-constrained and vulnerable [MKP+21]. If we consider the state-of-the-art FedAvg algorithm [MMR+17], then it simply drops the

straggler agents from the training process. However, dropping the stragglers can degrade the model performance, and also some agents may have valuable data. Instead of this approach, we need a strategy that can effectively handle the stragglers by counting every contribution, irrespective of its size. The authors of [LSZ+18] proposed the FedProx algorithm, which can enable partial amounts of work to be collected from the agents; however, they randomly selected agents for the training round. According to the authors of [CBM20], FedMax outperforms FedProx in terms of communication rounds by applying a strategy of limiting activation-divergence across multiple devices.

To tackle the above-mentioned issues in the context of the resilient operation of critical infrastructures and analyzing the existing works of FL, we are the first to propose a novel FL-based strategy that can predict the probable outages and resource-sharing capabilities of the network agents with the aim of improving resilience. Our FedResilience algorithm can select proficient agents by examining their resources and handle the stragglers by assigning feasible local computational tasks based on their capabilities. Our proposed technique relies on sharing local models of the infrastructure agents instead of sharing sensitive data and, finally, exploits the collaboratively learned knowledge on the probable outages and resource availability status of the whole FL network to enhance the resilience operations.

Contribution

The main contributions of this dissertation chapter are given below:

- To the best of our knowledge, this is the first FL application that can improve the resilience of critical infrastructures through early prediction;
- We present a pathway of collaborative learning for CIAs that enables on-device learning without sharing any data and by exchanging only model information;

- We choose only the proficient agents for the FL training process and enable partial works to be collected from the resource-constrained agents to resolve the straggler issues;
- To demonstrate the effectiveness of Federated Learning in improving resilience by the early prediction of the outages and resource-sharing scope of the agents, we evaluate the prediction performance considering a varying number of stragglers and compare the model with the popular FedAvg [MMR+17] algorithm.

7.1.2 Proposed System Description

FL is a distributed machine learning technique that allows the on-device training of network clients with their local data instead of sharing raw data with the server. Each client generates a local model by optimizing its local objective function that is shared with the FL server. After receiving local models from all participating FL clients, the FL server performs aggregation on the received models and updates a global model which is initialized as well as shared with all network clients at the initial stage of FL training. After that, the updated global model is disseminated to all FL clients, and each FL client tunes their local model by learning from the global model. The FL client–server interaction process is continued until the global model achieves a desired accuracy; hence, the model reaches a target convergence. The overall FL process is presented in Figure 7.1.

In critical infrastructure networks, we may observe heterogeneous agents with varying system configurations and data volumes. Therefore, it is not viable to assign a uniform number of tasks to all the agents that participate in the FL process. The authors in [ITW+22] conducted a comprehensive survey on leveraging FL for IoT devices, where they discussed the possible challenges faced while applying FL

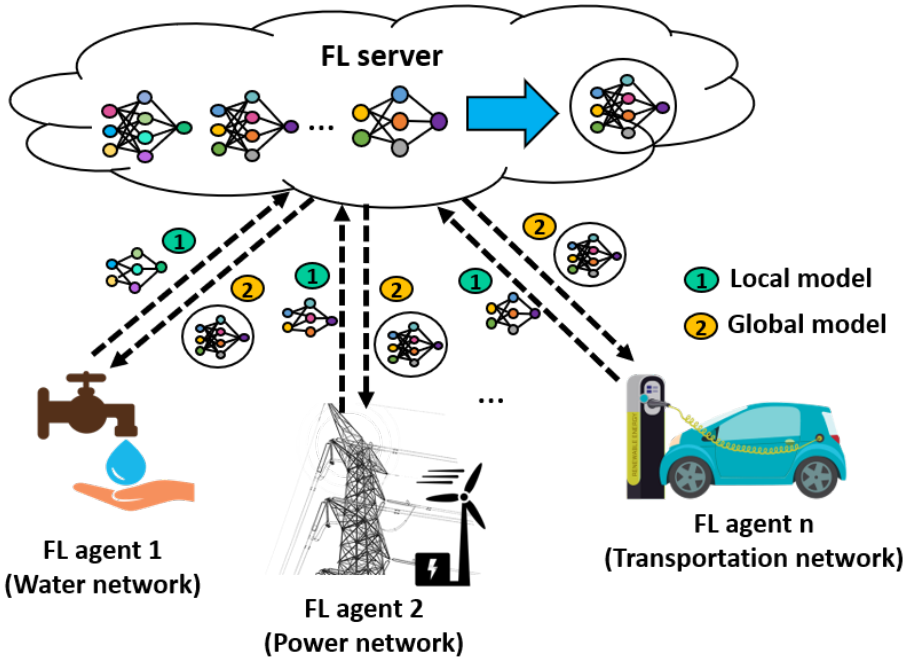


Figure 7.1: FL process considering critical infrastructure agents (CIAs).

on resource-constrained agents. Due to the varying and limited resource statuses, while one agent could perform a given computational task efficiently, another might turn into a straggler. An agent may become a straggler if the assigned computational task is overwhelming compared to its available resources. If the majority of the participating agents in the FL process turn into stragglers, then target convergence may never be obtained. Besides this, the IoT-enabled infrastructure agents are generally more prone to attacks that may cause divergent local model updates [IA20]. To improve the resilience, it is crucial to predict the infrastructure's outage, and in distributed systems, the main hindrance in the agent's learning process is stragglers. Therefore, it is essential to monitor the resource status and ensure that all agents are effectively operating by avoiding straggler issues. The typical FedAvg algorithm [MMR+17] assumes that all FL agents are resource-proficient and capable of accomplishing any given computational tasks. However, if a real-world FL-based

IoT scenario is considered, then the majority of the agents may possess very few resources. Therefore, it is not effective to randomly select a fraction of agents for the training process. If the agents' resource availability statuses are tracked and the weak agents are filtered out from the training process, then it may possible to move one step closer towards resolving straggler issues. To infuse resource-awareness functionalities into the FL process, the task publisher (i.e., FL server) needs to acknowledge each network agent's minimum requirements for accomplishing a published task. After that, all interested agents share their resource information (e.g., memory, processing ability, bandwidth, and battery-life) with the task publisher. By examining the interested agents' resources, the task publisher prepares a list of proficient agents and randomly selects a subset of agents for that task.

Handling Systems and Statistical Heterogeneity of Critical Infrastructure Agents (CIAs)

In this segment, we discuss how a strategy of allowing partial works from the FL agents can be adopted through a generalization of the FedAvg algorithm [MMR+17]. In Section 7.1.2, we explain how the comparatively proficient agents can be selected for an FL process. However, it is possible that, among the selected agents, some agents would not be able to accomplish their entire task. Particularly, this can occur when all the interested and available FL agents have constrained resources and there are no other options without considering a subset of those agents for the training phase. Now, if the conventional FedAvg algorithm is applied [MMR+17], which instructs the center to assign uniform local computational tasks to all the selected agents, then straggler effects may be observed that can slow down the model convergence, or we may never be able to reach the target convergence. Instead, if we allow the selected agents to perform computational tasks based on their resources,

then we would not require the straggler agents to be dropped, and every agent could contribute towards constructing a global model. In Figure 7.2, the high-level view of allowing partial works from the FL agents is presented. From the figure, it can be observed that the water network agent and transportation agent have limited resources while the power network has sufficient resource availability. Considering the resource status, the water network and transportation agents are performing 30% and 48%, respectively, of the overall computational tasks, while the power network is performing the entire task. Let us assume that the task publisher defines a local epoch of 100 that needs to be performed by all chosen FL agents. However, some of the agents are not capable of performing 100 local epochs on their data to generate a local model. In such a case, if an agent is capable of performing 30% of the overall computational task (i.e., 30 local epochs on their own data) rather than the whole task, then the agent would be allowed to perform that amount of the computational task and send back the model to the server. This proposed strategy solves two issues: first, the FL server does not need to wait a long time for a straggler agent, and second, every individual contribution from the agents can be counted. To reduce communication overheads, a popular strategy in federated optimization is that for each iteration period, each agent tries to achieve a local objective function that is used as a replacement of a global objective function. In each training round, a subset of agents is chosen, and each agent uses its resources to optimize the local objective function. After that, the agents share their model with the FL server, which performs aggregation and updates the global model. Allowing a flexible amount of work helps to solve the inexact nature of local objectives and assists in tuning the number of communications vs. local computations. While too many local epochs can overfit the model, a smaller number of local epochs increases communication overheads and the convergence time [IA20].

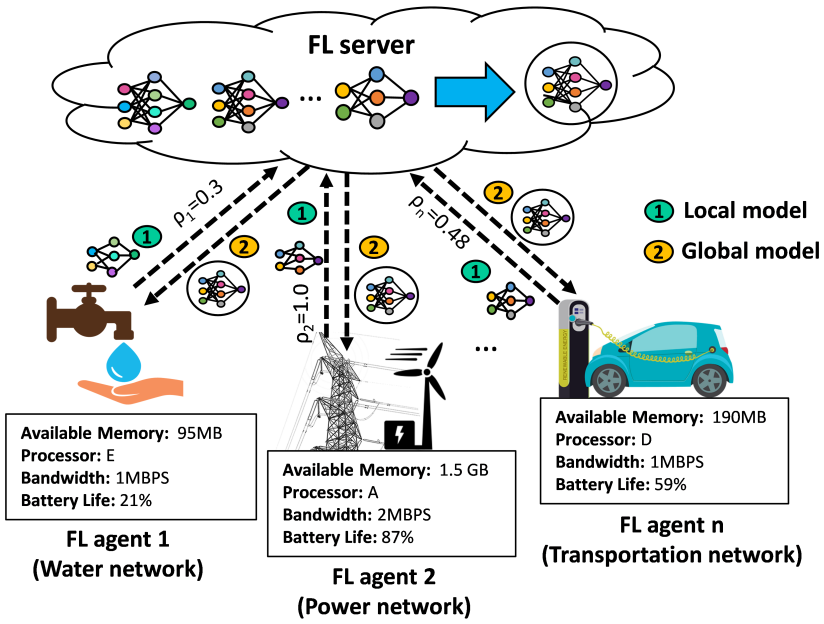


Figure 7.2: Allowing partial amounts of work from the Federated Learning agents.

Therefore, it is required to set local epochs through proper tuning to ensure robust convergence. We can consider a scenario from our real-life perspectives. Suppose we have a few power agents that agree to participate in an FL process and utilize their edge resources. Each agent may have some outage information about some past events and also can possess resource information about its neighboring agents. Now, if an agent wants to gather knowledge about outage events that were never seen by that agent and store resource information from the agents that are not its neighbors, then it needs to adopt a method to obtain the collective knowledge of the whole network. We can infuse the collective knowledge to each agent through the power of FL. In case, if a power agent does not have sufficient resources to complete an assigned computational task, we allow that agent to perform partial works. In this way, we do not ignore any agent's local knowledge. As a consequence, each agent is more capable of predicting an outage event and can locate an agent that needs a power supply.

Proposed FedResilience Algorithm

The proposed FedResilience algorithm is presented in Algorithm **10**. The goal of this algorithm is to predict the outages and resource-sharing scope of CIAs without sharing any agent's local data, utilizing the computational resources of the CIAs. Applying the FL strategy for critical infrastructures mainly involves two entities: the critical infrastructure server (CIS) and available CIAs within the networks. At the beginning of the FL process, the server initializes a global model that is disseminated to all available CIAs within the networks specifying task requirements (line **1**–**2**). Each interested CIA shares its current resource status with the CIS (line **3**). In each training round, the CIS examines the resource information (i.e., processing power, memory, bandwidth, battery-charge status, and data volume) of the interested CIAs by calling the **CheckResource()** function (line **4**–**5**). The **CheckResource()** function receives a CIA's information upon calling, stores the information in a list, and compares the resource availability status with the task requirements (line **13**–**15**). If the CIA's available resources satisfy the minimum task requirements, then that CIA's information is stored in another list and sent back from where the **CheckResource()** function is called (line **16**–**18**). Upon receiving the resource information from all the interested CIAs, the CIS sorts the eligible CIAs based on their resource status, selects a fraction from those CIAs, and randomly chooses a subset of proficient CIAs for the training phase (line **6**–**8**). After that, the CIS calls the selected agents to perform on-device training using the **AgentLocalUpdate()** function and shares the latest global model (line **9**–**10**). It is assumed that the total number of data samples within the network is n, which are distributed among the CIAs with a set of indexes $\mathcal{D}_a$ on CIA a, where $\mathcal{N}_a = |\mathcal{D}_a|$. Each CIA's local data in a communication round t are referred to by $\mathcal{N}_t$. During FL training, each selected CIA utilizes its local solver to determine the inexact minimizer ϱ_a^t to

solve the local objective function (line **19**–**20**). Further, each CIA splits its local samples into batches, performs SGD to achieve an optimal local solution, and shares the model with the CIS (line **21**–**25**). The CIS aggregates the local models to generate an updated global model, and the same iteration period is continued until the global model reaches convergence (line **11**–**12**).

Algorithm 10: FedResilience: An FL-based approach to predict the outages and resource-sharing scopes of critical infrastructures.
The $\mathcal{A}$ eligible clients are indexed by a; B = local minibatch size, $\mathcal{F}$ = client fraction, E = local epoch, and η = learning rate.

1 **CIS executes:** initialize global model w_0
2 Disseminate task requirements to all CIAs
3 Collect resource status of interested CIAs
4 **for** *each round* $t = 1, 2, \ldots$ **do**
5 $\mathcal{R}_t =$ **CheckResource** $(\mathcal{P}_t, \mathcal{M}_t, \mathcal{B}_t, \mathcal{C}_t, \mathcal{V}_t)$ for all interested CIAs
6 Sort CIAs based on $\mathcal{R}$ and store in a list $\mathcal{L}$
7 $\mathcal{E} \leftarrow$ Top $\mathcal{L} \cdot \mathcal{F}$ CIAs
8 $\mathcal{A}_t \leftarrow$ (random set of $\mathcal{E}$ CIAs)
9 **for** *each client* $a \in \mathcal{A}_t$ *in parallel* **do**
10 $w_{t+1}^a \leftarrow$ **AgentLocalUpdate** (a, w_t)
11 **for** *each CIA* $a \in \mathcal{A}_t$ **do**
12 $w_{t+1} \leftarrow w_{t+1} + \frac{\mathcal{N}_t}{\mathcal{N}} w_{t+1}^a$
13 **CheckResource** $(\mathcal{P}_a, \mathcal{M}_a, \mathcal{B}_a, \mathcal{C}_a, \mathcal{V}_a)$**:**
14 Store $(\mathcal{B}_a, \mathcal{M}_a, \mathcal{E}_a, \mathcal{V}_a)$ into a list $\mathcal{Q}_a$
15 Compare $\mathcal{Q}_a$ with $\mathcal{L}_{Req}$
16 **if** $\mathcal{Q}_a$ *satisfies* L_{Req} **then**
17 Add $\mathcal{Q}_a$ to list $\mathcal{R}$
18 **Return** $\mathcal{R}$
19 **AgentLocalUpdate** (a, w) : // Run on CIA a
20 Each CIA a finds a w_a^{t+1} which is a ϱ_a^t -inexact minimizer of: $w_a^{t+1} = F_a(w) + \frac{\xi}{2} \|w - w^t\|^2$ and measures maximum feasible round of local epochs E
21 $\mathcal{B} \leftarrow$ (split $\mathcal{D}_a$ into batch size B)
22 **for** *each CIA's local epoch e from 1 to* E **do**
23 **for** *batch* $b \in B$ **do**
24 $w \leftarrow w - \eta \nabla \ell(w; b)$
25 return w to server

7.1.3 Experimental Results

To evaluate the performance of the proposed FedResilience method, various distributed mobile robots are considered as critical infrastructure agents that possess heterogeneous resources in terms of processing power, battery life, memory, and data volume. To simulate the straggler effects and the effectiveness of the proposed FedResilience algorithm, the Electro-Maps dataset [HMGS20] is used to predict the power outages and resource-sharing scopes of the agents. The dataset is preprocessed considering temperature, number of weeks, hour, holiday, and population, and an additional column of resource availability is generated from the information regarding the population, holiday, and temperature. Using the information, we target the prediction of the outages and resource-sharing scopes of the critical infrastructure agents. A similar transmission rate is set for all the distributed agents for the simplicity of the FL implementation process. To simulate the effectiveness of allowing partial works from the distributed agents, different numbers of weak distributed agents are deliberately considered to create straggler effects; i.e., some of the agents fail to generate local models due to their constrained resources. It is assumed that there remains a global cycle that is followed by each agent, and each selected agent measures the amount of the local computational task it can perform in training round i as a function of its available resources and clock cycle.

In a conventional FL approach, a global epoch E is defined for all the participating agents to perform a particular task, and if any of the agents fail to generate a local model on time, the model simply drops that agent from the training process (no partial tasks are allowed). However, dropping slow clients from the training process may prolong the model convergence, or the model may even never reach the target convergence. To handle such issues, we adapt a generalization of the FedAvg algorithm that enables each agent to perform part of a computational task

by considering the agent's resource limitations. To present the motivation behind this research, we applied the FedAvg algorithm [MMR+17] for predicting the outages and resource-sharing scopes of CIAs and presented the straggler effects. We considered a varying number of CIAs and assumed that a majority of those agents would be stragglers. At first, we considered three agents (where two were stragglers) and computed the training loss and testing accuracy during the prediction of a probable outage by applying the state-of-the-art FedAvg [MMR+17] algorithm. In Figure 7.3, we can see that the training loss started to decrease in the initial few communication rounds and remained almost unchanged for further communication rounds due to the dropping of the majority of clients. In contrast, in Figure 7.4, it is clear that the improvement of testing accuracy was quite steady and each agent learned very slowly.

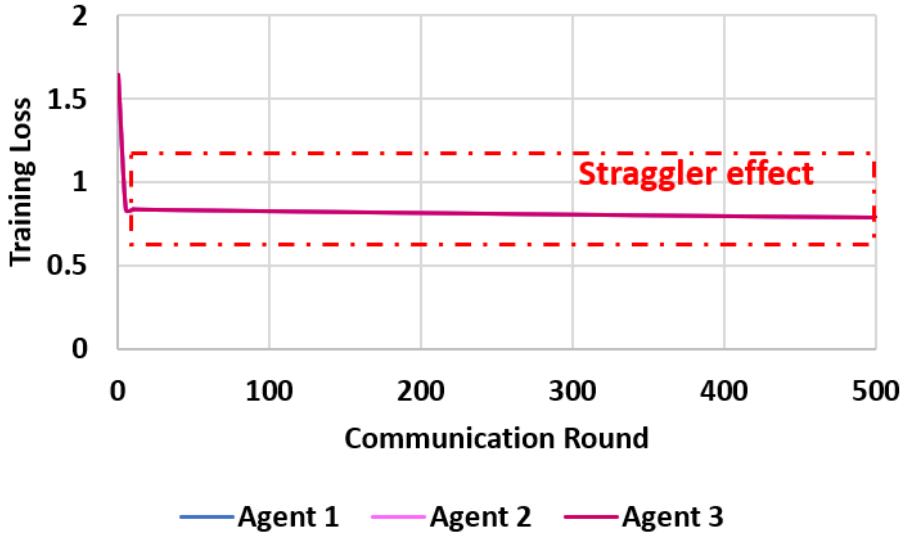


Figure 7.3: Straggler effects on participating Federated Learning agents' (three agents and two stragglers) model loss for the prediction of critical infrastructure outage.

After that, we simulated the straggler effects by increasing the number of agents (three stragglers out of five agents) and computing the training loss to predict a

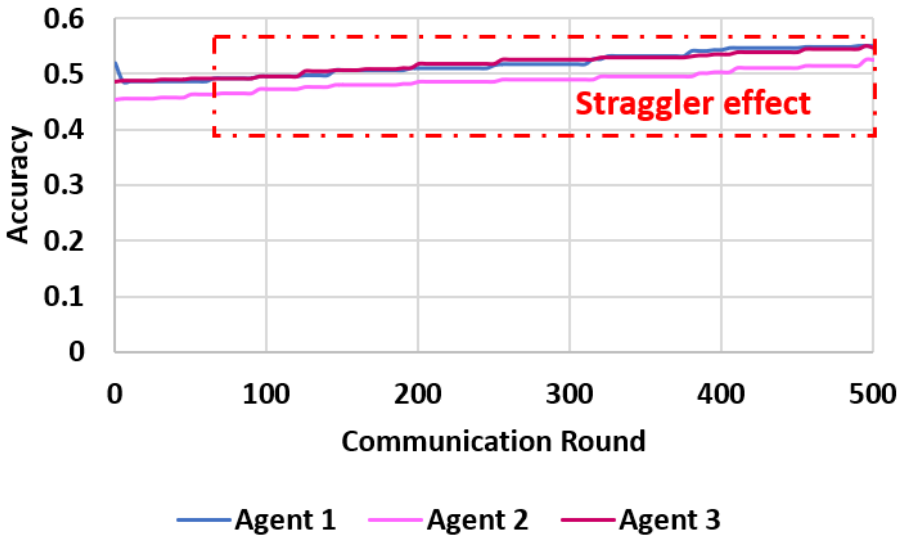


Figure 7.4: Straggler effects on participating FL agents' (three agents and two stragglers) model accuracy for the prediction of critical infrastructure outage.

probable outage by applying the state-of-the-art FedAvg [MMR+17] algorithm (see Figure 7.5). We can see a small decrease in training loss for communication round 500. In contrast, in Figure 7.6, it is observable that some agents had very low accuracy while other agents had comparatively high accuracy. However, none of the agents achieved satisfactory improvements in their accuracy.

We also simulated the straggler effects for eight agents (where six-of them were stragglers) and generated the training loss for a predicted outage by applying the state-of-the-art FedAvg [MMR+17] algorithm (see Figure 7.7). We can see that both of the non-straggler agents had a very slow learning process in spite of a higher communication round.

In Figure 7.8, we can see that the clients barely learned from each other and consequently were not able to improve their model quality significantly. Next, we simulated the straggler effects during the prediction of the agents' resource-sharing scope by applying the state-of-the-art FedAvg [MMR+17] algorithm. Similar to the

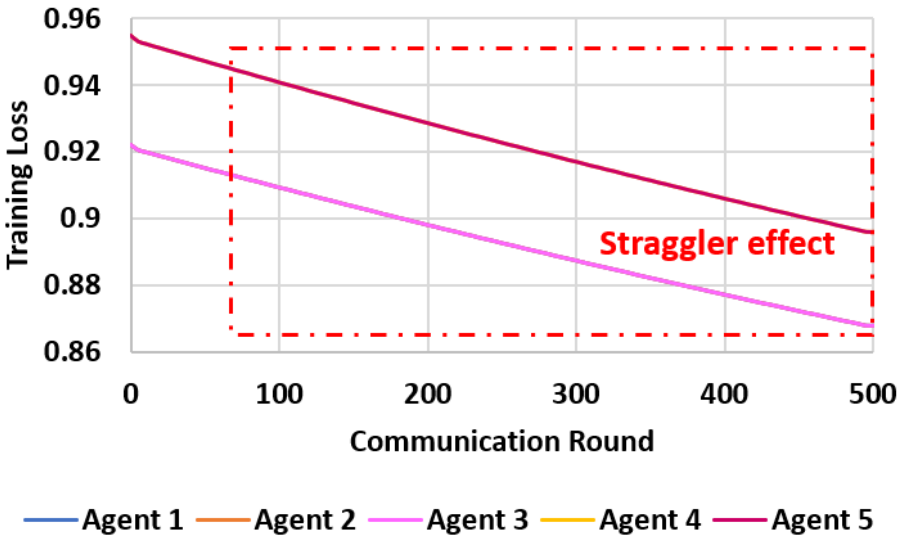


Figure 7.5: Straggler effects on participating Federated Learning agents' (five agents and three stragglers) model loss for the prediction of critical infrastructure outage.

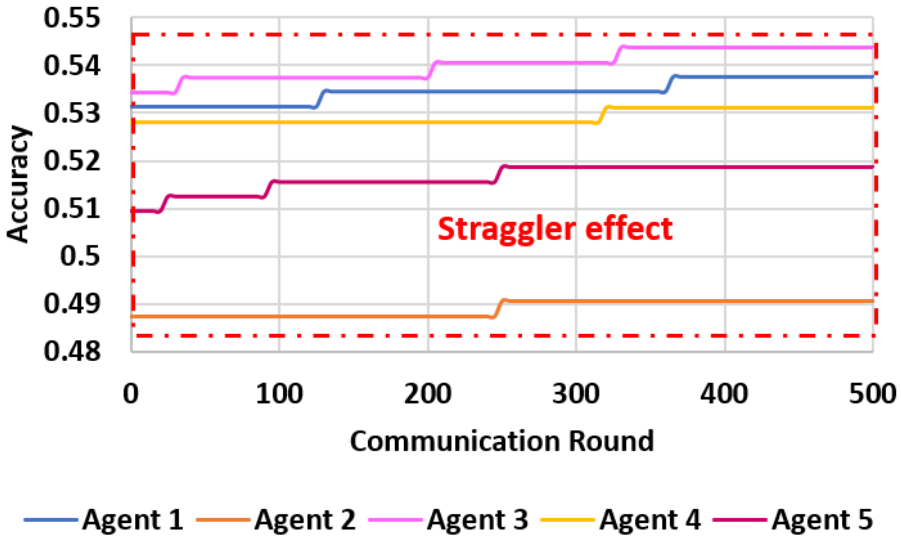


Figure 7.6: Straggler effects on participating FL agents' (five agents and three stragglers) model accuracy for the prediction of critical infrastructure outage.

outage prediction, we considered three agents (where two were stragglers) and generated the training loss and testing accuracy to predict the resource-sharing scope of the agents. In Figure 7.9, we can see that though the training loss was comparatively

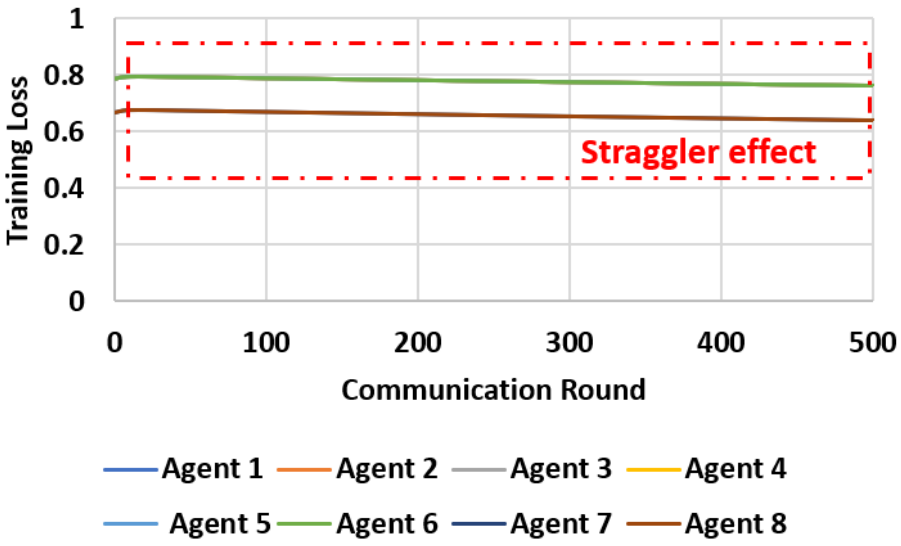


Figure 7.7: Straggler effects on participating Federated Learning agents' (eight agents and six stragglers) model loss for the prediction of critical infrastructure outage.

