Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration

A thesis submitted during 2025 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in School of Computer and Information Sciences

by

ADITYA KUMAR

Registration No. 20MCPC03

School of Computer and Information Sciences

University of Hyderabad

(P.O.) Central University,

Gachibowli, Hyderabad – 500046

Telangana

India

CERTIFICATE

This is to certify that the thesis entitled **Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration** submitted by **Aditya Kumar** bearing **Registration Number 20MCPC03** for partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **School of Computer and Information Sciences** is a bonafide work carried out by him under my supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Further, the student has the following publications before submission of the thesis for adjudication and has produced the evidence for the same in the form of an acceptance letter or a reprint in the relevant area of his research. (Note: At least one publication in refereed journal is required)

1. A. Kumar, S. N. Srirama: **FIDEL: Fog integrated federated learning framework to train neural networks**, Software: Practice and Experience, ISSN: 1097-024X, Volume: 54, Issue: 2, pp. 186-207, February, 2024.
2. A. Kumar, S. N. Srirama: **FedStrag: Straggler-aware federated learning for low resource devices**, Digital Communications and Networks, ISSN: 2352-8648 Volume: 11, Issue: 4, pp. 1214-1224, August, 2025. KeAi. DOI: 10.1016/j.dcan.2024.12.004.

and

has made presentations in the following conferences:

1. A. Kumar, S. N. Srirama: **Fog enabled distributed training architecture for federated learning**, 9th International Virtual Conference on Big Data Analytics (BDA 2021), Prayagraj, India, December 15-18, 2021, pp. 78-92. Springer.

Further, the student has passed the following courses towards fulfillment of coursework requirements for the Ph.D.:

Course Code	Name	Credits	Pass/Fail
CS803	Data Structures And Programming Lab	2	Pass
CS804	Algorithms	4	Pass
CS800	Research Methods in Computer Science	4	Pass
CS867	Soft Computing	3	Pass
CS871	Research and Publication Ethics	2	Pass

Supervisor

Satish Narayana Srirama
Professor
School of Computer and Information Sciences
University of Hyderabad
Hyderabad-500 046, India

DEAN

DEAN
SCHOOL OF COMPUTER & INFORMATION SCIENCES
UNIVERSITY OF HYDERABAD
HYDERABAD - 500046, T.S. INDIA

DECLARATION

I, **Aditya Kumar**, hereby declare that this thesis entitled **"Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration"** submitted by me under the guidance and supervision of **Prof. Satish Narayana Srirama** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodhganga/INFLIBNET. A report on piagiarism from the University Librarian is enclosed.

Date: 21/08/25

Aditya kumar.

Signature of the Student:

Name: **Aditya Kumar**
Reg. No. **20MCPC03**

Counter Signed by Supervisor

S. Srirama

Satish Narayana Srirama
Professor
School of Computer and Information Sciences
University of Hyderabad
Hyderabad-500 046, India

Thesis Title: Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration

Student Reg. No.: 20MCPC03

Supervisor: Prof. Satish Narayana Srirama

School: School of Computer and Information Sciences

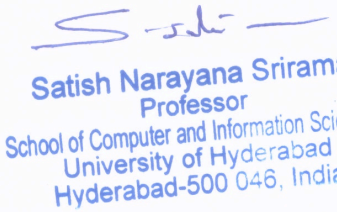
Among 17 goals (https://sdgs.un.org/goals), under the following SDGs, the work incorporated in the thesis will be addressed:

SDG 9 - Industry, Innovation, and Infrastructure

The past two decades have witnessed a digital revolution, with over 5.5 billion people now connected to the internet. The Internet of Things devices, sensors, and actuators have reached 21.5 billion in 2025, and it is projected to rise to 41.1 billion by 2030. This interconnected world provides a way to collectively build a system to make the world a better place. Our research on federated learning across the edge-to-cloud continuum directly contributes to SDG 9 by enabling resilient, inclusive, and sustainable digital infrastructure. This thesis proposes a practical solution for the interconnected world that enables development of a versatile system applicable across a wide range of real-world applications. By decentralizing model training and bringing intelligence closer to data sources, it reduces dependence on centralized cloud systems that lowers latency, and enhances data privacy. This promotes innovation in resource-constrained environments, especially in sectors like smart manufacturing, predictive maintenance, and real-time healthcare monitoring. This work is focused on developing a scalable infrastructure that leverages IoT devices to drive innovation and enable the creation of novel products and services. Finally, all the contributions are open-sourced, which will enable future researchers to innovate and build next-generation AI infrastructure that empowers industries.