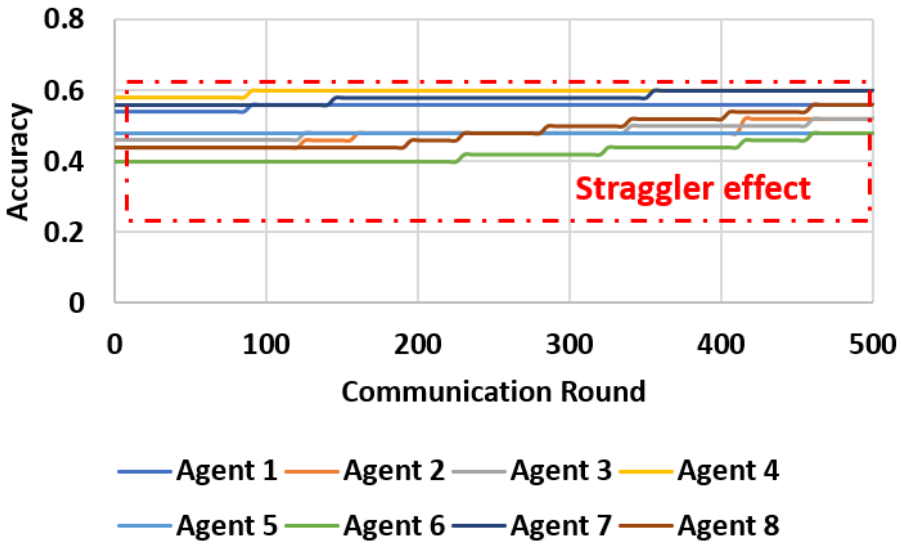


Figure 7.8: Straggler effects on participating FL agents' (eight agents and six stragglers) model accuracy for the prediction of critical infrastructure outage.

lower than the outage prediction loss for the three agents–two stragglers scenario, a similar training loss was observed for all agents (i.e., the agents did not learn through collaboration). On the other hand, we can see from Figure 7.10, that agent

2 had a comparatively lower accuracy than other agents, but it slightly improved its accuracy through the FL process. However, after 200 communication rounds, all agents' testing accuracies improved very slowly.

Besides, we simulated the straggler effects during the prediction of the agents' resource-sharing scope by increasing the number of agents (five agents, where three of them were stragglers). We generated the training loss and testing accuracy by applying the state-of-the-art FedAvg [MMR+17] algorithm. In Figure 7.11, we can see that though the training loss dropped significantly in the few initial communication rounds, almost constant training loss was observed for all agents. Moreover, in Figure 7.12, all the agents failed to obtain a marginal improvement in their accuracy.

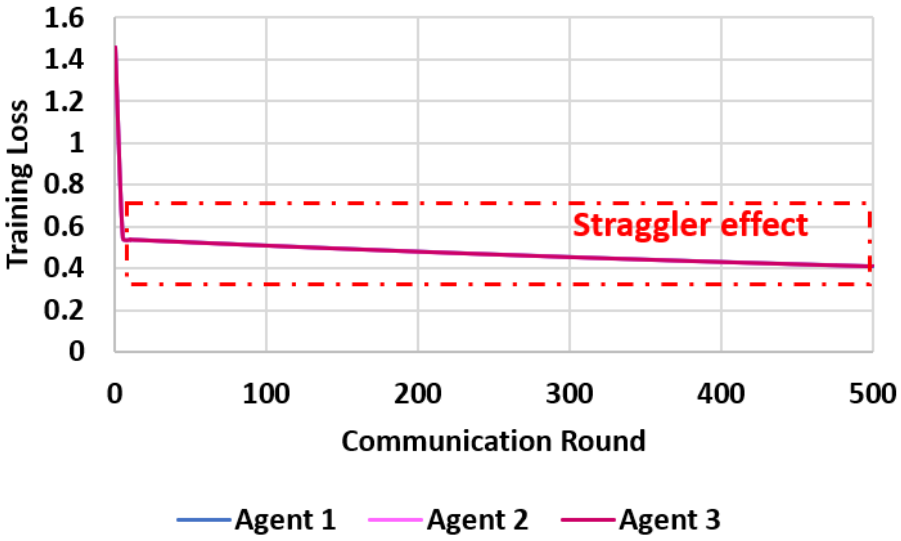


Figure 7.9: Straggler effects on participating Federated Learning agents' (three agents and two stragglers) model loss for the prediction of agents' resource-sharing capability.

Similarly, we simulated the training loss and accuracy by considering eight agents (where six were stragglers) and applied the FedAvg algorithm [MMR+17] during the prediction of the agents' resource-sharing scope. From Figures 7.13 and 7.14, we

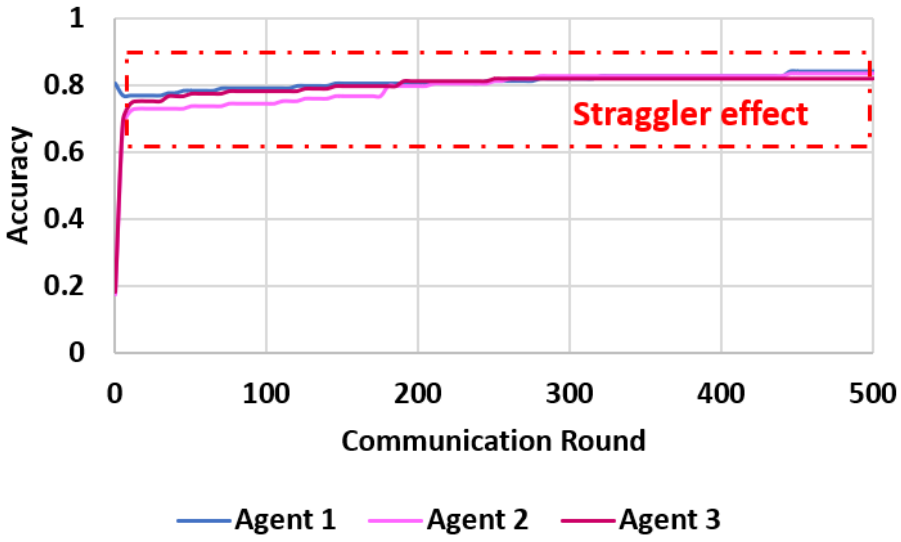


Figure 7.10: Straggler effects on participating FL agents' (three agents and two stragglers) model accuracy for the prediction of agents' resource-sharing capability.

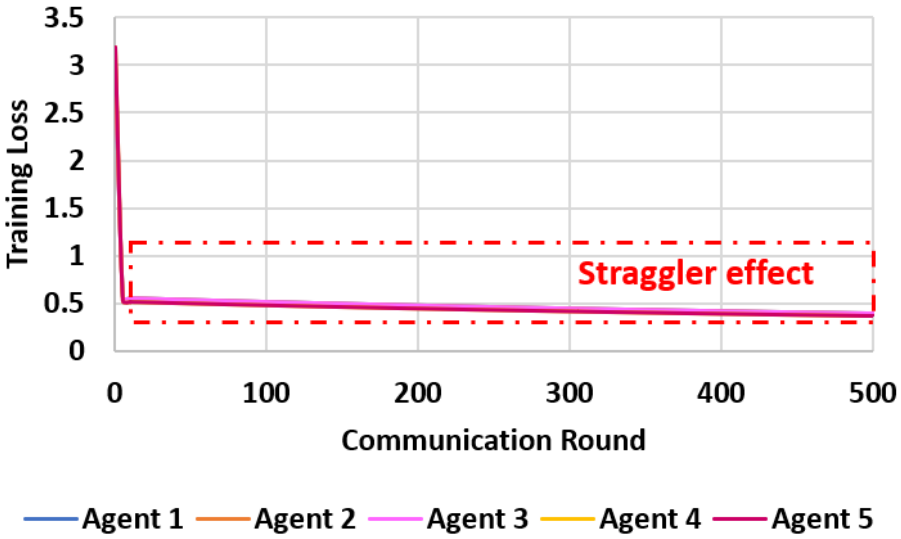


Figure 7.11: Straggler effects on participating Federated Learning agents' (five agents and three stragglers) model loss for the prediction of agents' resource-sharing capability.

can see that the training loss and accuracy improved as we increased the number of agents; however, both accuracy improved little with the increment of communication rounds due to the straggler effects. In a summary, for all the considered cases,

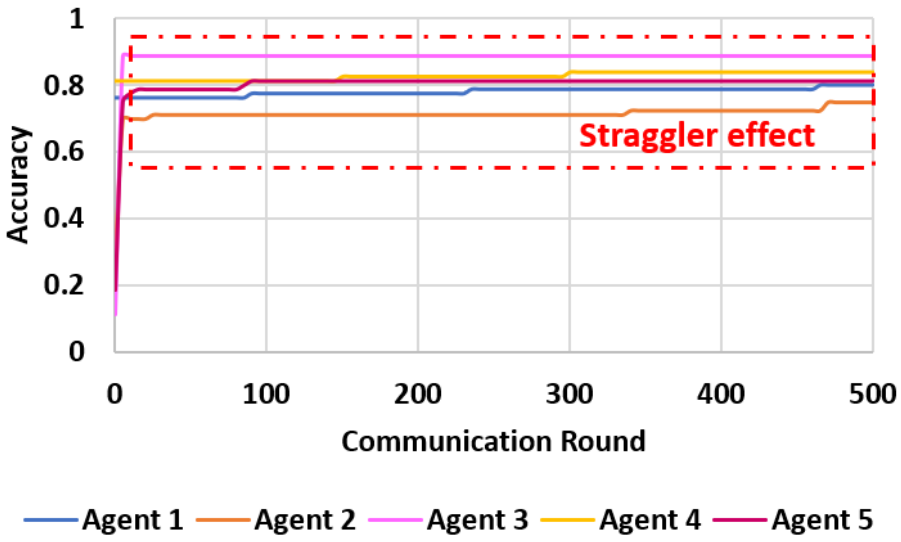


Figure 7.12: Straggler effects on participating Federated Learning agents' (three agents and two stragglers) model accuracy for the prediction of agents' resource-sharing capability.

the agents struggled to minimize loss and remained very steady in terms of improving accuracy due to the straggler effect.

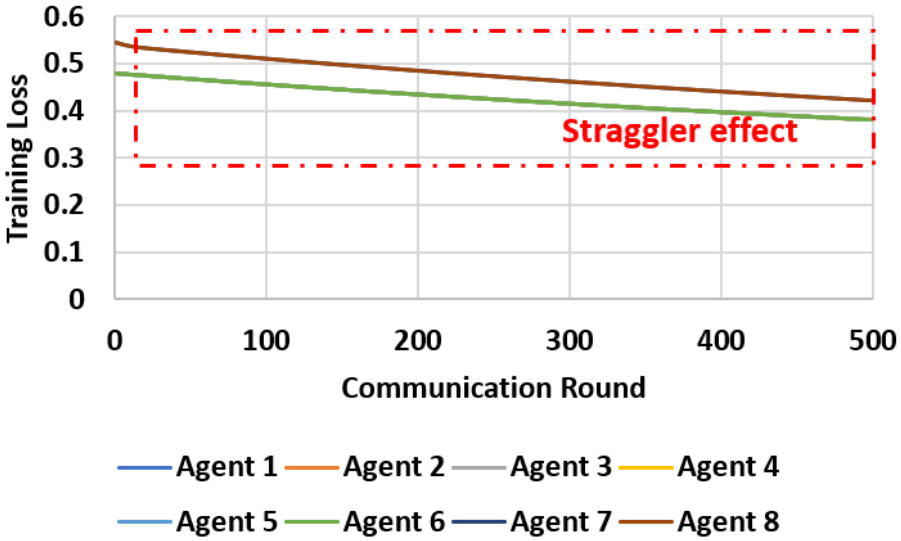


Figure 7.13: Straggler effects on participating Federated Learning agents' (eight agents and six stragglers) model loss for the prediction of agents' resource-sharing capability.

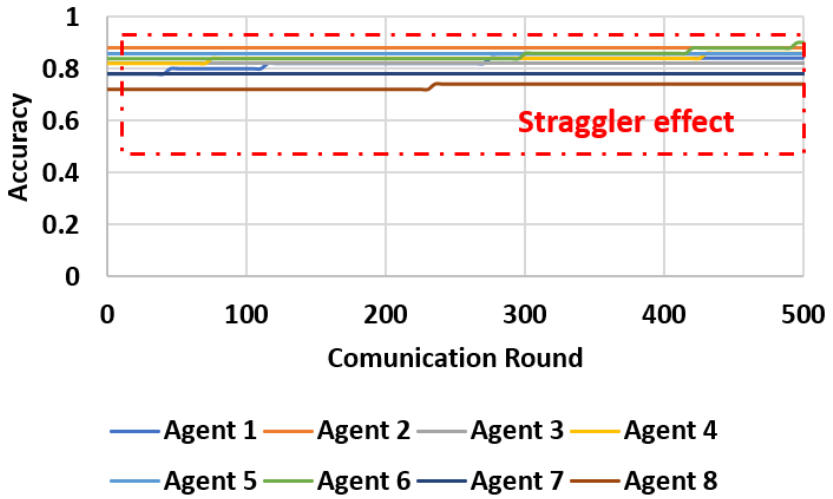


Figure 7.14: Straggler effects on participating FL agents' (eight agents and six stragglers) model accuracy in prediction of agents' resource-sharing capability.

To eliminate the straggler effects during the prediction of the power outage and resource-sharing information, we leveraged allowing of partial works from the straggler agents; i.e., we assigned computational tasks based on the agents' available resources. To evaluate the performance, we considered the same number of agents (three, five, and eight agents) during training and observed their learning process. At first, we considered three agents (where two agents were stragglers) and checked their loss (Figure 7.15) and accuracy (Figure 7.16) during the prediction of power outages. Though the training loss increased due to the deviation of the local model updates as the stragglers performed low computational tasks, the agents started to reduce their training loss by learning from the global model and from their own data. On the contrary, the accuracy of the agents started to increase after 320 communication rounds because of the low number of resource-sufficient agents (Figure 7.16). We also simulated the loss (Figure 7.17) and accuracy (Figure 7.18) during the prediction of resource-sharing scope by considering the same number of agents and achieved better performance than the FedAvg [MMR+17] algorithm.

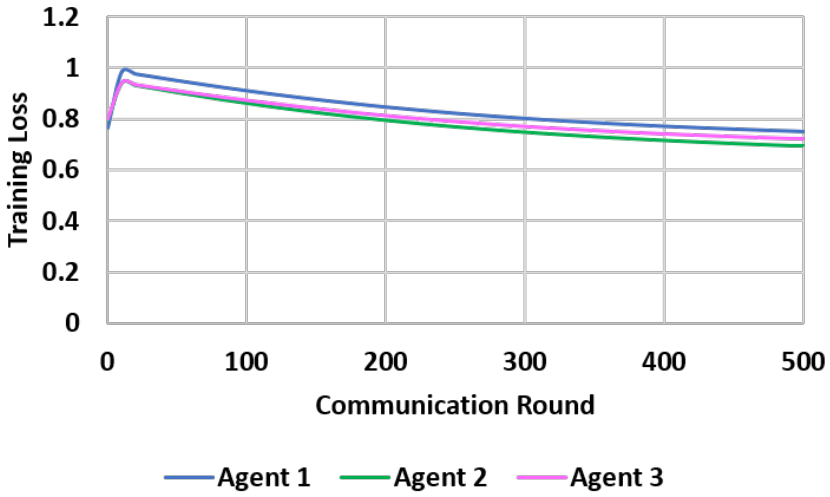


Figure 7.15: FedResilience's impact on participating Federated Learning agents' (three agents and two stragglers) model loss during prediction of outages.

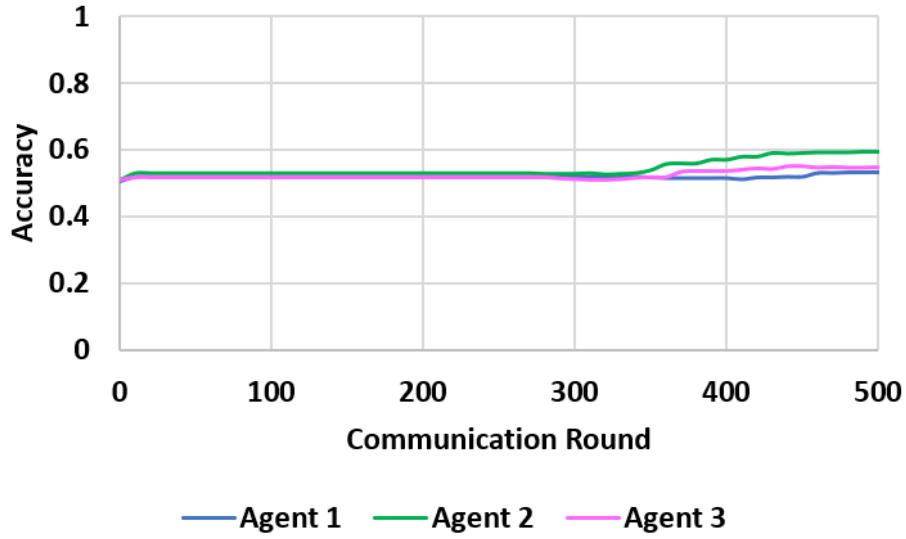


Figure 7.16: FedResilience's impact on participating Federated Learning agents' (three agents and two stragglers) model accuracy during prediction of outages.

After that, we considered five agents (where two agents were stragglers) and checked their loss (Figure 7.19) and accuracy (Figure 7.20) during the prediction of power outages. We observed similar patterns to those in Figures 7.15 and 7.16, but obtained better performance due to the higher number of active clients. The ac-

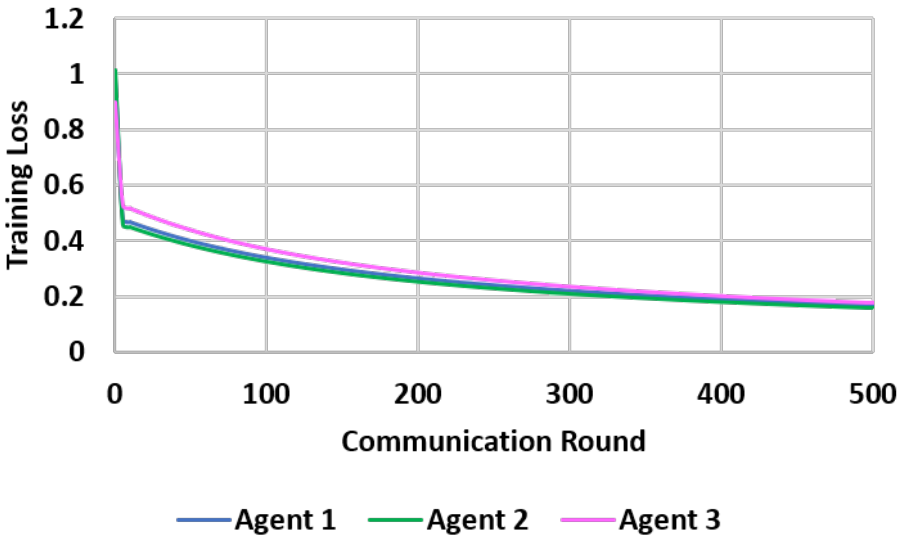


Figure 7.17: FedResilience's impact on participating Federated Learning agents' (three agents and two stragglers) model loss during prediction of resource-sharing capability.

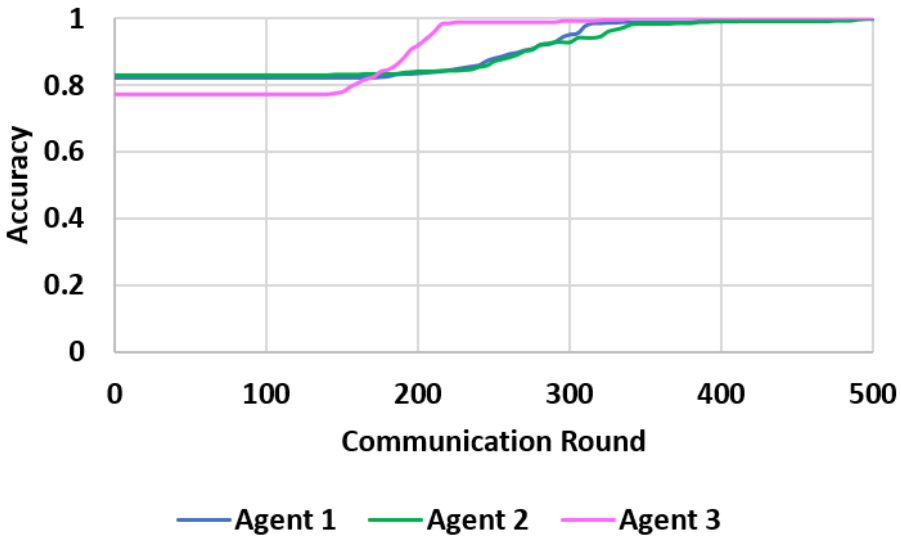


Figure 7.18: FedResilience's impact on participating Federated Learning agents' (three agents and two stragglers) model accuracy during prediction of resource-sharing capability.

curacy of the agents started to increase after 100 communication rounds because of the comparatively higher number of resource-sufficient agents (Figure 7.20).

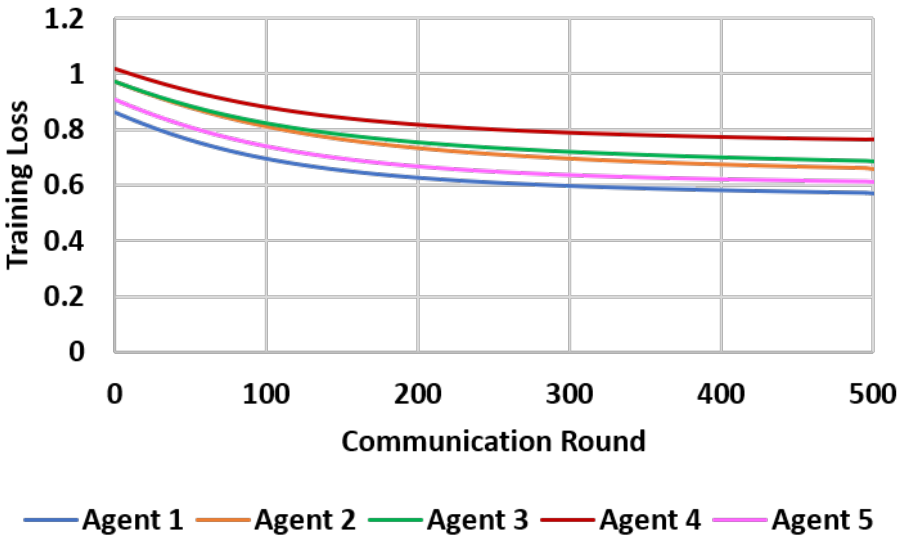


Figure 7.19: FedResilience's impact on participating Federated Learning agents' (five agents and three stragglers) model loss during prediction of outages.

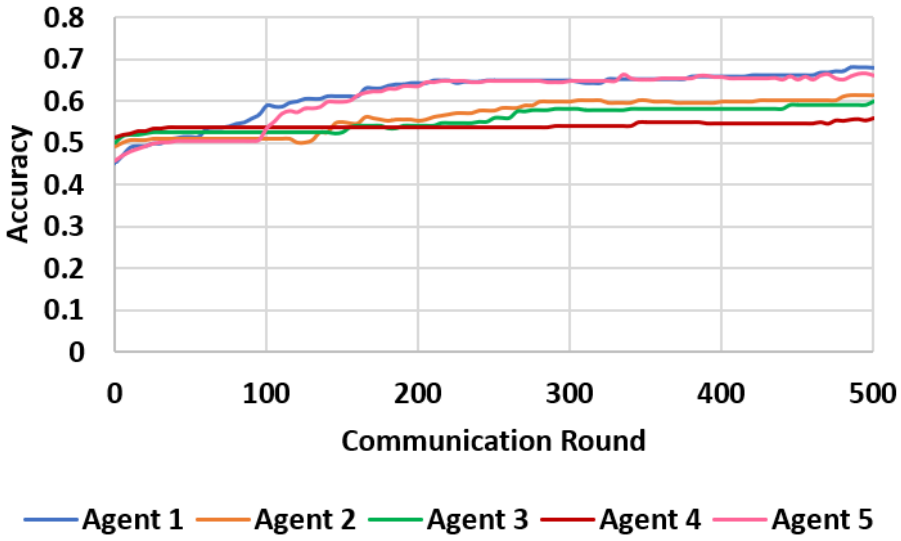


Figure 7.20: FedResilience impact on participating FL agents' (five agents and three stragglers) model accuracy during prediction of outages.

We also simulated the loss (Figure 7.21) and accuracy (Figure 7.22) during the prediction of resource-sharing scope by considering five agents and achieved better performance than the FedAvg [MMR+17] algorithm.

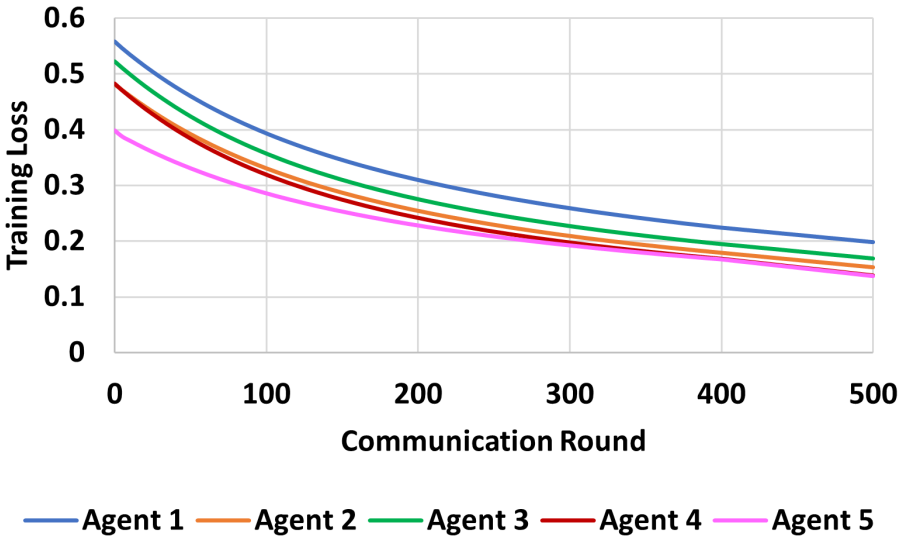


Figure 7.21: FedResilience's impact on participating Federated Learning agents' (five agents and three stragglers) model loss during prediction of resource-sharing capability.

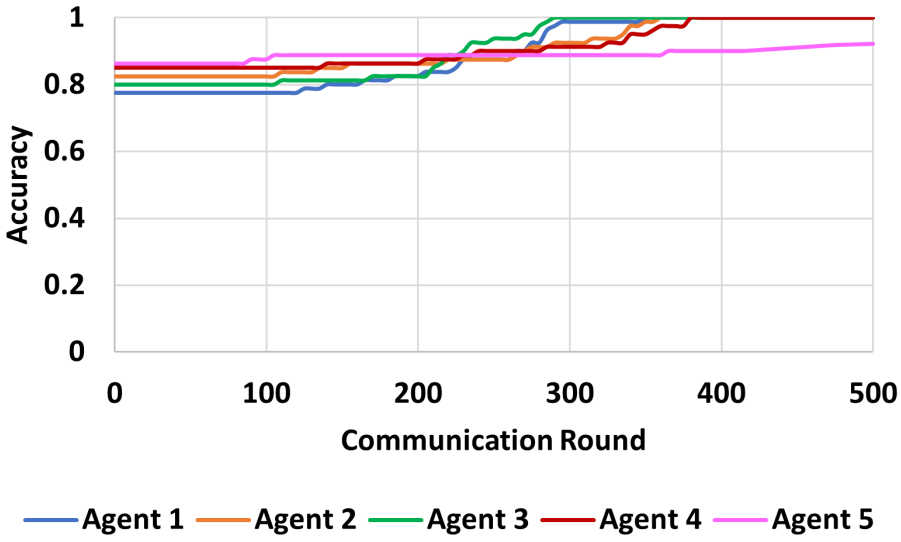


Figure 7.22: FedResilience's impact on participating Federated Learning agents' (five agents and three stragglers) model accuracy during prediction of resource-sharing capability.

Further, we simulated the performance of eight agents (where six agents were stragglers) and checked their loss (Figure 7.23) and accuracy (Figure 7.24) when

predicting power outages. Here, it was clear that the agents improved their knowledge base (i.e., the training loss decreased and a significant accuracy improvement is observed) due to the improved quality of the global model. In a similar fashion, we

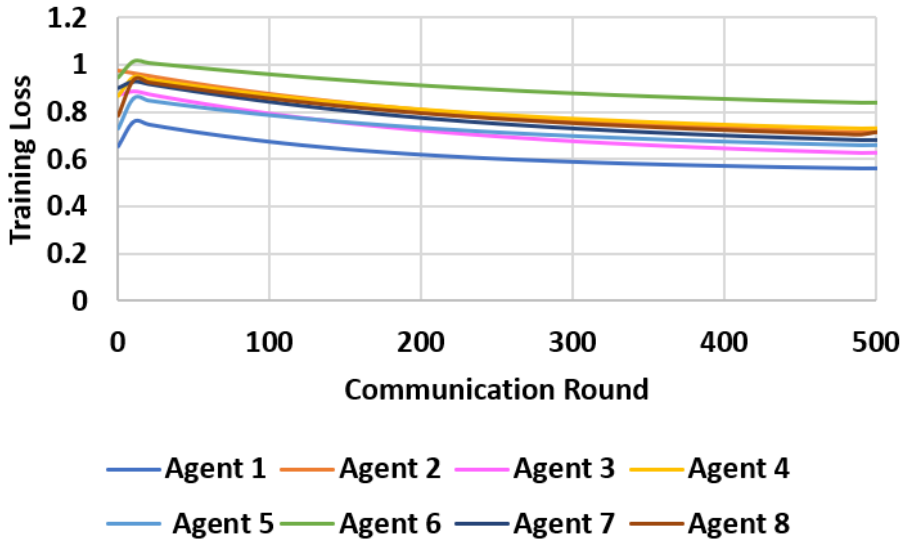


Figure 7.23: FedResilience's impact on participating Federated Learning agents' (eight agents and six stragglers) model loss during prediction of outages.

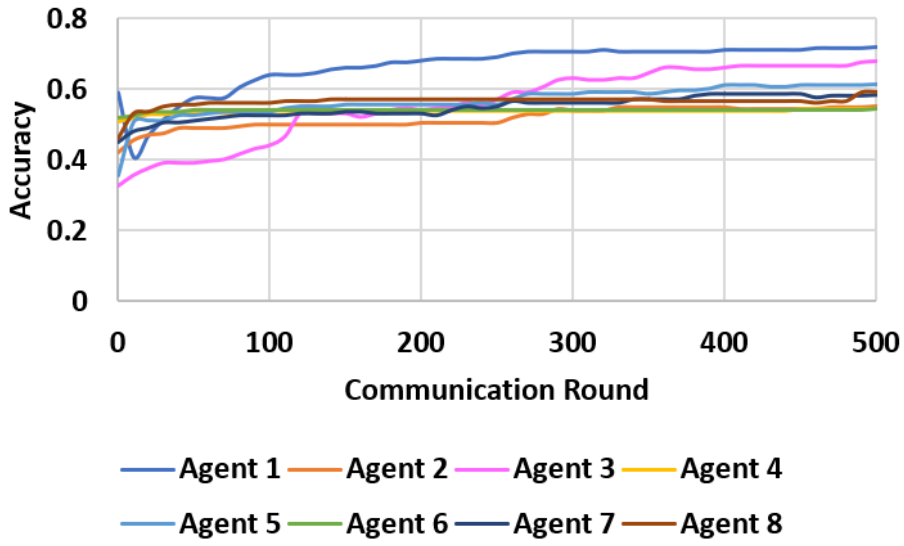


Figure 7.24: FedResilience's impact on participating Federated Learning agents' (eight agents and six stragglers) model accuracy during prediction of outages.

simulated the loss (Figure 7.25) and accuracy (Figure 7.26) during the prediction of resource-sharing scope by considering eight agents and achieved a remarkable performance improvement compared to the FedAvg [MMR+17] algorithm. The training loss became close to 0.1 (Figure 7.25), and some of the agents achieved higher accuracy within $250 - 300$ communication rounds (Figure 7.26).

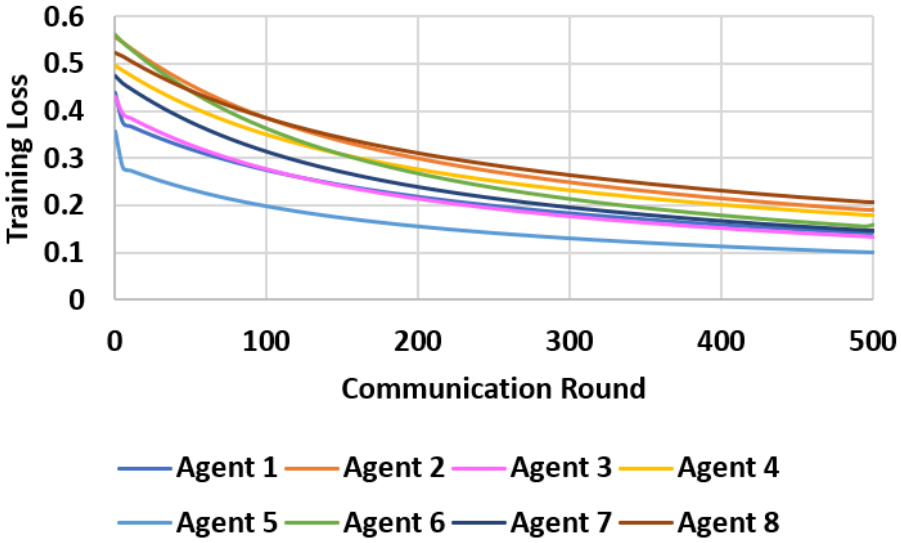


Figure 7.25: FedResilience's impact on participating Federated Learning agents' (eight agents and six stragglers) model loss during the prediction of resource-sharing capability.

From the simulation results, it is observable that FedResilience has better performance than the conventional FedAvg model. As we increase the number of agents and count partial works from each of them, then agents can learn quickly, and the global model accuracy also increases. When we considered eight agents and six stragglers and applied the FedAvg algorithm, then the global model only contained the knowledge of the two active agents. As a result, the agent could not upgrade its knowledge base and showed a steady learning curve. However, when we counted the partial computational tasks by those stragglers, then the accumulation of those partial works generated an upgraded global model. As the quality of the global model

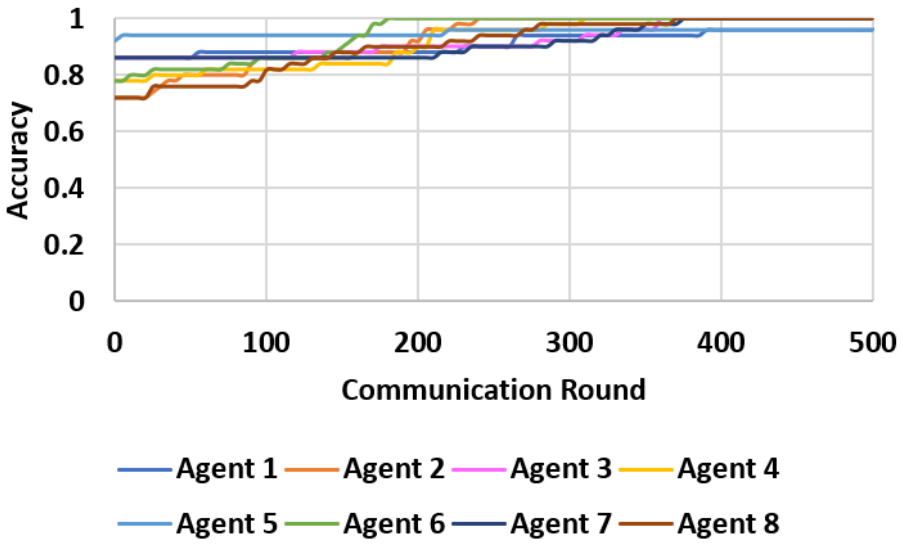


Figure 7.26: FedResilience's impact on participating Federated Learning agents' (eight agents and six stragglers) model accuracy during the prediction of resource-sharing capability.

improved and each agent tuned their local model by learning from the latest global model, the agents' learning process was accelerated. Our simulation results demonstrate two trends: first, the FedAvg algorithm is not suitable to predict outages or the resource-sharing information of resource-constrained CIAs as the algorithm cannot handle the straggler effects, which eventually slows down the agents' learning process; second, the FedResilience algorithm can handle straggler effects and is suitable even when we have a large number of stragglers within the network. In Figure 7.27, we can see that the FedResilience algorithm outperforms the FedAvg algorithm [MMR+17], achieving higher global model accuracy (cumulative updates of all the participating agents' local models) while predicting the outages and resource-sharing information of CIAs even with a large number of stragglers.

In Figure 7.28, a linear approximation of the real system and the performance of the proposed FedResilience algorithm for a disaster event is presented. It can be seen that the real system performance index decreases after time t_d and reaches a minimal

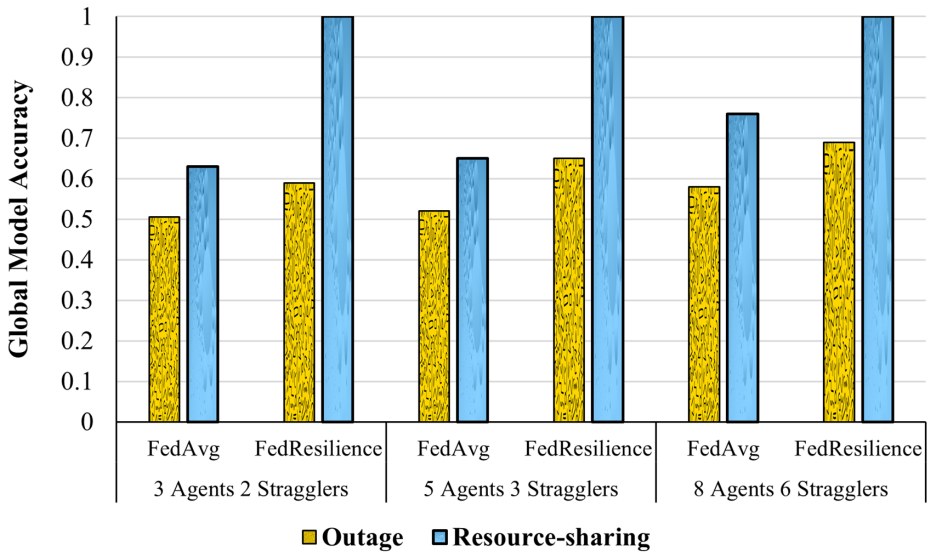


Figure 7.27: Comparison of global model accuracy of FedAvg and proposed FedResilience in presence of stragglers.

index at time t_m. In the beginning, the performance index is stable due to a preventive outage; however, as soon as the preventive outage is finished, the curve starts to move downwards and reaches a minimal performance index ($P_{\min}$). The low-performance index remains until a certain time interval, and after that, the system starts to recover. In contrast, when the FedResilience algorithm is applied during an outage, the performance index does not move down at minimal performance index ($P_{\min}$); instead, using the power of edge intelligence, the system can recover swiftly and a remarkable performance index can be achieved.

7.1.4 Discussion

We proposed a strategy to improve the resilience operations of critical infrastructures even when the network agents have limited resources. To evaluate our approach, the impact of straggler agents on the overall learning process is presented by considering resource-constrained distributed agents. After that, the effectiveness of our

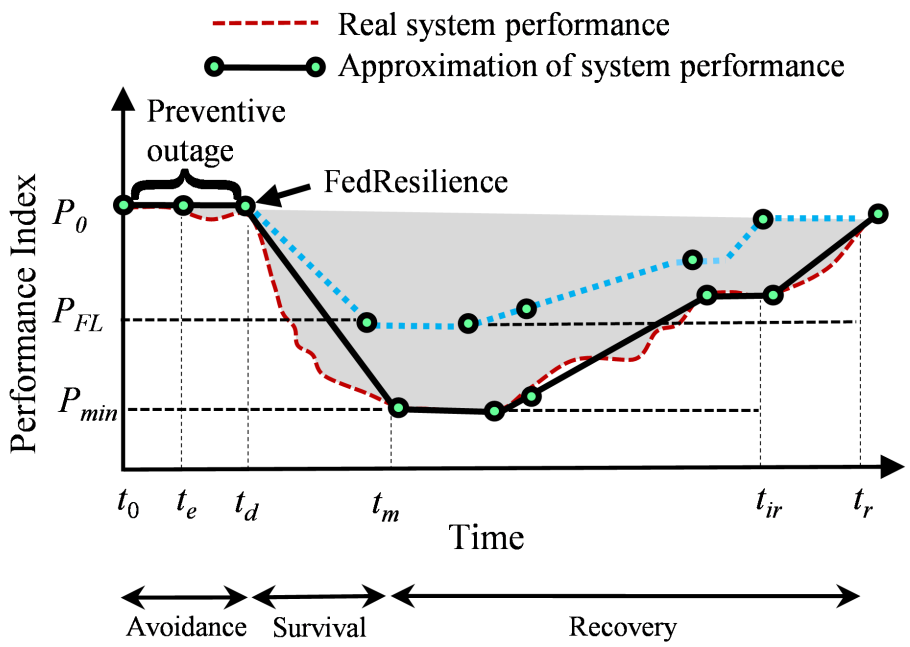


Figure 7.28: Eight-point linear approximation of the performance of the FedResilience algorithm during a disaster event.

proposed FedResilience algorithm is evaluated, demonstrating the acceleration of the distributed agents' learning process despite heterogeneous system resources and model updates. By choosing proficient agents, performing on-device training, transferring knowledge, and allowing partial works, a robust and consistent FL model is achieved with higher global model accuracy compared to the state-of-the-art FedAvg algorithm; the model can also accelerate the learning process of unreliable IoT-enabled heterogeneous environments. The proposed concept can be applied to any resource-constrained heterogeneous IoT environment that is disrupted by straggler effects and struggles to reach convergence due to slow learning.

7.2 Exploiting Federated Learning Technique to Recognize Human Activities in Resource-Constrained Environment

7.2.1 Abstract

The conventional machine learning (ML) and deep learning (DL) methods use large amount of data to construct desirable prediction models in a central fusion center for recognizing human activities. However, such model training encounters high communication costs and leads to privacy infringement. To address the issues of high communication overhead and privacy leakage, we employed a widely popular distributed ML technique called *Federated Learning (FL)* that generates a global model for predicting human activities by combining participated agents' local knowledge. The state-of-the-art FL model fails to maintain acceptable accuracy when there is a large number of unreliable agents who can infuse false model, or, resource-constrained agents that fails to perform an assigned computational task within a given time window. We developed an FL model for predicting human activities by monitoring agent's contributions towards model convergence and avoiding the unreliable and resource-constrained agents from training. We assign a score to each client when it joins in a network and the score is updated based on the agent's activities during training. We consider three mobile robots as FL clients that are heterogeneous in terms of their resources such as processing capability, memory, bandwidth, battery-life and data volume. We consider heterogeneous mobile robots for understanding the effects of real-world FL setting in presence of resource-constrained agents. We consider an agent unreliable if it repeatedly gives slow response or infuses incorrect models during training. By disregarding the unreliable and weak

agents, we carry-out the local training of the FL process on selected agents. If somehow, a weak agent is selected and started showing straggler issues, we leverage asynchronous FL mechanism that aggregate the local models whenever it receives a model update from the agents. Asynchronous FL eliminates the issue of waiting for a long time to receive model updates from the weak agents. To the end, we simulate how we can track the behavior of the agents through a reward-punishment scheme and present the influence of unreliable and resource-constrained agents in the FL process. We found that FL performs slightly worse than centralized models, if there is no unreliable and resource-constrained agent. However, as the number of malicious and straggler clients increases, our proposed model performs more effectively by identifying and avoiding those agents while recognizing human activities as compared to the state-of-the-art FL and ML approaches.

7.2.2 Introduction

Motivation

Human activity recognition (HAR) is one of the widely popular classification machine learning (Ml) applications that is used to identify activities (e.g., sitting, standing, walking, laying, or driving a vehicle) of a person. The HAR classifier model has immense applicability in various other applications such as surveillance system, fitness applications (e.g., counting number of steps), patient monitoring, context aware applications (e.g., automatic response to a phone call while driving) and so on. The prevailing HAR methods requires centralized server to store the collected data in order to constructing a powerful model. In such approaches, a user has no idea whether data would be leaked or modified by other entities or not once they are shared with the central server. Besides, the centralized ML approaches

tends to several other issues. For example, in some cases, it is difficult to share data across various organizations due to privacy or liability concerns. Also, the amount of data collected by a human agent could be large and sending them to a central server would incur huge communication cost [CY+18]. We present a rational conversation between an AI service provider (AI) and a customer (C) regarding data sharing and ML model generation:

C : "*We need a solution for recognizing human activities in a factory to provide early warning of unusual activities.*"

AI : "*No problem. We need to collect some data from your factory to train a human activity recognition model.*"

C : "*Of course. We already have sufficient amount of data which are collected from the smart devices placed in our factory.*"

AI : "*Awesome! Please upload all the data in our server.*"

C : "*I am afraid to say that I can not authorized to share those sensitive data with a third party!*"

AI : "*We can send our ML experts to work on-site with those data but you need to pay additional costs for that.*"

C : "*Sorry, we are not able to bear additional expanses as it would exceed our current budget!*"

This is one of the situations that urges AI community to rethink inventing a new strategy of model training keeping user information private. With the same motivation, a privacy preserving distribute ML technique, Federated Learning (FL) [MMR+17] was invented that enables on-device model training and produces a global model accumulating knowledge of all the clients. FL eliminates the need of uploading dataset to a central server, reduces communication overhead, and minimizes data leakage as model are trained through model aggregation rather than data

sharing. Under the FL framework, we only require to train a HAR model locally on each data owner's (i.e., FL client) side using its own computing resources, and upload the model parameters to the FL server for model aggregation. The existing FL-based HAR applications [ZLL+20], [OXZ+21] assumed that all the FL clients have sufficient resources to perform assigned local computation and generate a local model that would be shared with the server. However, if we consider an FL-based Internet-of-Things (IoT) environment, where the small and resource-constrained IoT devices would be considered as the FL clients, then a lot of such FL clients would be incapable to perform a whole computational task. Besides, as the IoT devices are unreliable and more prone to attacks [ITW+22], it may result in ineffective model update from the clients. Hence, the client selection part is crucial in an FL setting as an unreliable and slow client (also called stragglers) can prolong the learning process. A client turns into a straggler if it is underpowered compared to the task assigned to it. Considering all these, we proposed an FL-based HAR approach by choosing proficient clients and enabling partial works based on the client resources, which is particularly suitable for resource-constrained environment. By leveraging asynchronous FL, we further reduce the straggler effect empowering immediate global model update on the FL server.

Literature Reviews

Human activity recognition (HAR) plays an important role in pervasive and ubiquitous computing. The state-of-the-art deep learning models replaced the traditional feature engineering and achieved higher accuracy [WCH+19], [YHZ+17]. However, all such centralized solutions for HAR rely on collected data from network clients that causes privacy issues. Federated Learning (FL) comes forward with a unique solution of preserving client's privacy and eliminates the solitary dependency on a

central fusion center for model generation. As a consequence, the research domain of FL is inflating due to its unique feature of preserving data privacy, handling non-IID data, and minimizing communication overhead. Several research papers on FL [LSTS20] [LLH+20], [IA19], [YLS+20], [LFTL20] are available focusing on FL system design, components, applications, challenges, and their potential solutions. In turn, the authors in [ITW+22] presented a comprehensive survey on FL-IoT setting, where they pointed out the challenges, analyzed prospective solutions, and highlights several future directions that arise while applying FL on a resource-constrained IoT environment. The authors [IA21] proposed an all-inclusive FL model, FedPARL that can shrink the model size of the resource-constrained agents, selects the most proficient and trustworthy agents for training and dynamically allocate local tasks for the agents. Since the invention of FL method, it has been applied in various applications, such as recommendation system [CLD+18], keyword spotting [LCL+19], next-word prediction [HRM+18], smart robotics [IA20] and so on. Similar to such mobile applications and robotics systems, HAR is another vital aspect that can be benefitted from FL simplifying privacy management and providing user adequate flexibility over controlling their local data by choosing which data would be selected and how the selected data should be contributed in development of a HAR application. The authors in [SVG18] proposed a HAR system powered by FL and they demonstrated that their proposed model performs very close to the centralized model performance that may experience privacy issues. Zhao et al. [ZHS+20] designed an FL-based HAR system for human activity and health monitoring considering low cost edge devices (e.g., Raspberry Pi) and simulate the model inference time until reaching to an acceptable prediction accuracy. Besides, a locally personalized FL-based HAR system is introduced in [FRS+20] to predict human mobility. The above-mentioned proposed works adopted FL method due to the ultimate need

of utility and privacy guarantee. However, the existing FL-based HAR approaches randomly select clients for training without considering their capability to perform training task that arise straggler effects. Further, all the existing FL-based HAR systems consider a uniform computational task to be assigned to all selected FL clients while in real-world scenario, it is infeasible to make sure that all clients would be able to perform equivalently. Moreover, for an FL-based HAR setting, any resource-constrained device can act as a server; hence, it is not viable to send all the human activity data to a central server and develop a HAR classifier from the collected data as like conventional ML approaches. Thus, there is a research gap in FL-based HAR application development considering a resource-constrained IoT environment.

Contributions

Our main contributions of this proposed work are listed below:

- We proposed an FL model for HAR systems that is effective for a resource-constrained environment.
- We proposed a generalization of FedAvg algorithm to develop a HAR classifier that can allocate local computational tasks to federated clients based on their available resources.
- We adapted a mechanism of selecting proficient client during training that can mitigate the straggler effects.
- We infuse asynchronous FL mechanism in HAR systems so that the FL server can perform immediate model update and accelerate the convergence.

7.2.3 System Description

Federated Learning Background

Federated Learning (FL) is a privacy-preserving distributed ML technique that leverages on-device model training of the network agents utilizing their edge-resources and learning from the accumulated knowledge of the other agents. In FL process, the FL server initializes a global model which is shared with all the available agents within the network. Each selected agent (also called participant) generates a local model using its available data and learning from the global model. The agents apply an optimization method to solve their local objective functions which are shared with the server. The server aggregates all the received local models and generates an updated global model that holds all the latest knowledge of the network agents. The interactions between FL server and agents are continued until the global model achieves a target convergence. While applying FL for HAR within a resource-constrained environment, it is obvious that we may observe agents possessing variant system configurations and data samples. Due to resource heterogeneity, while one agent could perform an assigned task, another agent may struggle to accomplish that task and turn into a straggler. Therefore, it is not reasonable to assign a uniform computational task to all the selected FL agents. Otherwise, if a majority of the FL agents fail to share local model, then the global model quality would increase slowly and overall model convergence would be prolonged [DTN+19]. Besides, as the resource-constrained IoT devices are more prone to attacks, they may inject false model. A comprehensive survey on FL for IoT devices is presented in [ITW+22], where they pointed out such potential challenges (e.g., straggler issues, false model injection) FL process may face due to resource-constrained agents. To recognize human activities in a resource-constrained environment in a distributed

fashion, it is essential to examine agent's resources, and leverage a strategy so that the learning process continues without having any effects due to stragglers or diverge model update. The state-of-the-art FedAvg algorithm [MMR+17] consider that any FL agent that shows interest to be a part of the learning process has sufficient resources for performing an assigned task. Hence, the FL server declares a uniform task for all the selected agents during training. However, in a real-world setting, such assumption is not viable that results in unsuccessful model update from the weak agents. Besides, FedAvg algorithm randomly selects agents for training that increases the risk of selecting vulnerable agents. Considering these, we focus on developing an FL model, particularly for HAR by solving the straggler issues and handling diverge model update. We propose to select proficient and honest agents tracking agents' resource status and their contributions towards model convergence. Besides, we enable partial works from the agents to solve straggler issues during recognition of human activities.

Construction of a Activity and Resource-aware Framework

In our proposed FL-based HAR model, we integrate a mechanism that can assign trust score to the agents based on various events, and select only the proficient and trustworthy agents for the training phase. We consider several events for categorizing trust score, e.g., successful task completion, diverge model infusion, interested to be a part of the training phase, response delay, and incapable to accomplish a given task. We design the trust score table considering the significance of the events, which is presented in Table **1**. At the beginning of FL process, we assign an initial trust score $T_{init} = 50$ to all the federated agents. Any agent who shows interest to participate in training, satisfies the resource requirement to perform task but not selected for training, we assign $T_{\text{Interest}} = 1$ score to motivate that agent to

Table 7.1: Various events and corresponding trust value.

Event	Trust value
$\mathcal{T}_{init}$	50
$\mathcal{T}_{Interest}$	1
$\mathcal{T}_{Reward}$	8
$\mathcal{T}_{Warn}$	-2
$\mathcal{T}_{Blame}$	-8
$\mathcal{T}_{Ban}$	-16

participate in future tasks. Besides, any agent who successfully accomplish a task within a specified time window, we add a reward score $T_{\text{Reward}} = 8$ to that agent's existing trust score. In case, an agent has previous history of becoming straggler in $< 20\%$ of its all participation in the FL process, we assign penalty score $T_{\text{Penalty}} = -2$. In turn, if an agent gives slow response for more than 20%, but less than 50% of its overall participation, then we add up the existing the trust score of that agent by $T_{\text{Blame}} = -8$. Further, any agent that becomes straggler for $>= 50\%$ of its overall activities, or sends back a diverge model, we add a ban score ($T_{\text{Blame}} = -16$) .