SDG 11 - Sustainable Cities and Communities

In urban environments, huge amounts of data are generated daily from sensors, traffic systems, healthcare devices, and energy grids. However, sending all this data to centralized cloud servers is costly, slow, and raises privacy concerns. The thesis directly addresses these issues by proposing the FIDEL framework. By training AI models locally at the edge (close to where data is generated), our approach allows cities to make faster, privacy-preserving, and energy-efficient decisions. For example, real-time traffic prediction, air quality monitoring, and smart energy management can all benefit from this setup. The work also improves and addresses key challenges of stragglers in the network, which is inevitable. Additionally, the use of explainability ensures that these AI systems are not black boxes. Hence, city officials and citizens can understand why a model made a decision that builds trust and transparency. This directly contributes to creating inclusive, safe, resilient, and sustainable urban spaces, as envisioned in SDG 11.



Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, **Prof. Satish Narayana Srirama**, for his unwavering support, invaluable guidance, and constant encouragement throughout the course of my PhD. His deep insights, academic rigor, and meticulous attention to detail have profoundly shaped the direction and quality of my research. He gave me the absolute freedom to explore my potential and directed me towards achieving the goal. His mentorship has helped me grow as a researcher and instilled in me a disciplined and thoughtful approach to problem-solving. His passion for science and commitment to scientific betterment have been a constant source of inspiration. It has been a privilege to work under his supervision, and I am truly grateful for the trust he placed in me.

I want to extend my heartfelt gratitude to the members of my Doctoral Research Committee, **Prof. Rajeev Wankar**, and **Prof. Salman Abdul Moiz** for their insightful feedback, constructive suggestions, and continuous encouragement at every stage of my research. Their diverse perspectives and thoughtful inputs greatly enriched the quality of my work and helped me stay focused on my research goals. Their mentorship has been integral to my academic growth, and I remain sincerely grateful for their invaluable contributions to the successful completion of my PhD.

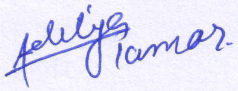
I would also like to thank the Dean of SCIS, **Prof. Atul Negi** and former Dean **Chakravarthy Bhagvati**, for fostering a research-conducive environment and for his consistent academic and administrative support. I am equally grateful to all the faculty members and staff of SCIS for creating a stimulating, collegial, and supportive atmosphere that has played a vital role in my doctoral journey.

Being the first research scholar in my family, I am deeply grateful to my family members for their unwavering faith in me. This journey would not have been possible without their patience, constant support, and encouragement. I would also like to express my heartfelt appreciation to **Harshini Easankarla** for her motivation and moral support throughout the course of my research.

I would like to express my sincere gratitude to the members of the Cloud and Smart Lab — **Rahul Siyanwal**, **Tejasri Reddy Kadari**, **Nirma Devi**, **Sreenivasu Mirampalli**, **Jashmin Swain** (WiseCom lab), and my other fellow researchers for their valuable suggestions and stimulating discussions throughout the course of my thesis work.

I would like to express my heartfelt gratitude to my friends, **Raj Kumar Sanayaima Singh** and **Charan Ramtej Kodi**, whose companionship has been a true blessing throughout this journey. Their constant support, both in academics and in life beyond, gave me strength and encouragement at every step. I deeply cherish the countless memories, guidance, and camaraderie we shared, which made this journey all the more meaningful.

Finally, I would like to thank the **University Grants Commission** for supporting me financially through the Junior Research Fellowship and Senior Research Fellowship, without which this journey would not have been possible.



Aditya Kumar

Abstract

The significant growth of Internet of Things (IoT) devices has led to the generation of a voluminous amount of decentralized data. The data is produced at various sources distributively, such as sensors, cameras, and smart gadgets, which presents a significant opportunity for developing intelligent applications. The traditional approach shares this data with the cloud server for model training. Due to intensive data transfer, the centralized computing paradigm faces various challenges such as high communication costs, latency, bandwidth, privacy concerns, and network congestion. Edge computing enables data processing directly on devices. Fog computing brings computational resources closer to the data source to facilitate computation. It enables distributed computing over geographically distributed nodes that can be used for real-time processing. However, Edge nodes can produce data continuously. The compute continuum (cloud, fog, edge) provides a network to process data at various layers.