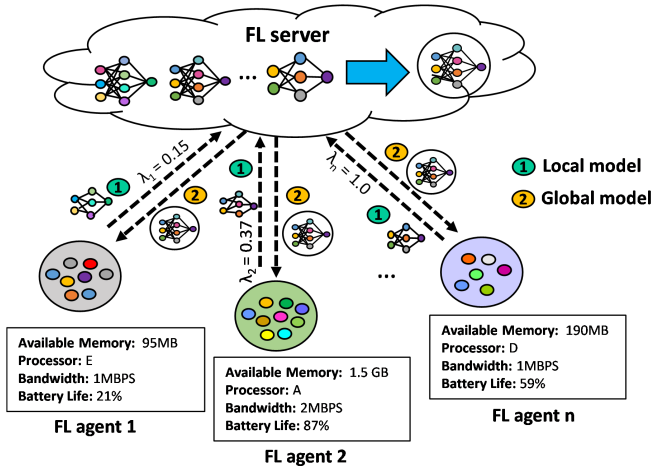


Figure 7.29: Allowing partial works from the Federated Learning agents.

Enabling Partial Works from the Agents

As we discussed in Section **7.2.3**, one of the issues of the FedAvg algorithm [MMR+17] is that it does not allow partial works from the agents. As a result, any agent who fails to complete a whole task due to resource-limitations, can not share its model with the FL server. Such issue hamper the improvement of the global model quality. Considering that, we apply a strategy of generalization of FedAvg algorithm from [LSZ+20] that enable each selected participant to share partial works and upgrade the global model quality by counting every little contributions from the agents. In Fig. **1**, we presented a schematic overview of leveraging partial works from federated agents. It can be observed that while agent n can perform the whole task, agent 1 and 2 can only perform 15% and 37% of the whole task. The assigned task can be considered as the number of local iterations each agent needs to perform. Here, if the whole task is regarded as 100 iterations on the local data, then agent 1 and 2 can perform only 15 and 37 local epoch while the agent n successfully able to complete 100 local iterations. This strategy solves unnecessary waiting time due to straggler agents, and also reflects every little contributions from the agents on the global model quality.

Proposed FL-based Human Activity Recognition Algorithm

We presented our FL-based human activity recognition method in Algorithm **11** considering unreliable and heterogeneous resource occupied federated agents. In the algorithm, the $\mathcal{N}$ eligible FL-based HAR agents are indexed by a; $\mathcal{F}$ = fraction of available agent, $\mathcal{B}$ = local minibatch size, E = local epoch number, η = learning rate, and t = timeout. At the initial stage of the FL process, interested agents commits registration to be a part of the FL-based HAR process (line **1**). An initial trust value is assigned to registered agents, which is updated based on the agent's activities.

Algorithm 11: Federated learning model for human activity recognition in a resource-constrained unreliable environment.

Each interested FL-based HAR agent registers in the FL network
Each agent a possesses a trust score, T_a
System requirements are broadcasted to all interested agents
FL Server executes: initialize global model, w_0
Each FL-based HAR agent reveals its local resource availability
for *each communication round* $i = 1, 2, \ldots$ **do**
    $R_i =$ **ResourceStatus** $(i, \mathcal{P}_a, \mathcal{B}_a, \mathcal{M}_a, \mathcal{V}_a)$
    Sort agents according to T_a and R_a, and keep in a list $\mathcal{L}$
    $\mathcal{E} \leftarrow$ Top $\mathcal{L} \cdot \mathcal{F}$ agents
    $\mathcal{A}_i \leftarrow$ (random set of $\mathcal{E}$ agents)
    for *each agent* $a \in \mathcal{A}_i$ *execute in parallel* **do**
        $w_{i+1}^a \leftarrow$ **AgentModelUpdate** (a, w_i)
        if *Agent a respond within time t* **then**
            $w_{i+1} \leftarrow w_{i+1} + \frac{n_i^a}{n} w_{i+1}^a$
        $T_i =$ **ScoreUpdate** (i, a, w_a, t, γ)
ScoreUpdate (i, a, w_i, t, γ)
ResourceStatus $(i, \mathcal{P}_a, \mathcal{B}_a, \mathcal{M}_a, \mathcal{V}_a)$**:**
    Store $(\mathcal{P}_a^i, \mathcal{B}_a^i, \mathcal{M}_a^i, \mathcal{V}_a^i)$ into a list $\mathcal{G}_a^i$
    Compare $\mathcal{G}_a^i$ with $\mathcal{L}_{Req}^i$
    if $\mathcal{G}_a^i$ *satisfies* L_{Req}^i **then**
        Add $\mathcal{G}_a^i$ in $\mathcal{R}$
    return $\mathcal{R}$
AgentModelUpdate (a, w) : // Run on agent a
    Each FL-based HAR agent a generates a local model w_a^{i+1} which is a λ_a^i -inexact minimizer of: $w_a^{i+1} = F_a(w) + \frac{\delta}{2} \left\| w - w^i \right\|^2$ and determines optimal round of local epoch, E
    $\mathcal{B} \leftarrow$ (split local data point of agent a, $\mathcal{P}_a$ into batch size B)
    for *local epoch* 1 *to* E **do**
        for *batch* $b \in B$ **do**
            $w \leftarrow w - \eta \nabla \ell(w; b)$
    return model w to server

At the beginning, the FL server assigns an initial trust value to all newly registered agents, which is updated accordance to the activities of the agents, and the FL server broadcasts system requirements for performing a task to all available agents (line **2-3**). The FL server initializes the global model, w_0 and shares with the agents (line **4**). Each agent shares information about its available resources with the FL server (line **5**). At each communication round of the FL process, the FL server examines resource status of each interested agents by calling **ResourceStatus** function (line **6-7**). In **ResourceStatus** function, we consider communication round number, processing power ($\mathcal{P}$), battery-life ($\mathcal{B}$), memory ($\mathcal{M}$), and data volume ($\mathcal{V}$) as the resources of the agents, and store the resource status of agent a for communication round i into a list, $\mathcal{G}_a^i$ (line **17-18**). After that, the FL server compares the agent's resource status with the task requirements, $\mathcal{L}_{req}$ (line **19**). If the agent has sufficient resources for that task, then we store the agent's resource status information into a list, R and return the list, $\mathcal{R}$ (line **20-22**). After receiving the resource status information and checking the trust score of the agents, the FL server performs sorting and ranks the available agents, which are stored into a list $\mathcal{L}$ (line **8**). From the list, fraction of agents are selected and further, only a few agents are randomly chosen for the training round (line **9-10**). Each selected agent are requested to perform local computational tasks on their on-device data through **AgentLocalUpdate** function and the latest global model is passed as a function parameter (line **11-12**). After receiving command to perform on-device training from the FL server, each agent uses local solver determining inexact minimizer λ_a^i and performing local training to resolve local objective function (line **23-24**). Each agent splits local data points, $\mathcal{P}_a$ into batches, performs stochastic gradient descent (SGD) optimization technique considering feasible number of local epoch which is determined through local solver as well as using batches, sends back local model to the FL server (line **25-29**). After

receiving model from an agent a, the FL server follows asynchronous FL strategy to perform immediate aggregation for generating a latest global model without waiting for other agents (line **13-14**). The total number of data samples within FL networks is referred by n, which are distributed among the available agents, and the number of local data samples of a during a communication round i is represented by n_i^a.

Algorithm 12: Trust Model. The global model of i^{th} training round is represented by G^i, unsuccessful record of a agent a on iteration i is denoted by $(\mathcal{U}_a^i.)$, trust score C_a for a^{th} client, t represents timeout, and γ indicates deviation.

1 **ScoreUpdate** (i, a, w_i, t, γ): **if** *agent a sends local model* w^i *within* t **then**
2 set $\mathcal{U}_a^i = 0$
3 set $\mathcal{T}_a = \mathcal{T}_a + \mathcal{T}_{Reward}$
4 **else**
5 set $\mathcal{U}_a^i = 1$
6 **if** $\frac{1}{\mathrm{i}} \sum_{\mathrm{c}=1}^{\mathrm{i}} \mathcal{U}_{\mathrm{a}}^{\mathrm{c}} < 0.2$ **then**
7 set $\mathcal{T}_a = \mathcal{T}_a + \mathcal{T}_{Penalty}$
8 **if** $\frac{1}{\mathrm{i}} \sum_{\mathrm{c}=1}^{\mathrm{i}} \mathcal{U}_{\mathrm{a}}^{\mathrm{c}} < 0.5$ *and* $\frac{1}{\mathrm{i}} \sum_{\mathrm{c}=1}^{\mathrm{i}} \mathcal{U}_{\mathrm{a}}^{\mathrm{c}} \geq 0.2$ **then**
9 set $\mathcal{T}_a = \mathcal{T}_a + \mathcal{T}_{Blame}$
10 **else if** $\frac{1}{\mathrm{i}} \sum_{\mathrm{c}=1}^{\mathrm{i}} \mathcal{U}_{\mathrm{a}}^{\mathrm{c}} \geq 0.5$ *or* G^i-$L_a^i > \gamma$ **then**
11 set $\mathcal{T}_a = \mathcal{T}_a + \mathcal{T}_{Ban}$
12 Append $\mathcal{T}_a$ to $TrustList$ and Return

Further, the FL server updates trust score of all participated agents based on their performance. The trust score is updated on calling based on the trust model in Algorithm **12** upon calling of **ScoreUpdate** function in line **15** of Algorithm **11**. If an agent successfully sends local model update within time t, then we assign unsuccessful record during communication round i for agent a, $\mathcal{U}_a^i$ to 0 and give a reward to that agent by increasing its trust score (line **1-3**). In turn, if a client failed to send back model update within time t, we set $\mathcal{U}_a^i$ to 1 and check the agent's previous participation activities. If the $U_k^i = 1$ case occurs less than 20% of that

agent's overall participation history, then we add a penalty score, $\mathcal{T}_{Penalty}$ to the agent's current trust score (line **4-7**). Likewise, if the $U_k^i = 1$ case happens >= 20% but < 50%, then we add a blame score, $\mathcal{T}_{Blame}$ (line **8-9**), and if the $U_k^i = 1$ event happen >= 50%, then a ban score is added to the agent's trust score (line **10-11**). Finally, the updated trust score, $\mathcal{T}_a$ is appended to a list (line **12**) and the score can be used during participation selection in next communication round.

7.2.4 Experimental Results

As we explained in Section **7.2.3**, the trust score of participated agents in the FL process is updated based on their activities. The score is increased whenever an agent successfully accomplish a task and, decreased upon slow response, or improper model update. In Fig. **7.30**, we visualize the activity data points that clearly shows that the dataset values are feasible to classify, and in Fig. **7.31**, we simulate the trust score update of four FL agents corresponding to their activities in different training rounds. As per the trust score simulation of Fig. **7.31**, we select Agent 1, Agent 2 and Agent 4 in training period $t7$.

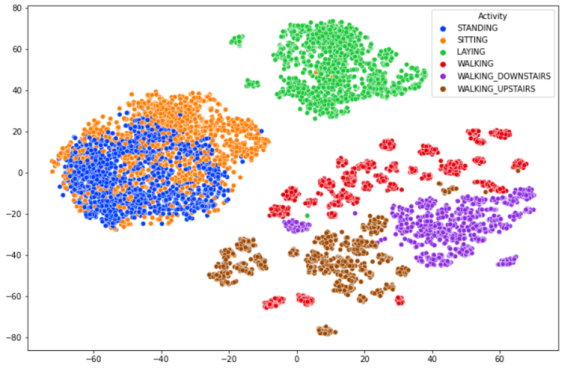


Figure 7.30: Activity data point visualization of human activity recognition dataset.

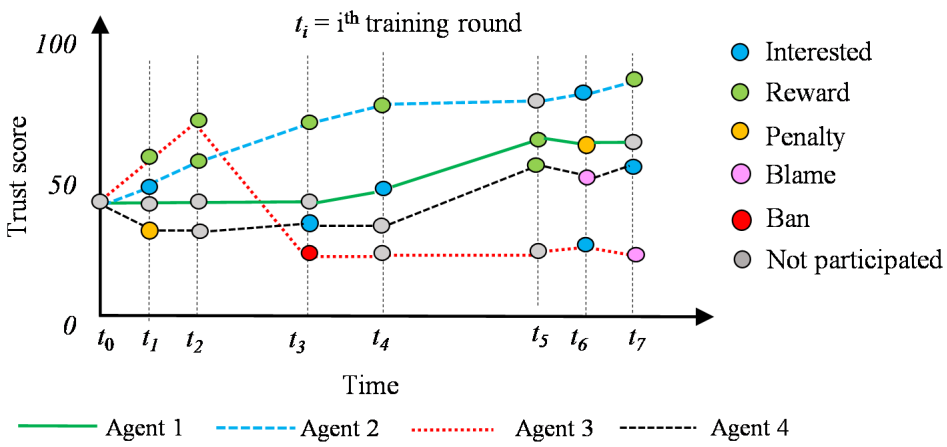
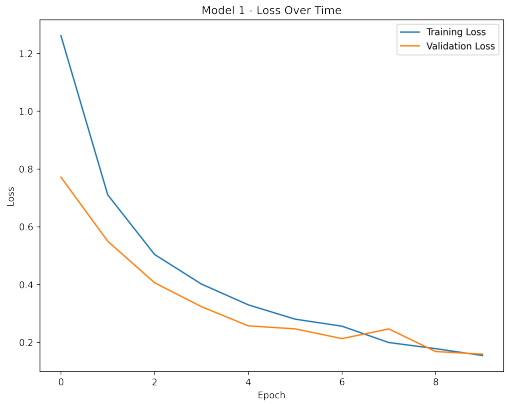
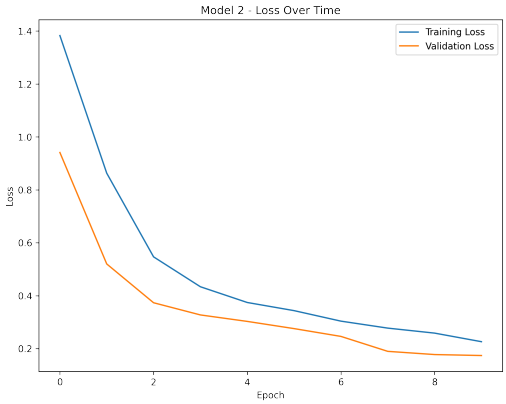


Figure 7.31: Trust scores of four FL agents in various training rounds.

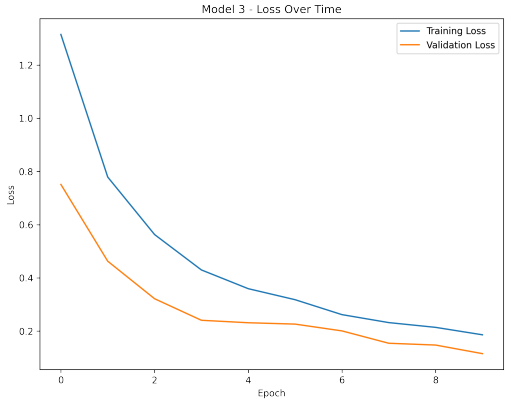
To evaluate our proposed FL model performance, we considered distributed mobile robots as FL agents that possess variant processing capability, battery-life, memory, and data volume. We used the UCI HAR dataset [A$^+$13], which is a multivariate, time-Series dataset consisting of attributes, such as triaxial angular velocity and acceleration measured using gyroscope and accelerometer, respectively, a 561-feature vector consisting of time and frequency variables, the corresponding activity level and an identifier who collected the related data. We distribute data among agents to recognize human activities and simulate how each agent perform on-device training and generate a local model. We set up a similar transmission rate for FL agents to bring simplicity of the implementation process. To simulate the straggler effects and demonstrate the effectiveness of our proposed model in HAR, we deliberately considered a number of weak agents, i.e., the agents fail to accomplish a given task due to limited resources. We assume that a global clock cycle is followed by each agent, and each chosen distributed agent can determine the number of local epoch it can perform using its resources and specified time window in training round i. Instead of using a uniform local epoch E, we apply



(a) Agent 1



(b) Agent 2



(c) Agent 3

Figure 7.32: Training loss of three FL agents applying our proposed FL framework during recognition of human activities.

a generalization of FedAvg algorithm that allows to execute partial works based on its available resources. We apply our proposed FL algorithm (Algorithm **11**) that enables on-device model training of the participated agents, and apply SGD optimization method locally on each agent. The number local iterations on each agent side are dependent on the available resources and local data of the agents. We simulate our model for a small-scale during prediction of human activities considering three distributed agents and simulate the loss and accuracy of those three distributed agents' local models when 1 out of 3 agents are straggler (see Fig. **7.32** and **7.33**). We also tested our model when 2 out of 3 agents are stragglers and

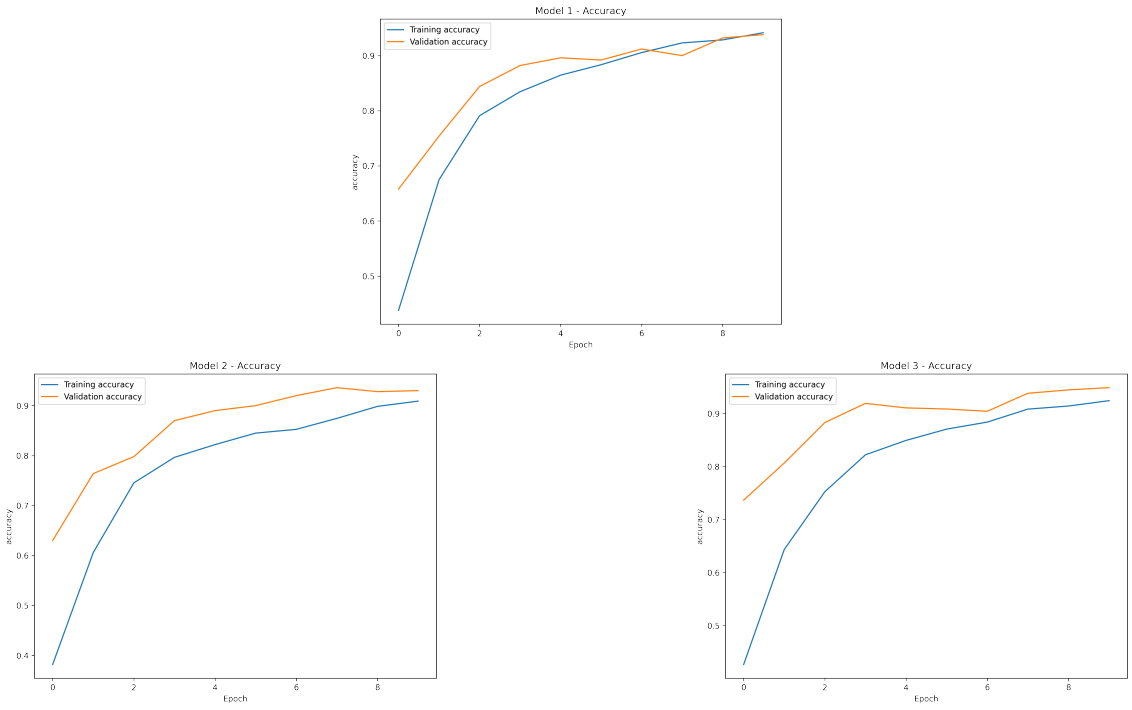


Figure 7.33: Testing accuracy of three FL agents applying our proposed FL framework during recognition of human activities.

Table 7.2: Comparison of our proposed FL model with FedAvg model during prediction of human activities considering different local epochs and batch sizes.

Epoch	**Batch size**	**Global Model Accuracy (1 out of 3 agent is straggler)**		**Global Model Accuracy (2 out of 3 agents are stragglers)**	
		FedAvg	**Proposed Model**	**FedAvg**	**Proposed Model**
5	64	74.25%	76.38%	62.10%	73.45%
10	64	87.90%	90.35%	69.4%	85.7%
10	30	84.35%	87.10%	71.2%	83.4%
15	64	81.4%	84.90%	65.28%	79.8%
15	40	83.10%	85.95%	69.85%	80.5%
20	64	85.25%	88.20%	69.25%	84.10%
25	64	86.6%	89.4%	71.2%	84.9%
25	60	82.8%	86.04%	68.8%	80.4%

compare our results with the FedAvg [MMR+17] algorithm for the same simulation settings. In table **7.2**, we demonstrated that the global model accuracy of our pro-

posed FL model outperforms the FedAvg [MMR+17] algorithm during prediction of human activities in presence of stragglers. Here, the epoch resembles the number of local epoch that is considered as the amount of whole computational task and batch size refers to how the local data are splitted to apply SGD optimization method. We found that for the epoch 10 and batch size 20, our proposed FL model shows superior performance even in presence of stragglers.

7.2.5 Conclusion

We developed an FL model for HAR that can be effectively applied in a highly unreliable resource-constrained environment. The existing prediction method based on HAR require human activity data in a central fusion center that could violate privacy and also do not consider the generation of the prediction model from the federated agents that could reveal significant information. Besides, there is currently no existing application on FL that consider the HAR within a resource-constrained environment. We integrated a resource-checking and trust score scheme within the FL process that helps us to find out proficient and trustworthy agents for the training phase. Further, we leverage a generalization of FedAvg algorithm that allows the resource-constrained agents to perform partial tasks instead of accomplishing the whole tasks that mitigates the straggler effects and helps the FL server to count every little contributions from all the participated agents. We tested the performance of our proposed FL model using HAR dataset and considering a real-word setting, and achieved a superior performance comparing to the existing HAR methods.

7.3 Leveraging Asynchronous Federated Learning to Predict Customers Financial Distress

7.3.1 Abstract

In recent years, as economic stability is shaking, and the unemployment rate is growing high due to the COVID-19 effect, assigning credit scoring by predicting consumers' financial conditions has become more crucial. The conventional machine learning (ML) and deep learning approaches need to share customer's sensitive information with an external credit bureau to generate a prediction model that opens up the door of privacy leakage. A recently invented privacy-preserving distributed ML scheme referred to as *Federated learning (FL)* enables generating a target model without sharing local information through on-device model training on edge resources. We propose an FL-based application to predict customers' financial issues by constructing a global learning model that is evolved based on the local models of the distributed agents. The local models are generated by the network agents using their on-device data and local resources. We used the FL concept because the learning strategy does not require sharing any data with the server or any other agent that ensures the preservation of customers' sensitive data. To that end, we enable partial works from the weak agents that eliminate the issue if the model convergence is retarded due to straggler agents. We also leverage asynchronous FL that cut off the extra waiting time during global model generation. We simulated the performance of our FL model considering a popular dataset, Give me Some Credit [LLL11]. We evaluated our proposed method considering a a different number of stragglers and setting up various computational tasks (e.g., local epoch, batch size), and simulated the training loss and testing accuracy of the prediction

model. Finally, we compared the F1-score of our proposed model with the existing centralized and decentralized approaches. Our results show that our proposed model achieves an almost identical F1-score as like centralized model even when we set up a skew-level of more than 80% and outperforms the state-of-the-art FL models by obtaining an average of $5 \sim 6\%$ higher accuracy when we have resource-constrained agents within a learning environment.

7.3.2 Introduction

In this section, we start with the motivation of developing an FL model for customers' financial distress prediction. After that, we discuss and analyze the prior works that are developed, particularly in the FL domain. To the end, we present our contributions, which is followed by presenting the organization.

Motivation

Banking sectors are recently facing huge challenges due to the borrower's unwillingness and inability to repay their borrowed money, leading to bad debts. The fundamental challenge is to undertake appropriate decisions while giving loans and identifying unqualified applicants. One solution towards tackling such a problem is to perform an analysis of risk management that requires a necessary amount of agent data. Besides, the heavy consumption of credit limits by the unpaid customers can also cause severe disasters for the banks. In 2019, $22,780$ business bankruptcy cases were filed in the US [ST19]. To avoid such huge losses, one of the popular solutions is to construct a powerful model to predict customers' financial disasters by storing the training data centrally. In such an approach, the financial company holding customer's data performs labeling and passes to a central cloud server for

model training. After proper training, the model can be used for inference tasks. However, in such a method, the financial companies or the customers do not have any control over how and where their data would be utilized once they are shared with the central database server. Besides, the traditional centralized model training has the following problems:

- Nowadays, it is getting difficult to broadcast data across different organizations because of liability concerns. Different data sharing regulations (e.g., the General Data Protection Regulation (GDPR)) limit data sharing across organizations.
- The overall centralized model training takes a long period which depends on when the next off-line training round occurs. Whenever an agent collects some data, it must need to upload the data in the server and have to wait for the next training round, which the agent cannot control. This encounter lagging feedbacks that delay the error correction in model inference.
- The vast amount of data that is updated and stored in a central server can cause a high communication cost.

As we mentioned earlier, developing a prediction model using customers' data have greater risks of privacy violation. Besides, the prevailing training approaches may face issues in a distributed learning setting due to response delay by the network agents, infrastructure fallibility, or weak network connectivity [LO$^+$18]. To mitigate such losses, a prediction scheme needs to be constructed that deals with sensitive information of the agents while maintaining privacy. A recently popular distributed ML technique called *Federated learning* preserves agent's data privacy by performing computation at the edges without sharing any local data and generating a cumulative model utilizing the model information of the edge devices. In recent times, the FL

applications are highly acceptable due to their ability to preserve user privacy. On top of that, FL can handle non-independent and identically distributed (non-IID) data [MMR+17, BIK+17], i.e., FL can still work if data samples are not uniformly distributed among the agents. The typical FedAvg algorithm [MMR+17] assigns a similar number of local epochs using their available data. However, in a real-world setting the situation could be different. Any of the FL agents may have a huge amount of size while other agents may have fewer data samples. If we think about the same scenario considering credit bureau as the server and the banks and financial agencies as agents, then a bank may possess a larger amount of agent information while a financial agency may have comparatively less agent information. Thus, the FL agents may require a variant number of local computations depending on the number of their available samples. However, the state-of-the-art FedAvg algorithm [MMR+17] does not propose any mechanism to handle straggler agents and there is no existing FL application that can predict customer's financial distress considering a resource-constrained environment. We aim to develop an FL model using agent's sensitive financial information without sharing those with any external entity. That means, unlike conventional ML approaches, we do not pass any raw data to the outside server, i.e., each bank does not share their agent information with a credit bureau or any other bank that could potentially violate privacy [YLCT19a]. In this way, the personal information of the agents has less probability to be known by the outsider, and agents' confidentiality can be preserved. Further, we adopt a generalization of the FedAvg algorithm to assign variant local epochs based on the available data of the banks and financial agencies so that they need to utilize fewer resources and can generate a local model without overfitting. [2]

[2]Throughout this dissertation chapter, FL agent indicates banks or financial agencies that hold the customer's data.

Literature Reviews

Over the years, several methods have been proposed to assist decision-makers and analysts with useful methods to predict financial distress considering various mathematical models and financial parameters that include logistic regression [UHZN17], linear regression [PR19], support vector machines [SW18], deep learning [Jan21], and artificial neural network [TRRK20]. Besides, numerous centralized techniques are developed to forecast personal bankruptcy by mining customer's data. For instance, the authors in [XWMM13] proposed a sequence mining technique to apply in credit card data by applying a SVM classifier. The sequence mining reveals the bankruptcy features which are used as the main predictors. Besides, the authors in [JJW15] examined the performance of various binary classifiers for credit rating application. Moreover, the authors in [Lee07] applied a grid-search algorithm to figure out the optimal parameter of SVM for predicting credit rating of the customers.