Federated Learning (FL) has emerged as a promising decentralized training paradigm that allows learning from distributed data while preserving user privacy. It provides distributed training over various compute nodes. In FL, multiple clients participate in training with private data without sharing their raw data. Hence, the convergence of FL with the edge-to-cloud continuum has the potential to build next-generation AI applications. But FL training requires a strong compute node and a stable connection, which is a key issue in an edge-to-cloud continuum. At the same time, deploying FL over resource-constrained IoT devices introduces significant bottlenecks, especially in the presence of stragglers. The implementation of federated learning on the compute continuum can enable numerous AI applications. However, nodes in the fog layer are heterogeneous and resource-constrained. Further, the continuous growth of the data aggravates the challenge.

The thesis hypothesizes to build a solution for machine learning training on resource-constrained devices. The idea is to build a decentralized system on the compute continuum. This is achieved in two phases. First, we created an architecture to support the integration of FL on the edge-to-cloud continuum. It is a 3-layered architecture that supports continuous training on continuously growing data. Cloud layer works as an aggregator, Fog layer works as a client in FL training, and Edge layer produces continuous data during training. To address the resource limitations at the Fog layer, the work proposed an online approach. The simulation results suggest to us the feasibility of the ML training on the fog node. However, in the practical scenarios, device heterogeneity and stragglers are the major challenges. Considering all these issues, we built a framework, FIDEL, to support large-scale distributed training in a heterogeneous IoT network using Docker. The framework is implemented with both synchronous and asynchronous strategy that ensures seamless execution of the overall system. The training result of FIDEL for an Industrial IoT use case reaches the same result as centralized training.

In an IoT network, stragglers are inevitable, which slow down the training. Although asynchronous strategy reduces computation time, the impact of the straggler during training is unknown. The thesis investigated the impact of stragglers in model training. The experimental results suggested that stragglers have a negative impact on model convergence due to the existing aggregation method. The finding reveals that the FedAvg method aggregates stale updates during training that lead to prolonged convergence. To improve the system, we proposed a stragglers-aware weighted averaging scheme, FedStrag, to deal with stale updates. FedStrag prioritizes the latest update and penalizes stale updates. It provides a mathematical model to penalize stale updates based on the staleness of the model. The hybrid aggregation approach leverages both updates to learn faster and generalize well. We tested FedStrag on fixed and random strangers using both IID and non-IID data. The result suggests that it outperforms the FedAvg method on all possible straggler scenarios.

Training across the edge-to-cloud continuum is often constrained to limited data samples available at distributed nodes. The collaborative training is coordinated by the server using an aggregation strategy like FedAvg. FedAvg operates as an average-based aggregation function that synthesizes decentralized knowledge into a unified global model. However, the analysis of the convergence of the aggregation methods is limited to various assumptions that may not fit the real-world applications. The thesis hypothesizes that model convergence is inherently influenced by the contribution and importance of individual features. To analyze this, we integrated the SHAP-based xAI method into FL, which shows behaviors and convergence patterns. The proposed approach not only reveals the dynamics between feature importance and model convergence but also provides insights into model drift. The approach will contribute towards improving the transparency, reliability, and fairness of federated learning systems.

Overall, the thesis presents a system design perspective of federated learning implemented on real-world IoT devices. It addresses key challenges in the practical deployment of such systems across the compute continuum. The proposed work holds the potential to contribute towards the development of next-generation, scalable solutions for a wide range of applications.