A strategy of forecasting digital currency using meta-heuristic signal processing methods is proposed in [AKB19], while a prediction model using multi-objective optimization strategy in presented in [KABA20]. Furthermore, the authors in [KASH18] proposed an ML model to forecast the Bitcoin process using time-series data. However, all these proposed methods are based on the theme of sharing sensitive customers' data to a central fusion center for model construction, which fails to preserve customers' data privacy. It is because sharing sensitive data (e.g., customer's financial information or credit card data) with other entities is privacy-intrusive and if we consider a large-scale network, then the computation time increases as the server needs to process a huge amount of data. Besides, processing huge data streams results in high-computational and maintenance costs. Therefore, instead of centralizing customers' data, we need to apply a strategy that generates an effective model without sharing or transferring any customer's sensitive data.

The research on the FL domain is growing prodigiously due to its benefits in maintaining privacy, handling non-IID data, sharing computational power, and encountering less communication overhead [MMR+17]. FL process maintains privacy by not exposing any agents' private data through on-device model training [YLCT19b], while computational burdens are distributed among the network agents instead of imposing on a central fusion center [LSZ+20]. The authors in [BIK+17] proposed a secure protocol of aggregating user data by encountering a low overhead, which can be used in an FL setting. Besides, an activity and resource-aware FL model is proposed in [IA20] that is designed for distributed mobile robots and can handle non-IID data while tracking the agents' resource status. Several kinds of research from different disciplines have been conducted to improve FL strategy, including ML, data mining, distributed systems, and cryptography. The FL term and the FedAvg algorithm were first introduced in [MMR+17], where they discussed how FL can be effective to generate a smart model preserving user's data privacy and distributing the processing loads among federated agents. Later on, the authors in [BEG+19a] pointed-out the complications while developing FL-based systems and proposed solutions towards designing effective FL model by avoiding challenges of a distributed learning environment, e.g., statistical heterogeneity, systems heterogeneity, etc. Several comprehensive survey papers are available including [KM+21] that explains the concepts of FL in more detail and analyzes the potential solutions of some of the existing FL challenges and presents the future directions. The authors in [LLH+20] conducted a detail analysis on optimizing FL for a large-scale mobile edge network by considering a heterogeneous edge devices and their varying constraints. A tri-layer FL framework is proposed by the authors in [IA21] that can shrink model size of the resource-constrained agents and can also handle systems and statistical heterogeneity. Besides, an FL model is designed in [SWMS19] that

can handle non-IID data and suitable for low bandwidth and low latency channels. Moving forward, a distributed sensing mechanism is proposed in [IA19] that can remotely trigger any of the network agents and prepare a federated dataset by encountering a very low communication cost. Their proposed mechanism is particularly useful when we have a large number of IoT devices within the network and the management and scheduling of those devices is challenging.

The privacy-preserving nature of FL attracts researchers to develop different FL applications for solving real-life problems. Particularly, the FL method fits best where we need to deal with sensitive data. For instance, heart attack prediction of a patient through personalized model training [HY+18], device-centric recommendation system [TLZY20], wake-word detector through on-device model training and recognition of user speech [LCL+19], leveraging resilience [IKKA21], recommendation system [YJSD20] and so on.

Researchers are coming forward to solve different issues during FL model training and improve its applicability in solving real-life problems. To solve the non-uniform distribution of samples, some recent works proposed to model target distribution or force the agent samples to adapt uniform distribution. Specifically, Mohri et al. [MSS19] designed a minimax optimization technique, named agnostic federated learning, where the global model is optimized for any target distribution that is formed by the infusion of agent distributions. Their proposed method is applied only at small scales. The authors in [LSS19] proposed q-Fair Federated Learning (q-FFL), where they assigned higher weights to agents with poor prediction performance so that the accuracy distribution within the network lowers in variance. The authors in [ZLG20] proposed an experience-driven FL model to improve the energy efficiency of the overall process by handling agents' CPU-cycle frequency. They formulated their objective function considering the training time of the agents and their respective

energy consumption. A joint FL agent scheduling and resource allocation strategy is proposed in [SZN+20] to optimize model accuracy considering a limited training time budget for a latency-constrained FL setting. By allocating more bandwidth to the agents with weaker computational abilities or worse channel conditions, they carry out a greedy algorithm-based scheduling approach during the training process. One of the potential challenges to applying FL in a resource-constrained environment is the straggler effects due to systems heterogeneity [ITW+22]. However, none of the proposed works deal with the effects of straggler issues due to systems heterogeneity when we apply FL model training in a resource-constrained environment.

We proposed a novel FL algorithm for predicting customers' financial distress by preserving privacy and considering the straggler issues. Our proposed method is effective when a large number of distributed agents possessed limited resources. To the best of our knowledge, there is currently no existing FL application for forecasting customers' financial distress. The existing theory of the FL method states that an algorithm can be applied across distributed edge agents without ever collecting any private data from them and a final model can be generated through a weighted average of the collected models that holds all the local information of the distributed agents. That means the FL technique enables each agent to gain the capability of predicting an event that is never seen or observed by that agent, but similar to that event observed by another network agent. However, the existing theory of the FL method missed the underlying issues when we apply that in a resource-constrained environment. In such an FL setting, a majority of agents would turn into stragglers, i.e., they cannot perform an assigned computational task. Thus, we proposed a generalization of the FedAvg algorithm that enables partial works from the straggler agents and accelerates the learning process by mitigating the straggler effects. Instead of dropping the weak agents as like the existing theory, the

local resources of the distributed agents are examined, and a feasible computational task is assigned to each agent that considered each agent's local knowledge into the global model. To preserve privacy and generate a smart model by collecting local model information from various financial sources, we construct an FL model that is trained based on the agent's personal information, financial stability, and loan information. Without collecting all the information of the agents and train model within a central server, each FL agent, e.g., bank, financial agency, or organization locally train model utilizing their available data and share their model information with the credit bureau (FL server). Instead of assigning a uniform local epoch to every FL agent, we set a variant local epoch based on the available data volume of the FL agents. To the end, the agent performs an asynchronous FL process to generate an updated global model using the collected model information of the FL agents. The main challenge of our proposed method is to quantify the amount of local computational tasks for the resource-constrained agents and ensure that a sufficient number of data samples are possessed by the participated agents. The main limitation is that our proposed method is not able to deal with a situation if the agents possess heterogeneous local models, i.e., we did not consider model heterogeneity of the distributed agents.

Contributions

We propose an FL model to predict an agent's (or loan requester's) financial situation by considering variant local epochs for the data holders of the agents (e.g., banks, financial organizations). We leverage FL strategy that considers customer's local resources to assign computational tasks during training. Particularly, the local computational tasks of each FL agent are assigned based on their data volume, bandwidth, and network availability. We analyze our prediction model by consider-

ing various batch sizes and agent numbers for the training phase. To the end, we visualize the performance of our FL model compared with a centralized model, and also with a mean local model, and the best local model in an FL process. Our key contributions are listed below:

- To the best of our knowledge, we introduced the first FL application on predicting the financial distress of the customers by analyzing their previous transaction history.
- Our proposed prediction model leverage on-device training of the agents and does not require any sensitive information to develop the prediction model.
- Our developed model utilizes the edge resources and cut-off the processing burden from a central fusion center.
- The proposed FL model is effective in a resource-constrained environment; i.e., when a large number of distributed agents have limited resources.
- To demonstrate the effectiveness of our proposed model in predicting financial distress, we evaluate the performance by considering a various number of stragglers and compare the model with the existing methods.

7.3.3 Background of Federated Learning and Our Proposed approach

FL term was first introduced in [MMR+17] where they proposed a popular FL algorithm called Federated Averaging (FedAvg). The algorithm was designed to manage multiple agents collecting data and a server coordinating the local update from the distributed agents to leverage global learning objectives. In particular, the goal is to minimize:

$$\min_{\omega} F(\omega) := \sum_{c=1}^{R} P_c F_c(\omega), \tag{7.1}$$

where, R is the number of agents, $P_c \geq 0$ $\sum_c P_c = 1$ ($P_c \geq 0$) which defines the relative impact of a local agent, satisfying $\sum_c P_c = 1$. The local objective of an agent measure the risk over possibly variant data distributions $\mathcal{D}_c$, i.e., $F_c(\omega) = E_{\omega_c \sim \mathcal{D}_c}$ $[f_c(\omega; \omega_c)]$, where r_c samples are available at each agent c. Therefore, $p_c = \frac{r_c}{r}$, where $r = \sum_c r_c$ refers to the total data points across the network.

Federated learning (FL) can be defined as a distributed machine learning technique that allows the edge agents to generate models based on their local data and builds an aggregated global model on the server learning from all the edge agents. The agents generally share their generated model or algorithm without passing raw data directly to the server. Thus, as data remains on-device, privacy is maintained to some extends. Besides, there could be a various number of agents that are interested to be a part of the FL process. However, the FL process considers a fraction of the agents to be part of the training process. One benefit of considering a fraction of agents could be to make the FL process simple, i.e., when we have a lot of agents within an FL environment, considering a subset of the agents would let us deal with a smaller number of agents; and the other benefit is that the agent activities can be handled effectively. They showed how to perform model training in several iterations through a server-agent interaction and finally, produce a smart model by learning from participated local agents. For initiating the process, the initial global model that is shared with all the agents and for reducing communication overhead, they perform a predefined number of local epoch on each agent side. Further, each agent performs local epochs by splitting data into batches and an optimization method to reach closer to the optimal local solution. Finally, the prepared local model by each agent is shared with the server.

FL generally includes the following three steps:

Step 1 (Initialization of Global Model): At the initial stage, the FL server initializes a global model for a specific task and selects a fraction of available agents for the training phase that are called participants. The initialized global model is disseminated to all the participated agents.

Step 2 (On-device or Local Training): After receiving the global model, each participant carries out on-device training on its local data and also learning from the global model. The generated local model is shared with the server.

Step 3 (Accumulation of Agent Models): The FL server waits for the response of the FL participants and upon receiving their model information, the FL server aggregates the model parameters and generates the latest global model. Again the FL server selects a new subset of FL agents and the updated global model is again shared with all the newly selected participants. The iterative process of step 2 and step 3 is continued until the global model reaches to a target convergence. We presented an overview of the FL process in Figure **7.34**.

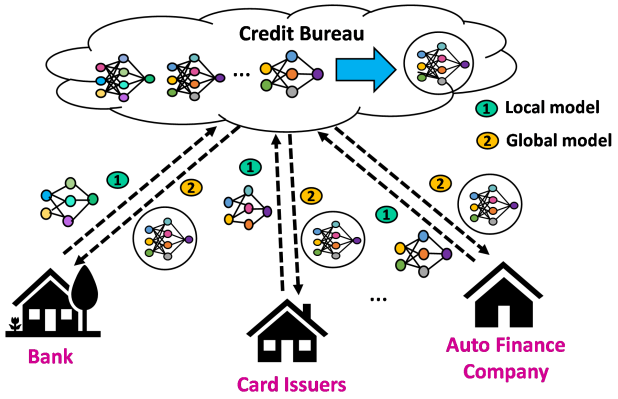


Figure 7.34: Overview of Federated Learning process for modeling agent's financial status prediction.

However, the FedAvg algorithm does not guarantee convergence when we have non-IID data within the network and the majority of the participated agents give

a slow response (called stragglers agents). The FedAvg algorithm simply drops the stragglers from the training round and in case most of the participated agents are dropped, then the training period would be prolonged, or convergence cannot be achieved. Instead of that, if we assign a local computational task to each agent based on their resources and count every little contribution from all participated agents including the stragglers, then our model convergence time can be accelerated. The proposed FL model is particularly useful when we have heterogeneous agents within a network in terms of available resources and non-uniform distribution of local data, and most of the participated agents have resource-constrained issues. In such situations, the traditional distributed learning techniques for financial distress prediction fail to generate an effective model. Besides, our proposed FL model can capture the global knowledge of the customers' financial distress while not exposing any customer's private data, while the traditional learning techniques are privacy-intrusive, and also not able to capture the global knowledge. Our main objective is to minimize the loss of the global objective function by performing a weighted average of the participated agents' local tasks in presence of a large number of stragglers. We enable a dynamic local task allocation technique through which the distributed agents can perform variable computational tasks according to their available resources. In that way, if a straggler holds vital information, we still can include that in the global model instead of dropping it from the training round. The proposed FL model can be significantly useful for financial sectors, where customers privacy has utmost importance and different financial sectors do not want to exchange the information due to customers privacy but still want to have the benefit of the prediction model.

A high level overview for modeling customer's financial status prediction scheme is illustrated in Figure **7.35**. We considered Credit Bureau as the FL server and

Bank, Card Issuers, and Auto Finance Company as the FL agents. At the initial stage, the FL server initializes a global model for a specific task and acknowledges each agent about their local computational task to perform, Upon receiving the global model, each participant performs the assigned computational task through on-device training and share their model with the server, and finally, the server performs aggregation on the received models. In Figure **7.35**, we assume that Bank and Auto Finance Company have limited resources and Card Issuers has sufficient resources. Considering the resource availability, Bank and Auto Finance Company perform 40% and 52%, respectively, of the overall computational task, and Card Issuers perform the whole task. To make the idea more clear, let us assume that the task publisher assigns a task to perform a 100 local epoch. However, some of the agents may not be capable to perform 100 local epochs or may have a few data for which excessive local epoch may overfit the model. Hence, if an agent has fewer resources, then it is allowed to perform a suitable number of local epoch and send back the model update to the FL server. This technique solves two issues: 1) it is not required to wait for a straggler agent while updating the global model, and 2) every participated agent's contribution can be counted. We provided details of FL-based modeling customers' financial distress prediction in Section 7.3.4.

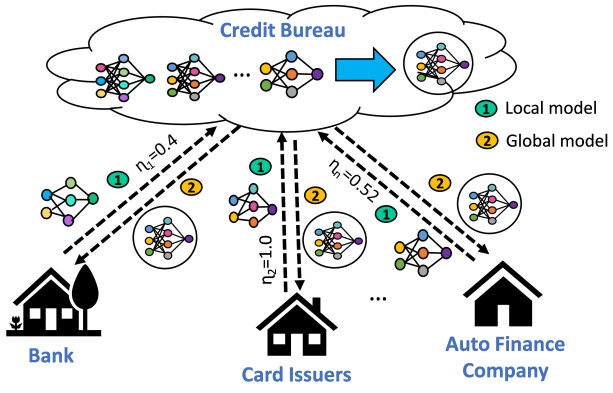


Figure 7.35: Our proposed Federated Learning approach for modeling agent's financial status prediction.

7.3.4 System Description

We assume that similar features of customers' personal and loan information are available to different banks, financial agencies, and card issuers. We consider each of such information sources as an FL agent and take into consideration a credit bureau as an FL server. A credit bureau is generally responsible for collecting information from different sources and providing suggestions to third parties on whether a loan can be given to a certain customer. However, many customers would not feel safe sharing their information directly with an external server as there is a high chance of privacy violation. Our proposed method is particularly suitable for customers' financial distress prediction task because we generate a smart prediction model that captures the knowledge of all necessary financial sectors even without sharing any raw data. Our method can generate a prediction model by distributing computational tasks among participated agents and setting up dynamic task allocation for resource-constrained distributed agents. Compare to the existing approaches, this is the first FL application for customers' financial distress prediction that can carry out model construction in a distributed fashion while preserving privacy and even when the distributed agents have systems heterogeneity and variant local data volume. We presented the flowchart of the overall process in Figure **7.36** and described our proposed approach step by step as follows:

A Global Model Initialized by FL server

At first, a task publisher define the task that is resided on the server. A global model is initialized by the FL server, which is disseminated to all the available network agents such as available banks, financial sources, and other agencies. All the interested agents share their consent with the credit bureau that acts as an FL server to be a part of the training process. That global model can be considered

a function and particularly, the credit bureau initialized the hyperparameters (i.e., coefficients, bias) and the learning rate of the global model.

Broadcasts Global Model Participants

After the global model initialization, the FL server (i.e., credit bureau) tracks agents that show interest to be a part of the training process. The FL server shares the hyperparameters (i.e., coefficients, bias) and the learning rate with the agents holding the customer's information such as available banks, financial sources, and other agencies.

Selection of FL Participants

After acknowledging the task and receiving the global model, a number of agents (e.g., banks or financial agencies that could hold customers' information) would show interest. However, we do not consider all the FL agents for the training phase because the high number of FL agents may overly complicate the overall process and the waiting time of the FL server would be prolonged to make sure that all the local models' update is considered while updating the global model. Instead, a subset of FL agents are selected for the training phase that shows interest and has sufficient available computational resources. Each selected agent constructs a linear model based on its available local data and applies an optimization algorithm, Stochastic Gradient Descent (SGD) until the local model finds out optimal solution to minimize the loss function.

Perform On-device Training and Generation of Optimal Local Model

If we consider a real-life FL setting, then some agents may possess only a few data while the other may contain a large amount of data. Therefore, assigning

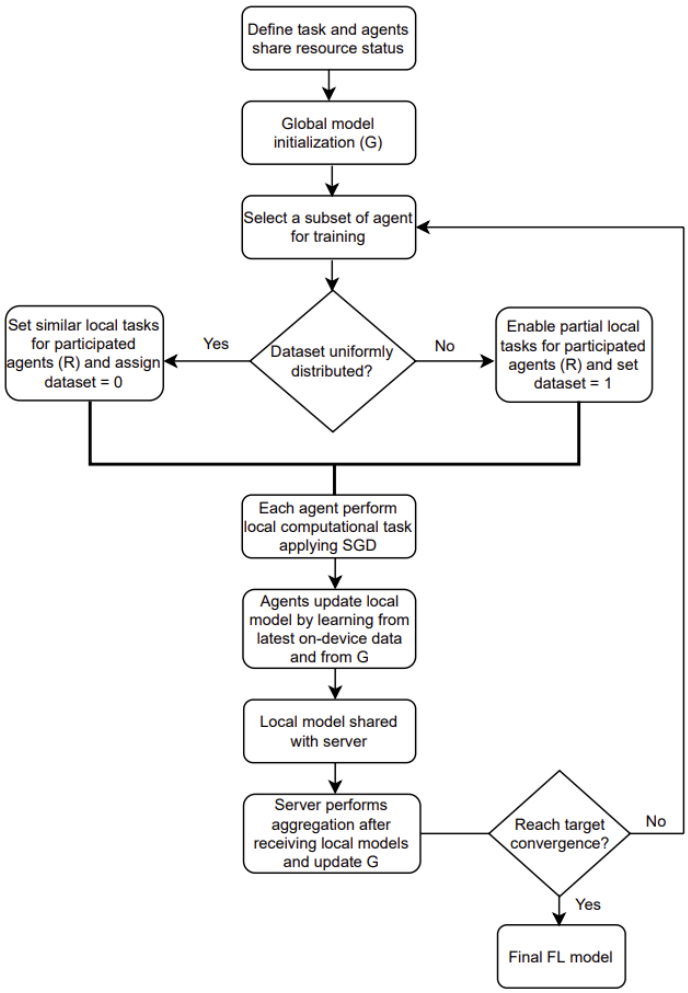


Figure 7.36: Flowchart of the proposed Federated Learning process to predict customer's financial disaster.

the same number of local epochs would not be feasible and also not effective for the overall learning process. Considering the data volume and resource availability, some agents may need to perform a higher number of local epochs to accomplish a target convergence while some agents may be capable to perform only a few rounds of the local epoch. Considering that, we proposed an FL model for customer's financial distress prediction by adapting an inexact solution through the generalization of the state-of-the-art FedAvg algorithm [MMR+17]. Particularly, the inexact solution can

help an agent to determine the number of local epoch (i.e., computational tasks) it needs to perform by analyzing its on-device resources. The η_c^r -inexactness for performing variant computational tasks by a agent c at training round r can be defined as follows [LSZ+18]:
Let us consider a function, $\mathcal{G}_c(\omega;\omega_r) = \mathcal{F}_c(\omega) + \frac{\alpha}{2}\|\omega-\omega_r\|^2$, and $\eta \in [0,1]$, we call ω^* is a η_c^r -inexact solution of $\min_\omega \mathcal{G}_c(\omega;\omega_r)$ if $\|\nabla\mathcal{G}_c(\omega^*;\omega_r)\| \leq \eta_c^r \|\nabla\mathcal{G}_c(\omega_r;\omega_r)\|$, where, $\nabla\mathcal{G}_c(\omega;w_r) = \nabla\mathcal{F}_c(\omega) + \alpha(\omega-\omega_r)$.

Here, the effectiveness of η-inexactness is that it enables variable local computational tasks to be performed by the selected FL agents. Due to systems heterogeneity, as the agents may have dissimilar progress in solving local objective functions, therefore, it is necessary to enable adaptive η considering the FL agent's available resources. After receiving the global model, each agent splits their samples into batches and constructs a model by applying SGD optimizer on the mini-batch of data. Each agent performs a predefined number of local epochs that are set by the FL server and it is calculated based on the available number of customer data.

Update Global Model and Repeat the Training Process

After allocating variant computational tasks, the FL server waits for the local model update from the participated agents and performs an aggregation whenever it receives a model from any of those agents.

As the participated agents may have variant local epochs that may take different computation times for generating the local model, thus, as soon as any model is received by the FL server, it performs aggregation on the local models for updating the global model. That means, we perform asynchronous FL approach to update the global model. The key contrast of asynchronous FL with a synchronous FL is that in the synchronous FL approach, the FL server only updates the global model

when it receives a model from all the agents while in the asynchronous FL method, the global model update is performed whenever an FL agent model is received by the FL server. After updating the global model, the FL server shares that latest global model with the network agents and newly selects participants for the next round of the training phase. The same process is continued until the global model completes a predefined iteration round or reaches to the target convergence. We presented the flowchart of the overall process and the framework of our proposed model in Figure **7.36** and **7.37**, respectively.

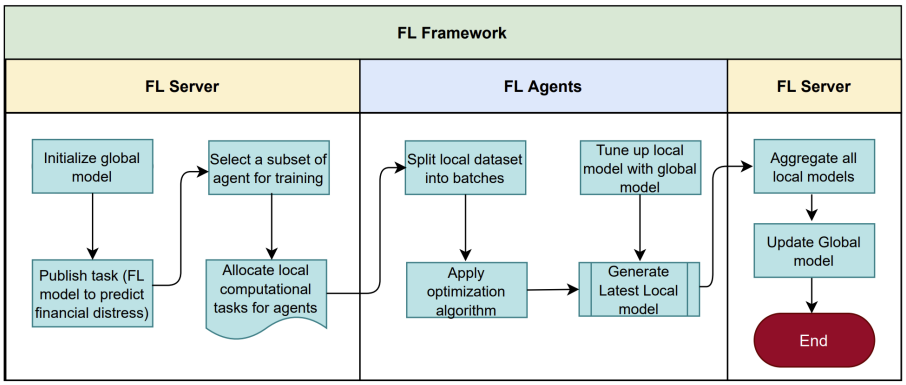


Figure 7.37: Federated Learning Framework to predict customer's financial disaster.

FL Model for Predicting Customer's Financial Disaster

In Algorithm **13**, we presented how we modified FedAvg [MMR+17] algorithm to perform prediction of customer financial disaster. Initially, the FL server (i.e., the credit bureau) defines the task (line **1**) and also initializes a global model with required parameters (line **2**). In every training round, the FL server chooses a fraction of agents, which are called participants (line **4-5**). We assume that there are total n number of data samples available within a network that are possessed by distributed agents. If the data samples are uniformly distributed, then we assign a similar local epoch to all participants (line **6-7**). In case of non-uniform distribution

of the data samples, the FL server requests each participant to share the status of their available data volume and the FL server estimates a local epoch for each individual participant V (line **8-10**). After that, the FL server call each participant by passing the agent id, latest global model and local epoch to find out an optimal local solution (line **11-12**). Each FL participant receives the shared global model, splits their local samples into batches, performs a local training based on the required number of local epochs, and shares that local model with the FL server (line **15-20**). Whenever the FL server receives an updated local model from any of the participants, it aggregates the received models to generate an updated global model (line **13-14**).

7.3.5 Experimental Evaluation

7.3.6 Simulation Settings and Dataset Preparation

For our simulation, we used Python programming language and tailored Google Colab developed by Google as the simulation environment. Google Colab is an online simulation framework for writing and executing ML, distributed ML, and deep learning codes. The simulation platform provides various runtime environments and python versions for the successful execution of the codes. It is also effective for downloading and mounting larger datasets directly from the server swiftly and provides a high computation speed during the construction of a complex learning model. We used the Give me Some Credit dataset [LLL11] from Kaggle that has $1,50,000$ data samples with a shape of $(150000, 12)$. The features are agent id, age, monthly income, debt ratio, number of dependents s, number of running credit account, number of loans or lines in real estate, revolving utilization of unsecured lines, number of times 30-59 days past but due unpaid, number of times 60-89 days

Algorithm 13: FL Model for Customers Financial Disaster Prediction. The number of available agents within the FL network is denoted by $\mathcal{N}$, the fraction of agents for the training phase is represented by f, local data samples on each agent is $\mathcal{P}_c$, number of local epoch on an agent is $\mathcal{E}$, and η is the learning rate.

Task is initiated and interested agents' resource status is received by credit bureau that acts as FL server
A global model ω_o is initialized by the credit bureau
for every training round $r = 0, 1, 2, 3 \ldots$ do
    $s \leftarrow \max(\lfloor \mathcal{N} \cdot f \rfloor, 1)$
    $R =$ random set of s agents
    if *dataset is uniformly distributed* **then**
        set a similar local epoch for all participants
        dataset = 0
    else
        Each agent acknowledges server about its available sample size V
        Determine number of local epoch $\mathcal{E}_c$ for each participant V
        dataset = 1
    for each agent $c \in R$ in parallel do
        $\omega_{r+1}^c =$ LocalTraining $(c, \omega_r, \mathcal{E}_c)$
        if *model received from c within time t* **then**
            $\omega_{r+1} \leftarrow \sum_{c=1}^{\mathcal{N}} \frac{n_c}{n} \omega_{r+1}^c$
LocalTraining $(c, \omega, \mathcal{E}_c)$:
    if *(dataset == 1)* **then**
        Each agent finds a ω_c^{r+1} which is a η_c^{r+1} -inexact minimizer of: $\omega_c^{r+1} = F_c(\omega) + \frac{\alpha}{2} \|\omega - \omega^r\|^2$ and determines maximum feasible local computational task (i.e., local epoch) $\mathcal{E}$ to perform
    $\beta \leftarrow$ (Split $\mathcal{P}_c$ into batches of size $\mathcal{B}$)
    for each local epoch i from 1 to $\mathcal{E}_c$ do
        for batch $b \in \beta$ do
            $\omega \leftarrow \omega - \eta \nabla \ell(\omega; b)$
    return ω to server

past but due unpaid, number of times 90 days past but due unpaid. The target column has a label that indicates whether the agent would face financial disaster in the next two years or not. The dataset was highly imbalanced, i.e., the number of samples with *label* 0 was 139974 while the samples with *label* 1 was only 10026 in number. To make our prediction model unbiased due to the imbalanced dataset, we performed downsampling on the dominant class *label* 0 using *resample*() function and matched the number of samples or instances of a class with *label* 1. After that, we used train_test_split to separate train and test data and allocated test size as $\frac{1}{3}$ of the whole dataset. Unlike conventional ML approach, we distribute the whole dataset among a number of agents who are responsible for performing local computation on those distributed data, generating a model and sharing that with the FL server.

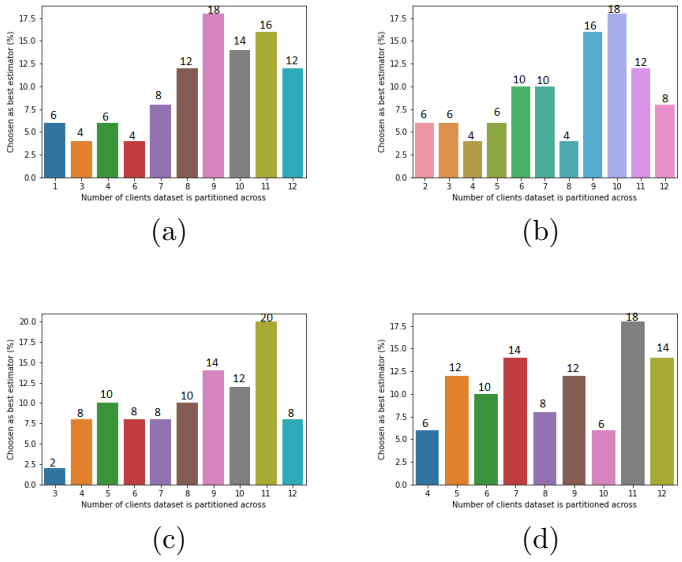
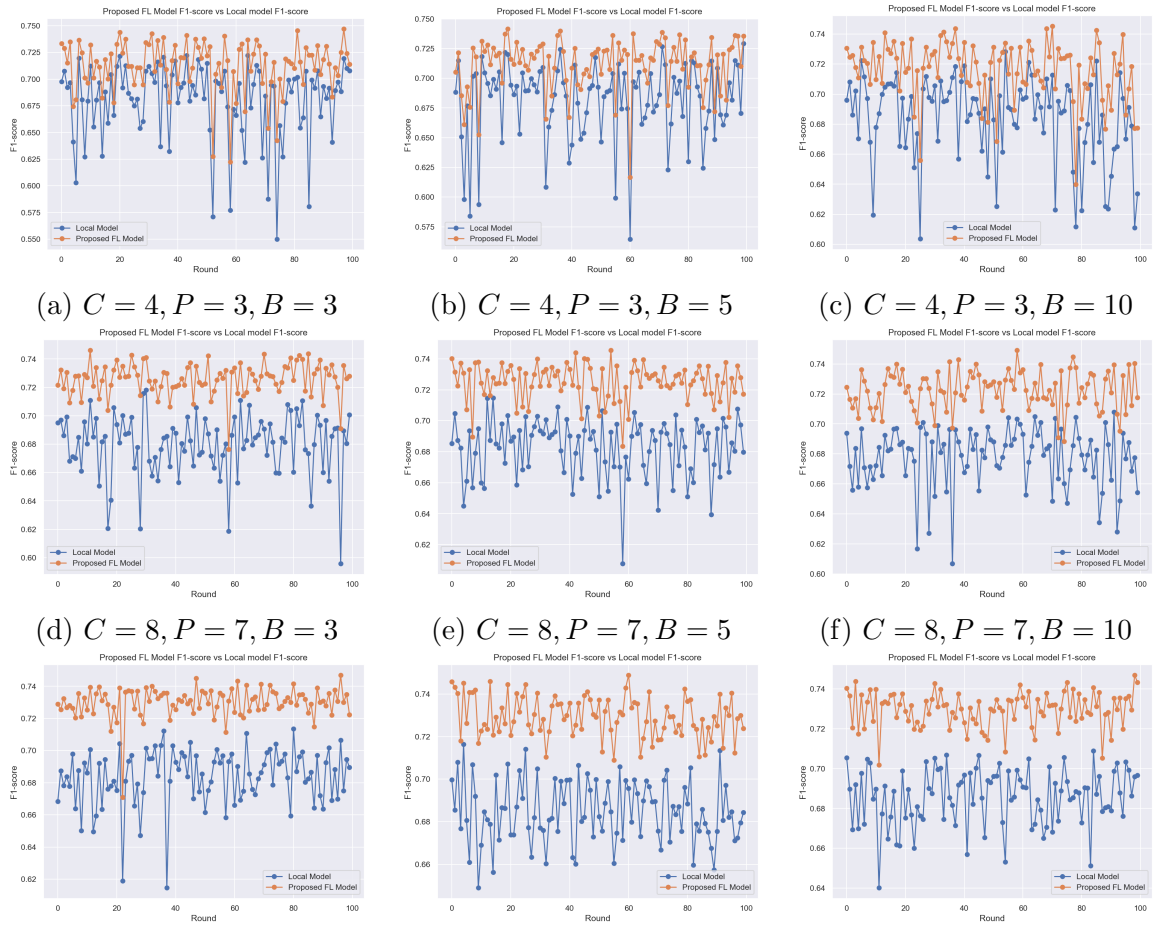


Figure 7.38: Split dataset across 12 agents and simulate performance of various number of participants with different batch sizes: (a) batch size = 0 (b) batch size = 3 (a) batch size = 5 (a) batch size = 10.

7.3.7 Experimental Results

For the convenience of our simulation, we define several agents under consideration and distribute our whole dataset among those agents. In a real-life scenario, the number of agents indicates the available interested agents within an FL network. However, we can distribute the dataset in a uniform or non-uniform fashion among the agents. If we uniformly distribute the dataset among the agents, we call it IID settings and in case we perform non-uniform distribution of the data among the agents, we call it non-IID settings.

We perform our simulation by considering both IID and non-IID FL settings. As we already discussed in Algorithm **13**, we need to select a fraction of agents who would carry out local training. We consider a variant number of agents to split our dataset and figure out the best estimator for those scenarios. In Figure **7.38**, we show the simulation results after splitting our whole dataset across 12 agents and figuring out what number of participants showing the best estimator in 50 training rounds for different batch sizes, i.e., batch size = 0, 3, 5, 10. From the simulation result, we understand that, while considering 12 agents to split our dataset, if we set batch size as 0, 3, 5, 10, then considering participants as 9, 10, 11, and 11, respectively, would give us the effective training outcome. For the IID settings, where the dataset is uniformly distributed across the agents, we applied our proposed Algorithm **13** derived from FedAvg model [MMR$^+$17] and checked out its performance with the mean of local performance of the selected participants. We considered variant number of agents to partition the dataset, selected optimal number of participants with the results of Figure **7.38**, and set different batch sizes to examine the performance. In Figure **7.39**, we can see that our proposed model outperforms the mean of the local model performance in all cases. We also store the best local model in every training round and compare that to our corresponding



(a) $C = 4, P = 3, B = 3$ (b) $C = 4, P = 3, B = 5$ (c) $C = 4, P = 3, B = 10$

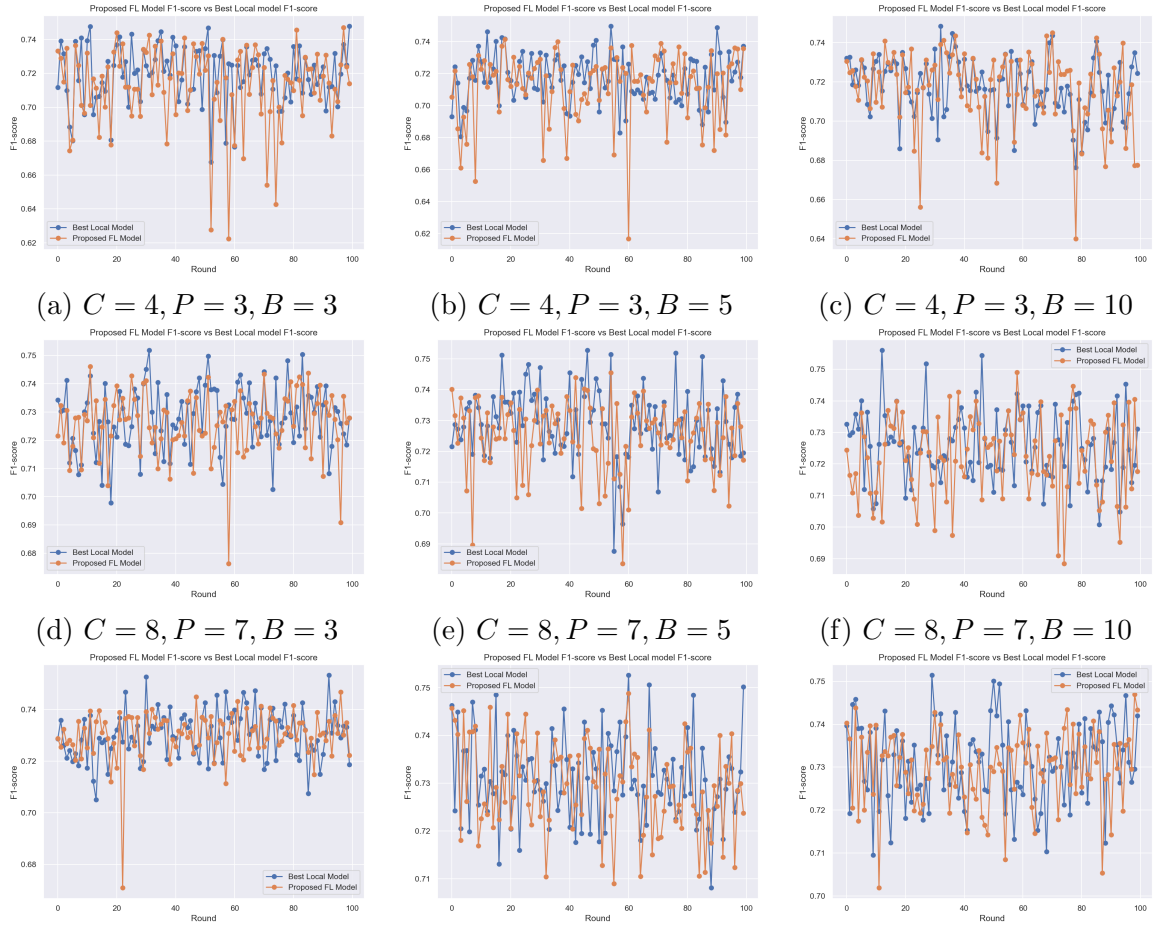
(d) $C = 8, P = 7, B = 3$ (e) $C = 8, P = 7, B = 5$ (f) $C = 8, P = 7, B = 10$

(j) $C = 12, P = 7, B = 3$ (k) $C = 12, P = 10, B = 5$ (l) $C = 12, P = 11, B = 10$

Figure 7.39: Performance comparison of our proposed FL model with variant local epochs vs local mean model of the participants (C = clients, P = participants, B = batch size).

model. We found that the proposed model has very close training accuracy compared to the best local model performance. The comparison of our proposed model with the best local model is visualized in Figure **7.40**.

We compared our result with a personalized FedAvg (Per-FedAvg) algorithm. In Per-FedAvg, the stochastic gradient for all the participated agents is computed based on their independent local dataset batches. However, if a participated agent is unable to execute the required number of local iterations considering the local gradient, then it may affect the global model accuracy. In the worst case, that



(a) $C = 4, P = 3, B = 3$ (b) $C = 4, P = 3, B = 5$ (c) $C = 4, P = 3, B = 10$

(d) $C = 8, P = 7, B = 3$ (e) $C = 8, P = 7, B = 5$ (f) $C = 8, P = 7, B = 10$

(j) $C = 12, P = 10, B = 3$ (k) $C = 12, P = 11, B = 5$ (l) $C = 12, P = 11, B = 10$

Figure 7.40: Performance comparison of our proposed FL model with variant local epochs vs best local model in every training round (C = agents, P = participants, B = batch size).

agent would not be able to send a response to the server within a preset time. If we consider that most of the participated agents in the training process cannot execute the assigned number of local epochs, then the model convergence would be prolonged. Though the setting up of local gradient accelerates the training process of the agents compared to the state-of-the-art FedAvg algorithm, it fails to address underlying issues when most of the agents have limited resources and can hardly perform an assigned computational task. Our proposed model can deal with the resource limitation issues by assigning higher computational tasks to the

resource-sufficient and reliable agents and setting lower computational tasks to the resource-constrained agents.

We also simulate our proposed FL model performance for non-IID settings by distributing data uniformly through label shewing and considering a various number of stragglers. We tested our system with various skew levels (i.e., non-IIDness), i.e., $0, 0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95$. A skew label 0.30 means the FL agent would have a minimum of 30% labels of a class. In figure **7.41- 7.43**, we compared our proposed method with the centralized model, local mean, FedAvg and Per-FedAvg method during prediction of agent disaster considering the non-settings with variant skewness and stragglers. We can see that our proposed FL model outperforms the local mean model and FedAvg model in every case because traditional local mean model and FedAvg simply drop the stragglers. Our proposed model also achieved higher F1 score than Per-FedAvg, particularly in presence of stragglers because its capability of accepting partial works from the resource-constrained agents. Specifically, Per-FedAvg sets the local starting point to reach at target convergence while our proposed method generates optimal local computational task to perform in a training round. Further, our proposed FL model performs almost similarly to the centralized model up to 70% skew label and has slow performance degradation with higher skew label. We also tested the computational time of our proposed model as well as other state-of-the-art learning methods such as decentralized technique with no model transfer, federated averaging (FedAvg) algorithm, personalized federated averaging (per-FedAvg) algorithm, activity and resource-aware FL model (FedAR), and we achieved less computation time for different FL scenarios considering a various number of agents with stragglers (see Figure **7.44**). The decentralized method resulted in highest computational time because whenever an agent became a straggler, the neighbor agent needed to

wait for the data from that agent to update its learning model. In turn, FedAvg directly drops the agents who become straggler during a training round, and both Per-FedAvg and FedAR do not accept partial works from the stragglers and carry-out training based on agent's personalization. FedAR achieves comparatively better performance than Decentralized, FedAvg, and Per-FedAvg because it has a mechanism to avoid the weak agents and not select them for training; however, it takes more computational time than our proposed method because it also needs to wait for the straggler agents if somehow an effective agent turns into straggler.

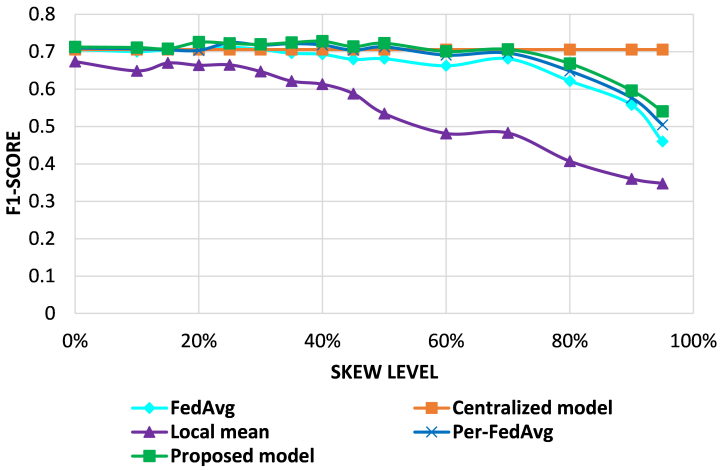


Figure 7.41: Evaluate F1-score of different learning methods in non-IID settings considering 2 stragglers out of 4 participants.

7.3.8 Discussion and Analysis

Our proposed FL model is particularly suitable for the unique task of predicting the financial distress of the customers because it maintains data privacy and leverage edge computation to generate a prediction model. As the leakage of customers' sensitive data is a major concern of the modern financial sectors, thus, applying our proposed model can cut off the tension of data leakage. This is the first FL appli-

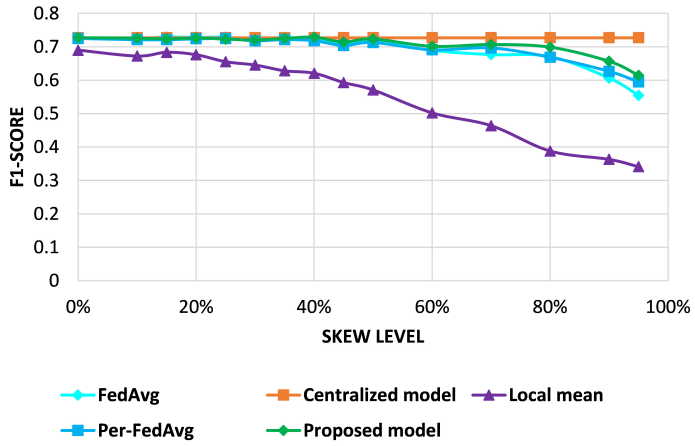


Figure 7.42: Evaluate F1-score of different learning methods in non-IID settings considering 4 stragglers out of 6 participants.

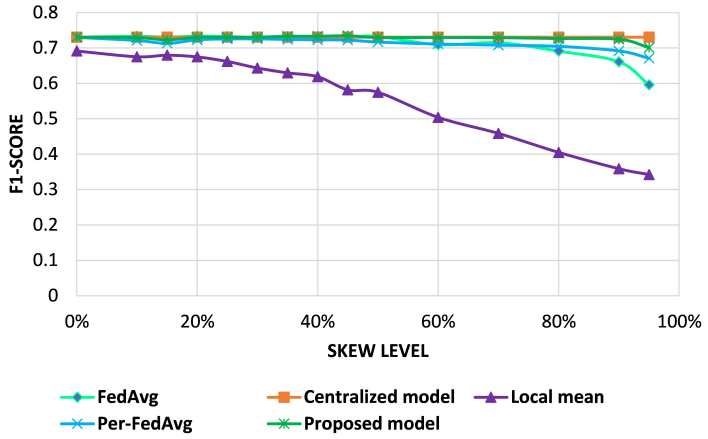
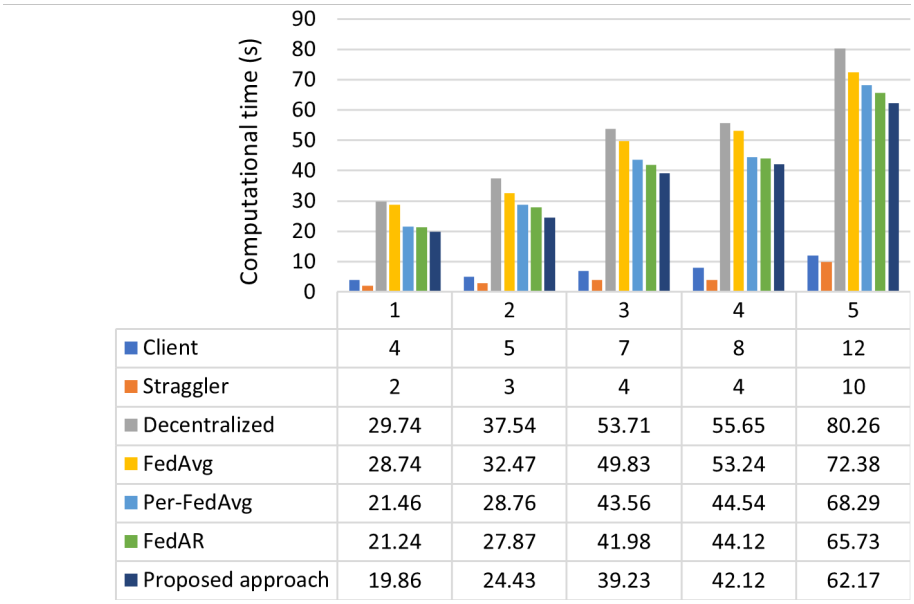


Figure 7.43: Evaluate F1-score of different learning methods in non-IID settings considering 8 stragglers out of 10 agents.

cation for predicting financial distress. To this end, we not only implemented the application using the state-of-the-art FedAvg algorithm but also propose a dynamic task allocation strategy that tailors the effective amount of computational tasks a distributed agent needs to perform. Through the distributed learning paradigm, the processing burden upon a central fusion center is eliminated and each distributed agent learns collaboratively from local data and global knowledge. Instead of assigning uniform computational tasks, the novel dynamic task allocation strategy

Table 7.3: Evaluation of our proposed model performance considering precision, recall and F1-score.

Test case	Skew Level	0%	20%	40%	60%	80%	90%	95%
2 stragglers among 4 agents	Precision	0.740	0.733	0.742	0.715	0.705	0.607	0.598
	Recall	0.686	0.711	0.688	0.687	0.634	0.585	0.492
	F1-score	0.712	0.722	0.714	0.701	0.668	0.596	0.540
4 stragglers among 6 agents	Precision	0.742	0.739	0.749	0.721	0.717	0.706	0.676
	Recall	0.713	0.717	0.693	0.705	0.679	0.613	0.562
	F1-score	0.727	0.726	0.720	0.706	0.698	0.656	0.614
8 stragglers among 10 agents	Precision	0.751	0.747	0.758	0.751	0.745	0.734	0.725
	Recall	0.710	0.711	0.711	0.708	0.707	0.714	0.678
	F1-score	0.730	0.729	0.734	0.729	0.726	0.724	0.701



	1	2	3	4	5
Client	4	5	7	8	12
Straggler	2	3	4	4	10
Decentralized	29.74	37.54	53.71	55.65	80.26
FedAvg	28.74	32.47	49.83	53.24	72.38
Per-FedAvg	21.46	28.76	43.56	44.54	68.29
FedAR	21.24	27.87	41.98	44.12	65.73
Proposed approach	19.86	24.43	39.23	42.12	62.17

Figure 7.44: Computational time simulation of our proposed Federated Learning model with other existing approaches.

helps us to tackle the issues regarding systems heterogeneity while applying the FL process.

Our study demonstrates that the heterogeneous resources of the distributed agents can significantly mitigate the overall performance of the model during the prediction of customer financial distress. Similar straggler effects were observed

in the past studies in different other settings, e.g., a recent study on collaborative mobile robots demonstrated that a higher number of stragglers lowers the model accuracy of the participated robots [IA20]. However, the proposed model can only select the effective agents for the training process but missed the underlying issues if most of the agents have resource limitations. Similarly, the authors in [VNN+21] also aimed to mitigate straggler effects by understanding its negative effect on the model convergence. They proposed an approach of selecting agents for the FL process while considering cell-free large MIMO networks. However, their proposed strategy mostly focused on network bandwidth, did not analyze the other resource status (e.g., locally available data) of the distributed agents, and failed to scale up diverse local model updates. Besides, the authors in [NY19] proposed an FL protocol by managing agents based on their resource conditions. Particularly, the protocol can resolve the agents' selection issues considering resource constraints that enable the FL server to perform an aggregation over the local model updates of as many agents as possible. However, their proposed approach does not work effectively when agents' local computation time and model update time fluctuate dynamically from agent to agent. Our proposed FL model can effectively predict financial disasters even when a large number of participated agents are stragglers due to their resource-constraint issues. We dynamically allocate local training tasks for the federated agents so that none of the agents incurs delays in sharing local model updates and thus, we can significantly reduce the computational time to reach our target accuracy.

The outcome of this research indicates that though the centralized model performs effectively in predicting financial distress, it possesses an underlying problem as it needs to collect all the sensitive information from different financial sectors. It is obvious that due to customers' privacy, a lot of companies would not be inter-

ested in avail the privilege of the prediction results. If we consider the traditional distributed ML systems as a scheme of sharing information, that also underperforms because the agents could not learn global knowledge instead only learn from their peers. After that, we check the performance of the state-of-the-art FedAvg and Per-FedAvg algorithm and observe improved performance compare to the traditional distributed learning-based systems. It is because both FedAvg and Per-FedAvg consider the global knowledge during the local model update of each agent. Between the FedAvg and Per-FedAvg algorithm, Per-Avg achieves better prediction results because it enables personalized model training of the agents. However, both FedAvg and Per-FedAvg did not quantify the task allocation during model training. As a result, when we have a resource-constrained environment and the task publisher sets up an overwhelming task for a resource-constrained agent, it fails to accomplish that task and turns into a straggler. We handle that issue by allocating local computational tasks analyzing agent available resources and enabling partial computational tasks for the resource-constrained agents. That is why our proposed model achieves higher accuracy even when we have a high number of stragglers within an FL setting because it allows partial works for the resource-constrained agents instead of dropping them from the training round that ensures inclusion of various observations of the distributed agents. Eventually, that also accelerates the convergence time because our model performs model aggregation whenever the server receives a model update from any of the agents instead of waiting for a model update from all the participated FL agents.

7.3.9 Methodological Contribution

Financial distress prediction of customers is pivotal for financial organizations who need to undertake critical decisions on loans based on the generated results from the forecasting model. Bankruptcy prediction of the customers is one of the main aspects in this regard. To construct a learning model for determining customers' financial distress, the traditional learning methods share the private information of the customers to a central fusion center. Later on, a central prediction model is developed based on the gathered data of all the customers. However, sharing the customers' private data may lead to privacy leakage, and any of the data could be modified in the middle of the data sharing process. Thus, we focus on preserving customers' data privacy while generating a prediction model. We perform on-device model training considering distributed network agents, collect the learning model from those agents, and obtain a final model that captures all the global knowledge. From the theoretical perspective, the final model can still obtain a smart model that has high accuracy and low loss because we are considering each agent's feedback on their local samples and perform weighted average on the shared model that eventually gives more importance to the agent's model that holds more data. We practically simulate the model for the financial disaster prediction model based on the state-of-the-art FedAvg algorithm and observed superior performance than the traditional decentralized model. Later on, we investigate how the FedAvg algorithm would perform if a majority of the distributed agents have resource-constrained issues, and turn into stragglers when a uniform local computational task is assigned to them. Our implementation results show that the system heterogeneity has negative effects on the global model performance, and it prolongs the model convergence if a majority of stragglers are dropped from the training process. To solve this issue, we dynamically allocate local computational tasks based on the available resources of

the distributed agents to predict customers' financial distress. The hypothesis that we considered is if we can handle the situation of turning an agent into a straggler, then we can accelerate the convergence process and can obtain a high-quality model. Particularly, an agent turns into a straggler if the assigned computational task is too overwhelming for that agent. Thus, if we enable partial computational tasks for an agent, then it can contribute towards model convergence using its limited resources. It gives us two major advantages: first, every small contribution of the agents is counted, and second, we do not have to lose the effect of any agent's data that could be crucial. By leveraging our proposed FL model, we found a significant improvement of accuracy even when a high number of agents have constrained resources. Besides, we achieved lower computational time to reach target convergence as the FL server did not have to wait for model updates from the weak agents. Thus, our theoretical and practical implications are validated.

7.3.10 Conclusion

We proposed an FL paradigm for predicting agent financial status, utilizing local computation of different information sources considered as FL agents. We enable a strategy of accepting partial works from the weak agents and demonstrate how to ensure robustness for financial distress prediction application even if we have a large number of stragglers. We simulated our prediction model in both IID and non-IID settings and considered available local data volume to determine the required number of local epoch for each FL participant. Due to variant data volume, i.e., different local epochs, we leveraged the asynchronous FL mechanism that performs global model update without waiting for all agents' local model updates. We tested the model by splitting the dataset among the variant number of agents and

simulated results for different batch sizes. By considering a different number of stragglers and setting up various computational tasks (e.g., local epoch, batch size), we demonstrated that our developed FL model is robust even if there are many straggler agents within the network. We also show a comparison graph of F1-score that proves the effectiveness of our proposed FL model outperforming the centralized model and the state-of-the-art FL model, particularly in non-IID settings. We achieved $5 \sim 6\%$ higher testing accuracy on average compared to the state-of-the-art FL models even when we have highly sparse non-IID data within the networks, and a large number of participated agents have limited resources. We also found that our proposed model achieved faster convergence compared to the state-of-the-art FL methods (e.g., 10.21s faster computational time than FedAvg when considered 10 straggler agents out of 12 agents).

CHAPTER 8

CONCLUDING REMARKS AND FUTURE WORKS

8.1 Summary

In this dissertation, we study the systems, statistical, and model heterogeneity issues while applying distributed machine learning algorithms in resource-constrained environments. In chapter 2, we conducted a comprehensive survey on Federated Learning-based Internet-of-Things (FL-IoT) settings by analyzing existing challenges, discussing potential solutions, presenting open research issues, and highlighting future research directions. We then design a distributed sensing mechanism presented in chapter 3 that can trigger any network clients remotely encountering low communication cost, and fetch necessary information for FL training. Afterward, in chapter 4, we introduce an activity and resource-aware FL model that examine the previous contributions of the clients and check out the resource status to select the effective client for FL method. In chapter 5, we develop a tri-layer FL framework that can effectively handle systems and statistical heterogeneity of the network clients in spite of the presence of the resource-constrained devices. Moving forward, we introduce a technique to handle heterogeneous local model architectures and local resources of the distributed clients via knowledge distillation and dynamic local task allocation in chapter 6. After that, in chapter 7, we present how our novel FL algorithms can be effective in real-world applications, such as improving resilience of critical infrastructures, human activity recognition, and finance.

We presented a novel approach of distributed intelligent sensing leveraging smart end-user nodes which can sense the environment and communicate with a central server by sending their asset. We proposed and implemented a solution to leverage mobile nodes for stimulating data. We further discussed the pathway to use collected

data as a facilitator towards enabling federated learning for effective communication among the sensor nodes. We then explained federated learning as a promising solution for our distributed intelligent sensing and decision making a framework to attain optimized outcomes while learning over autonomous IoT networks.

After developing the distributed sensing mechanism, we implemented an activity and resource-aware FL framework, *FedAR* to deal with the untrustworthy and resource-constrained FL environments. We constructed a trust model that labels each client with a score after monitoring their activities and previous training performance. Our proposed framework capable of avoiding the inconsistent and inadequate resource clients from the training process by checking out their resource-availability and previous participation history.

Though FedAR is capable to select proficient clients during training, it has limitations if any proficient clients somehow turns into a straggler during a training. Besides, handling statistical heterogeneity and training a large model would be ineffective for the resource-constrained devices. Thus, we develop a tri-layer FL framework, *FedPARL* where we additionally try to answer two things: first, how our system will respond if we have large data across network, and second how we can tackle a situation if majority of the available clients possess low resources and if we have no other way without selecting the resource-scarce clients for the training. So, in a nutshell, the main focus of *FedPARL* is how can we carry-out lightweight model training and count every little contributions from the resource-constrained clients. We tested our FedPARL framework with various datasets and achieved a robust, stable and consistent FL model that has remarkable performance within unreliable heterogeneous networks.

Though our *FedPARL* framework can handle systems and statistical heterogeneity, it can not handle an FL setting, where the clients possess heterogeneous

model architectures. Thus, we develop *FedMDP* that can deal with resource-constrained agents with heterogeneous local model architectures. We developed a distillation techniques that is effective even in a highly resource-constrained environment with heterogeneous resource and local model architectures. Our simulation results demonstrated that leveraging dynamic local task allocation and infusing knowledge distillation can solve the issues related to heterogeneous FL settings with non-identical local model architectures and local computational resources.

After developing various FL algorithms that can handle systems, statistical and model heterogeneity, we focus on applying our developed algorithms in solving real-world challenges. We demonstrated that our leveraged novel FL algorithm is effective in improving resilience of critical infrastructures, identifying human activities in a highly unreliable environment, and can also predict customer's financial distress when the network agents have constrained computational resources.

8.2 Future Works

We present significant future research directions that can be conducted in the current state-of-the-art distributed machine learning techniques for IoT environments:

- Our novel FL algorithms can avoid untrustworthy agents from the training process and can also handle a situation if an agent share a diverge model update. However, various types of cyber attacks could be observed in a learning environment, such as poisonous attack, sybil attack, backdoor attack. A comprehensive study needs to be conducted to detection and prevention such attacks in an FL-IoT environments.
- In our novel FL algorithms, we aim to preserve data privacy of the participated agents either by exchanging only the local model parameters or collecting the

agents' feedback as a class scores based on a public dataset. However, still there remains possibility of leaking information of the network agents and more sophisticated privacy mechanism need to be developed for the learning process.

- The state-of-the-art FL research put less focus on hyperparameters tuning optimization and feature selection. In an FL process, there are local training hyperparameters (e.g., learning rate, batch size, local epoch, dropout) on the agent side and server hyperparamaters (e.g., number of agents, timeout) on the FL server side. Thus, extensive research needs to be conducted on optimal hyperparameter tuning and removing insignificant features from agents' local dataset to enable automated FL.
- We also would like to investigate the effectiveness of various algorithms, such as YOLO and tiny YOLO, and would like to explore their impact on resource-constrained Federated Learning environments.
- In a human-centered multi-layer FL setting, the network could be interdependent. In future, we plan to develop a holistic model that can capture the interdependence among human-centered multi-layer critical infrastructures. Extensive research needs to be performed to enable interdependent decision making in a distributed fashion by developing efficient solutions that are capable of finding globally optimum solutions using information from each network as well as modeling the interdependent information exchange.

BIBLIOGRAPHY

[A+13] Davide Anguita et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[A+15] Yossi Arjevani et al. Communication complexity of distributed convex learning and optimization. In *NIPS*, 2015.

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AAS19] M Hadi Amini, Hamidreza Arasteh, and Pierluigi Siano. Sustainable smart cities through the lens of complex interdependent infrastructures: Panorama and state-of-the-art. In *Sustainable Interdependent Networks II*, pages 45–68. Springer, 2019.

[AATM18] Abdulaziz Alqahtani, Rohit Abhishek, David Tipper, and Deep Medhi. Disaster recovery power and communications for smart critical infrastructures. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[ABDR+14] Reza Arghandeh, Merwin Brown, Alberto Del Rosso, Girish Ghatikar, Emma Stewart, Ali Vojdani, and Alexandra von Meier. The local team: Leveraging distributed resources to improve resilience. *IEEE Power and Energy Magazine*, 12(5):76–83, 2014.

[ACF+17] Gianluca Aloi, Giuseppe Caliciuri, Giancarlo Fortino, Raffaele Gravina, Pasquale Pace, Wilma Russo, and Claudio Savaglio. Enabling IoT interoperability through opportunistic smartphone-based mobile gateways. *Journal of Network and Computer Applications*, 81:74–84, 2017.

[AH17] Jemal H Abawajy and Mohammad Mehedi Hassan. Federated internet of things and cloud computing pervasive patient health monitoring system. *IEEE Communications Magazine*, 55(1):48–53, 2017.

[AIM20] M. Hadi Amini, Ahmed Imteaj, and Javad Mohammadi. Distributed machine learning for resilient operation of electric systems. In *2020 International Conference on Smart Energy Systems and Technologies (SEST)*, pages 1–6. IEEE, 2020.

[AIP20] M Hadi Amini, Ahmed Imteaj, and Panos M Pardalos. Interdependent networks: A data science perspective. *Patterns*, 1(1):100003, 2020.

[AKB19] Aytaç Altan, Seçkin Karasu, and Stelios Bekiros. Digital currency forecasting with chaotic meta-heuristic bio-inspired signal processing techniques. *Chaos, Solitons & Fractals*, 126:325–336, 2019.

[AKP21] Mohammad Mohammadi Amiri, Sanjeev R Kulkarni, and H Vincent Poor. Federated learning with downlink device selection. In *2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 306–310. IEEE, 2021.

[ALBR18] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael Rabbat. Stochastic gradient push for distributed deep learning. *arXiv preprint arXiv:1811.10792*, 2018.

[Ami19] Mohammadhadi Amini. *Distributed computational methods for control and optimization of power distribution networks*. PhD thesis, Carnegie Mellon University, 2019.

[AMK19] M Hadi Amini, Javad Mohammadi, and Soummya Kar. Distributed holistic framework for smart city infrastructures: Tale of interdependent electrified transportation network and power grid. *Ieee Access*, 7:157535–157554, 2019.

[AMWK19] Sotirios A Argyroudis, Stergios A Mitoulis, Mike G Winter, and Amir M Kaynia. Fragility of transport assets exposed to multiple hazards: State-of-the-art review toward infrastructural resilience. *Reliability Engineering & System Safety*, 191:106567, 2019.

[ANH13] Mohammad Hadi Amini, Behrouz Nabi, and Mahmoud-Reza Haghifam. Load management using multi-agent systems in smart distribu-

tion network. In *2013 IEEE Power & Energy Society General Meeting*, pages 1–5. IEEE, 2013.

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[ASK19] Jin-Hyun Ahn, Osvaldo Simeone, and Joonhyuk Kang. Wireless federated distillation for distributed edge learning with heterogeneous data. In *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2019.

[ASY+18] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. cpsgd: Communication-efficient and differentially-private distributed sgd. In *Advances in Neural Information Processing Systems*, pages 7564–7575, 2018.

[ATT+20] Siavash Alemzadeh, Hesam Talebiyan, Shahriar Talebi, Leonardo Dueñas-Osorio, and Mehran Mesbahi. Resource allocation for infrastructure resilience using artificial neural networks. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 617–624. IEEE, 2020.

[AudIK+19] Muhammad Ammad-ud din, Elena Ivannikova, Suleiman A Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. Federated collaborative filtering for privacy-preserving personalized recommendation system. *arXiv preprint arXiv:1901.09888*, 2019.

[AZTS17] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[BA05] Roberto J Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *21st International conference on data engineering (ICDE'05)*, pages 217–228. IEEE, 2005.

[BAKB20] Narayan Bhusal, Michael Abdelmalak, Md Kamruzzaman, and Mohammed Benidris. Power system resilience: Current practices, challenges, and future directions. *IEEE Access*, 8:18064–18086, 2020.

[BBC+21] Emanuele Bellini, Pierfrancesco Bellini, Daniele Cenni, Paolo Nesi, Gianni Pantaleo, Irene Paoli, and Michela Paolucci. An ioe and big multimedia data approach for urban transport system resilience management in smart cities. *Sensors*, 21(2):435, 2021.

[BDF+18] 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.

[BDKD19] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[BEG+19a] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[BEG+19b] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design, 2019.

[BGIWG21] Cennet Pelin Boyacı-Gündüz, Salam A Ibrahim, Ooi Chien Wei, and Charis M Galanakis. Transformation of the food sector: Security and resilience during the covid-19 pandemic. *Foods*, 10(3):497, 2021.

[BGTT17] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. *arXiv preprint arXiv:1705.08435*, 2017.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[BPC11] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, Norwell, MA, 2011.

[BRL+21] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. Benchmarking tinyml systems: Challenges and direction, 2021.

[BSX+19] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. Flchain: A blockchain for auditable federated learning with trust and incentive. In *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 151–159. IEEE, 2019.

[BVH+18] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[BZF+21] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, 2021.

[C+19] Luca Corinzia et al. Variational federated multi-task learning. *arXiv:1906.06268*, 2019.

[Car97] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[CBM20] Wei Chen, Kartikeya Bhardwaj, and Radu Marculescu. Fedmax: mitigating activation divergence for accurate and communication-efficient federated learning. *arXiv preprint arXiv:2004.03657*, 2020.

[CBSC16] Igor Colin, Aurélien Bellet, Joseph Salmon, and Stéphan Clémençon. Gossip dual averaging for decentralized optimization of pairwise functions. *arXiv preprint arXiv:1606.02421*, 2016.

[CDS+14] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint

high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.

[CFF+19] 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.

[CFJ+19] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

[CGW+20] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[CGZH21] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning, 2021.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[CHR21] Celian Colon, Stéphane Hallegatte, and Julie Rozenberg. Criticality analysis of a country's transport network via an agent-based supply chain model. *Nature Sustainability*, 4(3):209–215, 2021.

[CKES17] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[CKMT18] 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.

[CLD+18] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning with fast convergence and efficient communication. *arXiv preprint arXiv:1802.07876*, 2018.

[CLK+18] Nicholas Carlini, Chang Liu, Jernej Kos, Úlfar Erlingsson, and Dawn Song. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*, 2018.

[CLQ+18] Heng Cai, Nina SN Lam, Yi Qiang, Lei Zou, Rachel M Correll, and Volodymyr Mihunov. A synthesis of disaster resilience measurement methods and indices. *International journal of disaster risk reduction*, 31:844–855, 2018.

[CNDK19] Catalin Capota, Moritz Neun, Lyman Do, and Michael Kopp. Asynchronous federated learning for geospatial applications. In *ECML PKDD 2018 Workshops: DMLE 2018 and IoT Stream 2018, Dublin, Ireland, September 10–14, 2018, Revised Selected Papers*, volume 967, page 21. Springer, 2019.

[CPSC20] Mingzhe Chen, H Vincent Poor, Walid Saad, and Shuguang Cui. Convergence time optimization for federated learning over wireless networks. *arXiv preprint arXiv:2001.07845*, 2020.

[CS12] Jianshu Chen and Ali H Sayed. Diffusion adaptation strategies for distributed optimization and learning over networks. *IEEE Transactions on Signal Processing*, 60(8):4289–4305, 2012.

[CS16] Janice Canedo and Anthony Skjellum. Using machine learning to secure IoT systems. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 219–222. IEEE, 2016.

[CSJ19] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[CVD19] Jean-Marie Cariolet, Marc Vuillet, and Youssef Diab. Mapping urban resilience to disasters–a review. *Sustainable cities and society*, 51:101746, 2019.

[CWL+18] Sebastian Caldas, 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.

[CXL+18] Ci Chen, Kan Xie, Frank L Lewis, Shengli Xie, and Ali Davoudi. Fully distributed resilience for adaptive exponential synchronization of het-

erogeneous multiagent systems against actuator faults. *IEEE Transactions on Automatic Control*, 64(8):3347–3354, 2018.

[CXZG16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[CY+18] Laizhong Cui, Shu Yang, et al. A survey on application of machine learning for internet of things. *J. M. L. Cybernetics*, 9(8):1399–1417, 2018.

[CYS+19] 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. *arXiv preprint arXiv:1909.07972*, 2019.

[CYS+20] 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*, 2020.

[CYW+20] Mingshu Cong, Han Yu, Xi Weng, Jiabao Qu, Yang Liu, and Siu Ming Yiu. A vcg-based fair incentive mechanism for federated learning. *arXiv preprint arXiv:2008.06680*, 2020.

[DB19] Anirban Das and Thomas Brunschwiler. Privacy is what we care about: Experimental investigation of federated learning on edge devices. *arXiv preprint arXiv:1911.04559*, 2019.

[Dev17] Android Developers. Sensormanager. *Android Developers. URL: https://developer. android. com/reference/android/hardware/SensorManager. html Accessed*, 21, 2017.

[DGBSX12] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13:165–202, 2012.

[DGL+19] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. On-device machine learning: An algorithms and learning theory perspective. *arXiv preprint arXiv:1911.00623*, 2019.

[DRSD+19] Kevin Dick, Luke Russell, Yasmina Souley Dosso, Felix Kwamena, and James R Green. Deep learning for critical infrastructure resilience. *Journal of Infrastructure Systems*, 25(2):05019003, 2019.

[DTN+19] Canh Dinh, Nguyen H Tran, Minh NH Nguyen, Choong Seon Hong, Wei Bao, Albert Zomaya, and Vincent Gramoli. Federated learning over wireless networks: Convergence analysis and resource allocation. *arXiv preprint arXiv:1910.13067*, 2019.

[DTN20] Canh T Dinh, Nguyen H Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. *arXiv preprint arXiv:2006.08848*, 2020.

[E+04] Theodoros Evgeniou et al. Regularized multi–task learning. In *ACM SIGKDD*, 2004.

[E+19] Hubert Eichner et al. Semi-cyclic stochastic gradient descent. *arXiv:1904.10120*, 2019.

[EAF+19] Thomas Elmqvist, Erik Andersson, Niki Frantzeskaki, Timon McPhearson, Per Olsson, Owen Gaffney, Kazuhiko Takeuchi, and Carl Folke. Sustainability and resilience for transformation in the urban century. *Nature sustainability*, 2(4):267–273, 2019.

[EGSVR16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59, 2016.

[EPB+19] Anis Elgabli, Jihong Park, Amrit S Bedi, Mehdi Bennis, and Vaneet Aggarwal. Gadmm: Fast and communication efficient framework for distributed machine learning. *arXiv preprint arXiv:1909.00047*, 2019.

[ES10] Iddo I Eliazar and Igor M Sokolov. Measuring statistical heterogeneity: The pietra index. *Physica A: Stat. Mech. App.*, 389(1):117–125, 2010.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[FKAE21] Yasir Ali Farrukh, Irfan Khan, Zeeshan Ahmad, and Rajvikram Madurai Elavarasan. A sequential supervised machine learning approach for cyber attack detection in a smart grid system. *arXiv preprint arXiv:2108.00476*, 2021.

[FMO20] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020.

[FRS+20] Jie Feng, Can Rong, Funing Sun, Diansheng Guo, and Yong Li. Pmf: A privacy-preserving human mobility prediction framework via federated learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(1):1–21, 2020.

[FYB18] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[GBH09] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[GCM+17] Haixiang Gao, Ying Chen, Shengwei Mei, Shaowei Huang, and Yin Xu. Resilience-oriented pre-hurricane resource allocation in distribution systems considering electric buses. *Proceedings of the IEEE*, 105(7):1214–1233, 2017.

[GCU+16] Roberto Guidotti, Hana Chmielewski, Vipin Unnikrishnan, Paolo Gardoni, Therese McAllister, and John van de Lindt. Modeling the resilience of critical infrastructure: The role of network dependencies. *Sustainable and resilient infrastructure*, 1(3-4):153–168, 2016.

[GCZ+16] Bin Guo, Chao Chen, Daqing Zhang, Zhiwen Yu, and Alvin Chin. Mobile crowd sensing and computing: when participatory sensing meets participatory social media. *IEEE Communications Magazine*, 54(2):131–137, 2016.

[GDN+21] Mohammad Ghiasi, Moslem Dehghani, Taher Niknam, Hamid Reza Baghaee, Sanjeevikumar Padmanaban, Gevork B Gharehpetian, and Hamdulah Aliev. Resiliency/cost-based optimal design of distribution network to maintain power system stability against physical attacks: A practical study case. *IEEE Access*, 9:43862–43875, 2021.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[GJ+19] Zhongshu Gu, Hani Jamjoom, et al. Reaching data confidentiality and model accountability on the caltrain. In *IEEE DSN*, 2019.

[GKN17] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[GLL20] Huayan Guo, An Liu, and Vincent KN Lau. Analog gradient aggregation for federated learning over wireless networks: Customized design and convergence analysis. *IEEE Internet of Things Journal*, 2020.

[GPV19] Badih Ghazi, Rasmus Pagh, and Ameya Velingker. Scalable and differentially private distributed aggregation in the shuffled model. *arXiv preprint arXiv:1906.08320*, 2019.

[GS+17] Chirag Gupta, Arun Sai Suggala, et al. Protonn: Compressed and accurate knn for resource-scarce devices. In *ICML*, 2017.

[GT+19] Neel Guha, Ameet Talwlkar, et al. One-shot federated learning. *arXiv:1902.11175*, 2019.

[GWW+20] Suyu Ge, Fangzhao Wu, Chuhan Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Fedner: Privacy-preserving medical named entity recognition with federated learning, 2020.

[HBK19] Akhtar Hussain, Van-Hai Bui, and Hak-Man Kim. Microgrids as a resilience resource and strategies used by microgrids for enhancing resilience. *Applied energy*, 240:56–72, 2019.

[HCGW21] Stella E Hines, Katherine H Chin, Danielle R Glick, and Emerson M Wickwire. Trends in moral injury, distress, and resilience factors among healthcare workers at the beginning of the covid-19 pandemic. *International Journal of Environmental Research and Public Health*, 18(2):488, 2021.

[HDJ21] István Hegedűs, Gábor Danner, and Márk Jelasity. Decentralized learning works: An empirical comparison of gossip learning and feder-

ated learning. *Journal of Parallel and Distributed Computing*, 148:109–124, 2021.

[HHIL+17] 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.

[HK+19] Farzin Haddadpour, Mohammad Mahdi Kamani, et al. Trading redundancy for communication: Speeding up distributed sgd for non-convex optimization. In *ICML*, 2019.

[HKMC19] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck Cadambe. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[HLL+19] 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*, 2019.

[HLN+07] Wenbo He, Xue Liu, Hoang Nguyen, Klara Nahrstedt, and Tarek Abdelzaher. Pda: Privacy-preserving data aggregation in wireless sensor networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 2045–2053. IEEE, 2007.

[HM19] Farzin Haddadpour and Mehrdad Mahdavi. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*, 2019.

[HMGS20] Rajesh Hegde, Kalpan Mukherjee, and Thribhuvan Gupta S. Electromaps. `https://github.com/sabm0hmayahai/Electro-Maps`, 2020.

[HPS+15] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using internet-of-things (IoT) sensing with cloud-based processing: Opportunities and chal-

lenges. In *2015 IEEE International Conference on Services Computing*, pages 285–292. IEEE, 2015.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in neural information processing systems*. Curran Associates, Inc., 2015.

[HRM+18] 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.

[HRM+21] Eklas Hossain, Shidhartho Roy, Naeem Mohammad, Nafiu Nawar, and Debopriya Roy Dipta. Metrics and enhancement strategies for grid resilience and reliability during natural disasters. *Applied Energy*, 290:116709, 2021.

[HSK+19] Florian Hartmann, Sunah Suh, Arkadiusz Komarzewski, Tim D. Smith, and Ilana Segall. Federated learning for ranking browser history suggestions, 2019.

[Hus13] Asim Hussain. A day without power: Outage costs for businesses. `https://www.bloomenergy.com/blog/a-day-without-power-outage-costs-businesses`, 2013.

[HWL20] Pengchao Han, Shiqiang Wang, and Kin K Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *IEEE ICDCS*, 2020.

[HY+18] Li Huang, Yifeng Yin, et al. Loadaboost: Loss-based adaboost federated machine learning on medical data. *arXiv:1811.12629*, 2018.

[HYF+20] Li Huang, Yifeng Yin, Zeng Fu, Shifa Zhang, Hao Deng, and Dianbo Liu. Loadaboost: Loss-based adaboost federated machine learning with reduced computational complexity on iid and non-iid intensive care data. *Plos one*, 15(4):e0230706, 2020.

[HZSK19] Sukjong Ha, Jingjing Zhang, Osvaldo Simeone, and Joonhyuk Kang. Coded federated computing in wireless networks with straggling de-

vices and imperfect csi. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2649–2653. IEEE, 2019.

[IA19] Ahmed Imteaj and M Hadi Amini. Distributed sensing using smart end-user devices: pathway to federated learning for autonomous IoT. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1156–1161. IEEE, 2019.

[IA20] Ahmed Imteaj and M Hadi Amini. FedAR: Activity and resource-aware federated learning model for distributed mobile robots. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1153–1160. IEEE, 2020.

[IA21] Ahmed Imteaj and M Hadi Amini. FedPARL: Client activity and resource-oriented lightweight federated learning model for resource-constrained heterogeneous IoT environment. *Frontiers in Communications and Networks*, 2:10, 2021.

[IA22] Ahmed Imteaj and M Hadi Amini. Leveraging asynchronous federated learning to predict customers financial distress. *Intelligent Systems with Applications*, page 200064, 2022.

[IAA21] Ahmed Imteaj, Raghad Alabagi, and M Hadi Amini. Exploiting federated learning technique to recognize human activities in resource-constrained environment. In *International Conference on Intelligent Human Computer Interaction*, pages 659–672. Springer, 2021.

[IAM20] Ahmed Imteaj, M Hadi Amini, and Javad Mohammadi. Leveraging decentralized artificial intelligence to enhance resilience of energy networks. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[IKKA21] Ahmed Imteaj, Irfan Khan, Javad Khazaei, and Mohammad Hadi Amini. FedResilience: A federated learning application to improve resilience of resource-constrained critical infrastructures. *Electronics*, 10(16):1917, 2021.

[Imt20a] Ahmed Imteaj. Distributed machine learning for collaborative mobile robots: Phd forum abstract. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 798–799, 2020.

[Imt20b] Ahmed Imteaj. *Distributed Machine Learning for Collaborative Mobile Robots: PhD Forum Abstract*, page 798–799. Association for Computing Machinery, New York, NY, USA, 2020.

[ISK+18] Mohsen Imani, Mohammad Samragh, Yeseong Kim, Saransh Gupta, Farinaz Koushanfar, and Tajana Rosing. Rapidnn: In-memory deep neural network acceleration framework, 2018.

[ISW19] Silvana Ilgen, Frans Sengers, and Arjan Wardekker. City-to-city learning for urban resilience: the case of water squares in rotterdam and mexico city. *Water*, 11(5):983, 2019.

[ITW+20] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. Federated learning for resource-constrained IoT devices: Panoramas and state-of-the-art, 2020.

[ITW+22] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M Hadi Amini. A survey on federated learning for resource-constrained IoT devices. *IEEE Internet of Things Journal*, 9(1):1–24, 2022.

[Jan21] C-l Jan. Financial information asymmetry: Using deep learning algorithms to predict financial distress. symmetry 2021, 13, 443, 2021.

[JF03] Radu Jurca and Boi Faltings. An incentive compatible reputation mechanism. In *EEE International Conference on E-Commerce, 2003. CEC 2003.*, pages 285–292. IEEE, 2003.

[JJN+20] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 497–511, 2020.

[JJW15] Stewart Jones, David Johnstone, and Roy Wilson. An empirical evaluation of the performance of binary classifiers in the prediction of credit ratings changes. *Journal of Banking & Finance*, 56:72–85, 2015.

[JMC+21] Jinha Jung, Murilo Maeda, Anjin Chang, Mahendra Bhandari, Akash Ashapure, and Juan Landivar-Bowles. The potential of remote sensing and artificial intelligence as tools to improve the resilience of agriculture production systems. *Current Opinion in Biotechnology*, 70:15–22, 2021.

[JOK+18] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*, 2018.

[JST+14] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in neural information processing systems*, pages 3068–3076, 2014.

[JWK+19] Yuang Jiang, Shiqiang Wang, Bong Jun Ko, Wei-Han Lee, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326*, 2019.

[JZ13] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[KABA20] Seçkin Karasu, Aytaç Altan, Stelios Bekiros, and Wasim Ahmad. A new forecasting model with wrapper-based feature selection approach using multi-objective optimization technique for chaotic crude oil time series. *Energy*, 212:118750, 2020.

[KASH18] Seçkin Karasu, Aytaç Altan, Zehra Saraç, and Rifat Hacioğlu. Prediction of bitcoin prices with machine learning methods using time series data. In *2018 26th signal processing and communications applications conference (SIU)*, pages 1–4. IEEE, 2018.

[KFBT19] Mikhail Khodak, Maria Florina-Balcan, and Ameet Talwalkar. Adaptive gradient-based meta-learning methods. *arXiv preprint arXiv:1906.02717*, 2019.

[KG+17] Ashish Kumar, Saurabh Goyal, et al. Resource-efficient machine learning in 2 kb ram for the internet of things. In *ICML*, 2017.

[KKC+16] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 380–392, 2016.

[KM+21] Peter Kairouz, H Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[KMR15] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[KMR19a] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Better communication complexity for local sgd. *arXiv preprint arXiv:1909.04746*, 2019.

[KMR19b] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. First analysis of local gd on heterogeneous data, 2019.

[KMY+16] 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.

[KOY+19] Skanda Koppula, Lois Orosa, A Giray Yağlıkçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. Eden: Enabling energy-efficient, high-performance deep neural network inference using approximate dram. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 166–181, 2019.

[KP+19] Hyesung Kim, Jihong Park, et al. Blockchained on-device federated learning. *IEEE Communications Letters*, 2019.

[KPT+20] Latif U Khan, Shashi Raj Pandey, Nguyen H Tran, Walid Saad, Zhu Han, Minh NH Nguyen, and Choong Seon Hong. Federated learning for edge networks: Resource optimization and incentive mechanism. *IEEE Communications Magazine*, 58(10):88–93, 2020.

[KR13] Jakub Konečnỳ and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666*, 2013.

[KSH+21] Latif U Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Communications Surveys & Tutorials*, 2021.

[KSJ19] Anastasia Koloskova, Sebastian U Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. *arXiv preprint arXiv:1902.00340*, 2019.

[KTD+13] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *Cidr*, volume 1, pages 2–1, 2013.

[Kus20] Andrew Kusiak. Open manufacturing: a design-for-resilience approach. *International Journal of Production Research*, 58(15):4647–4658, 2020.

[KXN+20] 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.

[LASY14] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[LAT18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[LCK19] Eui Hoon Lee, Young Hwan Choi, and Joong Hoon Kim. Real-time integrated operation for urban streams with centralized and decentralized reservoirs to improve system resilience. *Water*, 11(1):69, 2019.

[LCL+19] 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.

[LCY18] Yang Liu, Tianjian Chen, and Qiang Yang. Secure federated transfer learning. *arXiv preprint arXiv:1812.03337*, 2018.

[LDT+16] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen. Cambricon: An instruction set architecture for neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 393–405, 2016.

[LeC98] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun. com/exdb/mnist/*, 1998.

[Lee07] Young-Chan Lee. Application of support vector machines to corporate credit rating prediction. *Expert Systems with Applications*, 33(1):67–74, 2007.

[LFTL20] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[LGN+20] Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M Shamim Hossain. Deep anomaly detection for time-series data in industrial IoT : A communication-efficient on-device federated learning approach. *IEEE Internet of Things Journal*, 2020.

[LHD+19a] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Transactions on Industrial Informatics*, 16(6):4177–4186, 2019.

[LHD+19b] 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*, 2019.

[LHZ+20] Yunlong Lu, Xiaohong Huang, Ke Zhang, Sabita Maharjan, and Yan Zhang. Communication-efficient federated learning and permissioned blockchain for digital twin edge networks. *IEEE Internet of Things Journal*, 2020.

[LKCT19] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. *arXiv preprint arXiv:1909.05830*, 2019.

[LKJK19] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173*, 2019.

[LKSJ20] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *arXiv preprint arXiv:2006.07242*, 2020.

[LKZ+19] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. A communication efficient vertical federated learning framework. *arXiv preprint arXiv:1912.11187*, 2019.

[LLH+19] 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. *arXiv preprint arXiv:1909.11875*, 2019.

[LLH+20] 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*, 2020.

[LLL11] Author1 LastName1, Author2 LastName2, and Author3 LastName3. Data title, 2011.

[LMX+19] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M Jorge Cardoso, et al. Privacy-preserving federated brain tumour segmentation. In *International Workshop on Machine Learning in Medical Imaging*, pages 133–141. Springer, 2019.

[LNL+20] Yi Liu, Jiangtian Nie, Xuandi Li, Syed Hassan Ahmed, Wei Yang Bryan Lim, and Chunyan Miao. Federated learning in the sky: Aerial-ground air quality sensing framework with uav swarms. *IEEE Internet of Things Journal*, 2020.

[LO+18] He Li, Kaoru Ota, et al. Learning IoT in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[LPH17] Sulin Liu, Sinno Jialin Pan, and Qirong Ho. Distributed multi-task relationship learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 937–946. ACM, 2017.

[LR20] Zhize Li and Peter Richtárik. A unified analysis of stochastic gradient methods for nonconvex federated optimization. *arXiv preprint arXiv:2006.07013*, 2020.

[LSC18] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus, 2018.

[LSS19] Tian Li, Maziar Sanjabi, and Virginia Smith. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019.

[LSTS20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[LSZ+18] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2018.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[LW19] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

[LW20] Yang Liu and Jiaheng Wei. Incentives for federated learning: a hypothesis elicitation approach. *arXiv preprint arXiv:2007.10596*, 2020.

[LWH19] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated gradient boosting decision trees. *arXiv preprint arXiv:1911.04206*, 2019.

[LWL19] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems. *IEEE Robotics and Automation Letters*, 4(4):4555–4562, 2019.

[LXW20] Lingjuan Lyu, Xinyi Xu, and Qian Wang. Collaborative fairness in federated learning. *arXiv preprint arXiv:2008.12161*, 2020.

[LYRZ19] Yanan Li, Shusen Yang, Xuebin Ren, and Cong Zhao. Asynchronous federated learning with differential privacy for edge intelligence. *arXiv preprint arXiv:1912.07902*, 2019.

[LYY20] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

[LZJ+20] Yong Li, Yipeng Zhou, Alireza Jolfaei, Dongjin Yu, Gaochao Xu, and Xi Zheng. Privacy-preserving federated learning framework based on chained secure multi-party computing. *IEEE Internet of Things Journal*, 2020.

[LZSL19] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. Edge-assisted hierarchical federated learning with non-iid data. *arXiv preprint arXiv:1905.06641*, 2019.

[LZZ+17] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

[LZZL17] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

[MB17] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[MBS+17] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In *Proceedings of the symposium on applied computing*, pages 506–509. ACM, 2017.

[MDB17] Axel Moinet, Benoît Darties, and Jean-Luc Baril. Blockchain based trust & authentication for decentralized sensor networks. *arXiv preprint arXiv:1706.01730*, 2017.

[MGA+21] Dillip Kumar Mishra, Mojtaba Jabbari Ghadi, Ali Azizivahed, Li Li, and Jiangfeng Zhang. A review on resilience studies in active distribution systems. *Renewable and Sustainable Energy Reviews*, 135:110201, 2021.

[MKG+20] Grigory Malinovsky, Dmitry Kovalev, Elnur Gasanov, Laurent Condat, and Peter Richtarik. From local sgd to local fixed point methods for federated learning. *arXiv preprint arXiv:2004.01442*, 2020.

[MKJ+17] Chenxin Ma, Jakub Konečnỳ, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848, 2017.

[MKP+21] 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.

[MLD+20] 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.

[MMR+17] 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*, pages 1273–1282, 2017.

[MMRyA16] H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

[MMZG21] Yuxi Mi, Yutong Mu, Shuigeng Zhou, and Jihong Guan. Fedmdr: Federated model distillation with robust aggregation. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 18–32. Springer, 2021.

[MRTZ17] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[MSDCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[MSS19] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. *arXiv preprint arXiv:1902.00146*, 2019.

[MYZ+17] 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.

[NDR19] Solmaz Niknam, Harpreet S Dhillon, and Jeffery H Reed. Federated learning for wireless communications: Motivation, opportunities and challenges. *arXiv preprint arXiv:1908.06847*, 2019.

[NM21] Vahid M Nik and Amin Moazami. Using collective intelligence to enhance demand flexibility and climate resilience in urban areas. *Applied Energy*, 281:116106, 2021.

[NMM+19] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N Asokan, and Ahmad-Reza Sadeghi. Dïot: A federated self-learning anomaly detection system for IoT. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 756–767. IEEE, 2019.

[NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, 2019.

[NSH+20] Hung T Nguyen, Vikash Sehwag, Seyyedali Hosseinalipour, Christopher G Brinton, Mung Chiang, and H Vincent Poor. Fast-convergent federated learning. *arXiv preprint arXiv:2007.13137*, 2020.

[NSM20] Takayuki Nishio, Ryoichi Shinkuma, and Narayan B Mandayam. Estimation of individual device contributions for incentivizing federated learning. *arXiv preprint arXiv:2009.09371*, 2020.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE, 2013.

[NY19] 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.

[oST] National Institute of Standards and Technology. Emnist dataset.

[OXZ+21] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 54–66, 2021.

[PR19] Lea Petrella and Valentina Raponi. Joint estimation of conditional quantiles in multivariate linear regression models with an application to financial distress. *Journal of Multivariate Analysis*, 173:70–84, 2019.

[Pur18] Christer Pursiainen. Critical infrastructure resilience: A nordic model in the making? *International journal of disaster risk reduction*, 27:632–641, 2018.

[PW20] Reese Pathak and Martin J Wainwright. Fedsplit: An algorithmic framework for fast federated optimization. *arXiv preprint arXiv:2005.05238*, 2020.

[PWE+19] Jihong Park, Shiqiang Wang, Anis Elgabli, Seungeun Oh, Eunjeong Jeong, Han Cha, Hyesung Kim, Seong-Lyun Kim, and Mehdi Bennis. Distilling on-device intelligence at the network edge. *arXiv preprint arXiv:1908.05895*, 2019.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[PYHH21] Shouzheng Pan, Hai Yan, Jia He, and Zhengbing He. Vulnerability and resilience of transportation systems: A recent literature review. *Physica A: Statistical Mechanics and its Applications*, page 126235, 2021.

[QGL+20] Youyang Qu, Longxiang Gao, Tom H Luan, Yong Xiang, Shui Yu, Bai Li, and Gavin Zheng. Decentralized privacy using blockchain-enabled federated learning in fog computing. *IEEE Internet of Things Journal*, 7(6):5171–5183, 2020.

[QWW+20] Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie. Fedrec: Privacy-preserving news recommendation with federated learning, 2020.

[RCZ+20] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[RMRB19] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard, 2019.

[RSB+21] Anja Rieckert, Ewoud Schuit, Nienke Bleijenberg, Debbie Ten Cate, Wendela de Lange, Janneke M de Man-van Ginkel, Elke Mathijssen, Linda C Smit, Dewi Stalpers, Lisette Schoonhoven, et al. How can we build and maintain the resilience of our health care professionals during covid-19? recommendations based on a scoping review. *BMJ open*, 11(1):e043718, 2021.

[RT16] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.

[RTD+18] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[RWT+18] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[SAL+19] Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. Low-memory neural network training: A technical report. *CoRR*, abs/1904.10631, 2019.

[SBB+17] Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, and et al. The arm scalable vector extension. *IEEE Micro*, 37(2):26–39, Mar 2017.

[SCST17] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[SE19] Yunus Sarikaya and Ozgur Ercetin. Motivating workers in federated learning: A stackelberg game perspective. *IEEE Networking Letters*, 2019.

[SEC13] Frost & Sullivanl S&C Electric Company. S&c's 2018 state of commercial & industrial power reliability report. `https://www.sandc.com/globalassets/sac-electric/documents/sharepoint/documents---all-documents/technical-paper-100-t120.pdf`, 2013.

[SFYB18] Muhammad Shayan, Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Biscotti: A ledger for private and secure peer-to-peer machine learning. *arXiv preprint arXiv:1811.09904*, 2018.

[SG15] Andrey Somov and Raffaele Giaffreda. Powering IoT devices: Technologies and opportunities. *IEEE IoT Newsletter*, 2015.

[SJ+18] Michael R Sprague, Amir Jalalirad, et al. Asynchronous federated learning for geospatial applications. In *ECML-PKDD*, 2018.

[SLS+18] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 3, 2018.

[SLY+20] Haifeng Sun, Shiqi Li, F Richard Yu, Qi Qi, Jingyu Wang, and Jianxin Liao. Toward communication-efficient federated learning in the internet of things with edge computing. *IEEE Internet of Things Journal*, 7(11):11053–11067, 2020.

[SMBJ09] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean-Marc Jézéquel. Meta-model pruning. In *International Conference on Model Driven Engineering Languages and Systems*, pages 32–46, Berlin, Heidelberg, 2009. Springer.

[SMGK18] Ananda Samajdar, Parth Mannan, Kartikay Garg, and Tushar Krishna. Genesys: Enabling continuous learning through neural network evolution in hardware, 2018.

[SPP+17] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A survey of neuromorphic computing and neural networks in hardware, 2017.

[SSCE21] Tomer Sery, Nir Shlezinger, Kobi Cohen, and Yonina C Eldar. Over-the-air federated learning from heterogeneous data. *IEEE Transactions on Signal Processing*, 2021.

[SSZ14] 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, 2014.

[ST19] Paige Marta Skiba and Jeremy Tobacman. Do payday loans cause bankruptcy? *The Journal of Law and Economics*, 62(3):485–519, 2019.

[Sti18] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[Sto16] Bill Stonehem. *Google Android Firebase: Learning the Basics*, volume 1. First Rank Publishing, 2016.

[SVG18] Konstantin Sozinov, Vladimir Vlassov, and Sarunas Girdzijauskas. Human activity recognition using federated learning. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 1103–1111. IEEE, 2018.

[SVK18] A Sujil, Jatin Verma, and Rajesh Kumar. Multi agent system: concepts, platforms and applications in power systems. *Artificial Intelligence Review*, 49(2):153–182, 2018.

[SVM20] V Salehi, B Veitch, and M Musharraf. Measuring and improving adaptive capacity in resilient systems by means of an integrated deamachine learning approach. *Applied ergonomics*, 82:102975, 2020.

[SW18] Noviyanti Santoso and Wahyu Wibowo. Financial distress prediction using linear discriminant analysis and support vector machine. In *Jour-*

nal of Physics: Conference Series, volume 979, page 012089. IOP Publishing, 2018.

[SWJ+21] Yunias Setiawati, Joni Wahyuhadi, Florentina Joestandari, Margarita M Maramis, and Atika Atika. Anxiety and resilience of healthcare workers during covid-19 pandemic in indonesia. *Journal of Multidisciplinary Healthcare*, 14:1, 2021.

[SWMS19] 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*, 2019.

[SWS20] Felix Sattler, Thomas Wiegand, and Wojciech Samek. Trends and advancements in deep neural network communication. *arXiv preprint arXiv:2003.03320*, 2020.

[SZG19] Yuxuan Sun, Sheng Zhou, and Deniz Gündüz. Energy-aware analog aggregation for federated learning with redundant data. *arXiv preprint arXiv:1911.00188*, 2019.

[SZN+20] Wenqi Shi, Sheng Zhou, Zhisheng Niu, Miao Jiang, and Lu Geng. Joint device scheduling and resource allocation for latency constrained wireless federated learning. *IEEE Transactions on Wireless Communications*, 20(1):453–467, 2020.

[T+12] Sebastian Thrun et al. *Learning to learn*. Springer Science & Business Media, 2012.

[TAM19] Om Thakkar, Galen Andrew, and H Brendan McMahan. Differentially private learning with adaptive clipping. *arXiv preprint arXiv:1905.03871*, 2019.

[TB+19] Urmish Thakker, Jesse Beu, et al. Compressing rnns for IoT devices by 15-38x using kronecker products. *arXiv preprint arXiv:1906.02876*, 2019.

[TBG+19] Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. Run-time efficient rnn compression for inference on edge devices, 2019.

[TBZ+19] Nguyen H Tran, Wei Bao, Albert Zomaya, Nguyen Minh NH, 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.

[TDBM19] Urmish Thakker, Ganesh Dasika, Jesse Beu, and Matthew Mattina. Measuring scheduling efficiency of rnns for nlp applications, 2019.

[TLR12] Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *2012 50th annual allerton conference on communication, control, and computing (allerton)*, pages 1543–1550, Monticello, IL, USA, 2012. IEEE.

[TLY+18] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. d^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856, 2018.

[TLZY20] Ben Tan, Bo Liu, Vincent Zheng, and Qiang Yang. A federated recommender system for online services. In *Fourteenth ACM Conference on Recommender Systems*, pages 579–581, 2020.

[TRRK20] Germanno Teles, Joel JPC Rodrigues, Ricardo AL Rabê, and Sergei A Kozlov. Artificial neural network and bayesian network models for credit risk prediction. *Journal of Artificial Intelligence and Systems*, 2:118–132, 2020.

[TWK+20] Tiffany Tuor, Shiqiang Wang, Bong Jun Ko, Changchang Liu, and Kin K Leung. Overcoming noisy and irrelevant data in federated learning. In *25th International Conference on Pattern Recognition (ICPR)*, 2020. Accepted.

[TZC+21] Shunpu Tang, Wenqi Zhou, Lunyuan Chen, Lijia Lai, Junjuan Xia, and Liseng Fan. Battery-constrained federated edge learning in uav-enabled IoT for b5g/6g networks. *arXiv preprint arXiv:2101.12472*, 2021.

[UHZN17] Ehsan Ul Hassan, Zaemah Zainuddin, and Sabariah Nordin. A review of financial distress prediction models: logistic regression and multivariate discriminant analysis. *Indian-Pacific Journal of Accounting and Finance*, 1(3):13–23, 2017.

[UYO+19] Akane Uemichi, Masaaki Yagi, Ryo Oikawa, Yudai Yamasaki, and Shigehiko Kaneko. Multi-objective optimization to determine installation capacity of distributed power generation equipment considering energy-resilience against disasters. *Energy Procedia*, 158:6538–6543, 2019.

[VBT17] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*, pages 509–517. PMLR, 2017.

[VD02] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[VNN+21] Tung T. Vu, Duy T. Ngo, Hien Quoc Ngo, Minh N. Dao, Nguyen H. Tran, and Richard H. Middleton. Straggler effect mitigation for federated learning in cell-free massive mimo. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[VTWA18] Sahar Voghoei, Navid Hashemi Tonekaboni, Jason G Wallace, and Hamid R Arabnia. Deep learning at the edge. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 895–901. IEEE, 2018.

[VYJ08] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. Privacy-preserving svm classification. *Knowledge and Information Systems*, 14(2):161–178, 2008.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WCH+19] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[WeB18] WeBank. Fate: An industrial grade federated learning framework, 2018.

[WHL+19] Wentai Wu, Ligang He, Weiwei Lin, Stephen Jarvis, et al. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *arXiv preprint arXiv:1910.01355*, 2019.

[WHW+19] 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.

[WLCS18] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[WPS+20] Blake Woodworth, Kumar Kshitij Patel, Sebastian U Stich, Zhen Dai, Brian Bullins, H Brendan McMahan, Ohad Shamir, and Nathan Srebro. Is local sgd better than minibatch sgd? *arXiv preprint arXiv:2002.07839*, 2020.

[WRS20] Yi Wang, Anastasios Oulis Rousis, and Goran Strbac. On microgrids and resilience: A comprehensive review on modeling and operational strategies. *Renewable and Sustainable Energy Reviews*, 134:110313, 2020.

[WRXM18] Shusen Wang, Fred Roosta, Peng Xu, and Michael W Mahoney. Giant: Globally improved approximate newton method for distributed optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[WTS+19] 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.

[WYS+20] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*, 2020.

[WZRB+19] Jing Wang, Wangda Zuo, Landolf Rhode-Barbarigos, Xing Lu, Jianhui Wang, and Yanling Lin. Literature review on modeling and simulation of energy infrastructures from a resilience perspective. *Reliability Engineering & System Safety*, 183:360–373, 2019.

[WZZ+17] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of

computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.

[XCL20] Ronghua Xu, Yu Chen, and Jian Li. MicroFL: A lightweight, secure-by-design edge network fabric for decentralized IoT systems. In *NDSS*, 2020.

[XKG19] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[XWC11] Ru Xue, Liang Wang, and Jie Chen. Using the IoT to construct ubiquitous learning environment. In *2011 Second International Conference on Mechanic Automation and Control Engineering*, pages 7878–7880. IEEE, 2011.

[XWMM13] Tengke Xiong, Shengrui Wang, André Mayers, and Ernest Monga. Personal bankruptcy prediction by mining credit card data. *Expert systems with applications*, 40(2):665–676, 2013.

[XY+19] Zirui Xu, Zhao Yang, et al. Elfish: Resource-aware federated learning on heterogeneous edge devices. *arXiv:1912.01684*, 2019.

[YA+18] Timothy Yang, Galen Andrew, et al. Applied federated learning: Improving google keyboard query suggestions. *arXiv:1812.02903*, 2018.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[YCS+19] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. Energy efficient federated learning over wireless communication networks. *arXiv preprint arXiv:1911.02417*, 2019.

[YCS+20] 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*, 2020.

[YGX20] Binhang Yuan, Song Ge, and Wenhui Xing. A federated learning framework for healthcare devices. *arXiv preprint arXiv:2005.05083*, 2020.

[YHZ+17] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th international conference on world wide web*, pages 351–360, 2017.

[YJSD20] 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.

[YL19] Peihua Yu and Yunfeng Liu. Federated object detection: Optimizing object detection model with federated learning. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, pages 1–6, 2019.

[YLCT19a] 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):12, 2019.

[YLCT19b] 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.

[YLL+20a] 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.

[YLL+20b] Han Yu, Zelei Liu, Yang Liu, Tianjian Chen, Mingshu Cong, Xi Weng, Dusit Niyato, and Qiang Yang. A sustainable incentive scheme for federated learning. *IEEE Intelligent Systems*, 2020.

[YLS+20] Tianlong Yu, Tian Li, Yuqiong Sun, Susanta Nanda, Virginia Smith, Vyas Sekar, and Srinivasan Seshan. Learning context-aware policies from multiple smart homes via federated multi-task learning. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoT DI)*, pages 104–115. IEEE, 2020.

[YYPH20] 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.

[YYS16] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. Survey of real-time processing technologies of IoT data streams. *Journal of Information Processing*, 24(2):195–202, 2016.

[Z+19] Zhi Zhou et al. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. of the IEEE*, 107(8):1738–1762, 2019.

[ZDW13] 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.

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[ZGH+22] Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and A Salman Avestimehr. Federated learning for the internet of things: Applications, challenges, and opportunities. *IEEE Internet of Things Magazine*, 5(1):24–29, 2022.

[ZHS+20] Yuchen Zhao, Hamed Haddadi, Severin Skillman, Shirin Enshaeifar, and Payam Barnaghi. Privacy-preserving activity and health monitoring on databox. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pages 49–54, 2020.

[ZLG+18] Zhenyu Zhou, Haijun Liao, Bo Gu, Kazi Mohammed Saidul Huq, Shahid Mumtaz, and Jonathan Rodriguez. Robust mobile crowd sensing: When deep learning meets edge computing. *IEEE Network*, 32(4):54–60, 2018.

[ZLG20] 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.

[ZLH+20] Yan Zhang, Yunlong Lu, Xiaohong Huang, Ke Zhang, and Sabita Maharjan. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Transactions on Vehicular Technology*, 2020.

[ZLL+18] 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.

[ZLL+20] Yuchen Zhao, Hanyang Liu, Honglin Li, Payam Barnaghi, and Hamed Haddadi. Semi-supervised federated learning for activity recognition. *arXiv preprint arXiv:2011.00851*, 2020.

[ZLQ+20] Yufeng Zhan, Peng Li, Zhihao Qu, Deze Zeng, and Song Guo. A learning-based incentive mechanism for federated learning. *IEEE Internet of Things Journal*, 7(7):6360–6368, 2020.

[ZLS09] Martin Zinkevich, John Langford, and Alex J Smola. Slow learners are fast. In *Advances in neural information processing systems*, pages 2331–2339, 2009.

[ZMR21] Andy Zhou, Rikky Muller, and Jan Rabaey. Memory-efficient, limb position-aware hand gesture recognition using hyperdimensional computing, 2021.

[ZN+15] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184. IEEE, 2015.

[ZX15] Yuchen Zhang and Lin Xiao. Communication-efficient distributed optimization of self-concordant empirical loss, 2015.

[ZXB+21] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[ZXLJ21] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *arXiv preprint arXiv:2106.06843*, 2021.

[ZZY+20] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. Local differential privacy based federated learning for internet of things. *IEEE Internet of Things Journal*, 2020.

VITA

AHMED IMTEAJ

Born	Chattogram, Bangladesh
2015	B.S., Computer Science and Engineering Chittagong University of Engineering and Technology, Chattogram, Bangladesh
2021	M.S., Computer Science Florida International University Miami, Florida, USA
2022	Doctoral Candidate, Computer Science Florida International University Miami, Florida, USA

PUBLICATIONS AND PRESENTATIONS

Ahmed Imteaj, U. Thakker, S. Wang, J. Li and M. Hadi Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," in IEEE Internet of Things Journal, vol. 9, no. 1, pp. 1-24, 1 Jan.1, 2022, doi: 10.1109/JIOT.2021.3095077.

Ahmed Imteaj, and M. Hadi Amini, "FedAR: Activity and Resource-Aware Federated Learning Model for Distributed Mobile Robots", in Proceedings of the 19^{th} IEEE International Conference Machine Learning And Applications, 2020, Miami, USA.

Ahmed Imteaj and M. Hadi Amini. "FedPARL: Client Activity and Resource-Oriented Lightweight Federated Learning Model for Resource-Constrained Heterogeneous IoT Environment." Frontiers in Communications and Networks 2 (2021): 10.

Ahmed Imteaj and M. Hadi Amini. "Leveraging Asynchronous Federated Learning to Predict Customers Financial Distress." Intelligent Systems with Applications (2022): 200064.

Ahmed Imteaj, Irfan Khan, Javad Khazaei, M. Hadi Amini, "FedResilience: A Federated Learning Application to Improve Resilience of Resource-Constrained Critical Infrastructures," Electronics 2021, 10(16):1917.

Ahmed Imteaj, M.Hadi Amini, Distributed Sensing Using Smart End-user Devices: Pathway to Federated Learning for Autonomous IoT, 2019 IEEE Conference on

Computational Science & Computational Intelligence, 2019.

Ahmed Imteaj, M. Hadi Amini, and Javad Mohammadi. "Leveraging decentralized artificial intelligence to enhance resilience of energy networks." In 2020 IEEE Power & Energy Society General Meeting (PESGM), pp. 1-5. IEEE, 2020.

*Meleik Hyman, *Calvin Mark, *Ahmed Imteaj, Hamed Ghiaie, Shabnam Rezapour, Arif M. Sadri, M. Hadi Amini, "Data Analytics to Evaluate the Impact of Infectious Disease on Economy: Case Study of COVID-19 Pandemic," Patterns Journal (2021). [*Authors contributed equally]

M. Hadi Amini, Ahmed Imteaj, and Panos Pardalos, "Interdependent Networks: A Data Science Perspective", Patterns Journal (2020).

Ahmed Imteaj, Raghad Alabagi and M. Hadi Amini, "Exploiting Federated Learning Technique to Recognize Human Activities in Resource-Constrained Environment", in Proceedings of the 13th International Conference on Intelligent Human Computer Interaction (IHCI-2021), 2021, Ohio, USA.

Ahmed Imteaj, Distributed machine learning for collaborative mobile robots: PhD forum abstract, in Proceedings of the 18^{th} Conference on Embedded Networked Sensor Systems (SenSys'20), Association for Computing Machinery, New York, NY, USA, 798–799, 2020.

Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. "Federated Learning for Resource-Constrained IoT Devices: Panoramas and State-of-the-art." in Federated and Transfer Learning (2022).

M. Hadi Amini, Ahmed Imteaj, and Javad Mohammadi, "Distributed Machine Learning for Resilient Operation of Electric Systems", in Proceedings of International Conference on Smart Energy Systems and Technologies (SEST)(2020).