Dedicated to my mother
Smt. Bachi Devi

Contents

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

We live in a world where Internet of Things (IoT) devices have become an integral part of our daily activities. From smart gadgets such as watches, heart rate monitors, temperature monitors, humidity sensors, smartphones that store information like photos, locations, text conversations, browsing history, chats, etc, to general-purpose CCTV cameras, GPS on vehicles. These devices are connected to each other over the internet to share data among themselves with multiple protocols that can perform a certain task [26]. Today, we are experiencing unprecedented growth in connected devices. According to the state of IoT summer 2024 report [7], there will be 21.5 billion connected IoT devices by 2025, which is forecasted to grow by 41.1 billion by 2030. These IoT devices include wired and wireless devices such as various sensors, industrial devices, medical equipment, home appliances, 4G/5G hardware, cameras, etc. As the number of devices has increased significantly, the amount of data generated has grown exponentially. According to the latest estimates from Statista, throughout 2024, approximately 402.74 million terabytes of data are being generated every day. As of 2024, total global data is estimated to be 149 zettabytes, which is likely to grow to more than 394 zettabytes by 2028 [112]. Such tremendous proliferation of digital data also provides opportunities to process this data, which can be applied to various use case scenarios such as smart home, smart health, security surveillance, autonomous driving, smart grid, continuous monitoring, digital twins, etc. These data are generated on various devices at various layers of the network. Extracting and processing such large data needs novel and sophisticated approaches that can apply business logic to solve a specific problem.

1. INTRODUCTION

A conventional way to extract information is to store everything on the server in one place and process it. This requires huge computational and storage resources, mostly at the server. The server processes the entire data to extract useful information to solve a specific business problem, which includes preprocessing, extraction, training, validation, etc. With virtually infinite resources, a cloud computing paradigm is generally preferred to store/process the data. Then, various machine learning algorithms, such as supervised, unsupervised, reinforcement, or semi-supervised learning, are applied to train a model for specific tasks. However, mentioned centralized approach can be inefficient, as it may not work well for a big data problem where data is continuously growing. At the same time, this may be an inefficient way to process large data at the server because data has to travel from the source to the server, which incurs massive bandwidth that may produce network congestion [15]. Additionally, the data has to be stored on a server, which requires huge storage and computational power to process it. Also, centralized processing is constrained by higher latency, which makes the system unsuitable for real-time processing. Distributed training is an alternative approach that can minimize the processing overhead of one place by various nodes. This can offload the work among multiple nodes that can process data simultaneously [55]. Since data is generated at various sources with multiple output devices/streams, it is more realistic to process it in a distributed manner. Fog computing [95] is a paradigm that offers computation near the source that supports real-time processing. It has limited computation and storage closer to the IoT device, which helps data processing on distributed nodes quicker than server-based processing [12].

One of the key challenges in data processing is the privacy and security of data. Since data is valuable and may contain private or sensitive information, a user/organization refrains from sharing raw data with the server. For example, a hospital is likely not to share its patient data due to patient privacy or laws like GDPR, HIPAA. For such a scenario, the data processing framework has to be privacy-preserving and sensitive to data security. Federated learning [48] is a distributed machine learning algorithm that offers privacy during training. McMahan [76] developed the paradigm that advocates privacy by restraining training data distributed over respective nodes. Federated learning (FL) aims to train a machine-learning model without sharing raw data with the centralized server. It has client-server components where the server works as a coordinator in distributed training, while clients own data and fine-tune the global model

with their local data. The training is done locally on multiple clients with private data. Once the local model is trained, it is shared with the server. The server works as a coordinator that collects and aggregates all local models into an aggregated global model. With this distributed nature of the training paradigm, it is compatible with the cloud-fog data processing architecture [78, 103, 118].

A machine learning model can be trained on distributed nodes with federated learning. However, the availability of resources is a key challenge in training at the edge of the network. In an IoT network, data is generated by sensing devices and shared for processing. Typically, edge devices have very limited resources that can not train any model by themselves. To facilitate efficient and real-time applications, we generally prefer the fog computing paradigm instead of cloud computing. However, fog computing still has limited computing resources. Additionally, in a cloud-fog-IoT architecture, the flow of data continuously increases over time. At the same time, the devices in the network are bound to fail/struggle due to various reasons that need practical solutions. A typical IoT network contains 3 layers: i) Cloud layer, ii) Fog layer, and iii) Edge layer. Each layer is designed to perform certain tasks based on compute/storage capability. Edge layers have data-generating devices that can produce continuous data. Fog layers have finite resources that facilitate lower latency in executing tasks. However, the cloud has virtually infinite resources to conduct in-depth analysis and report for the task. Hence, training a model in the edge-to-cloud continuum is a key challenge that we have tried to solve as a thesis work.

Training a model on a cloud-edge network has multiple applications, such as real-time security surveillance, fault detection in IIoT scenarios, anomaly detection, continuous monitoring, etc. Federated learning can be applied to an IoT network to learn a machine learning model in real-time. Here, the edge layer will generate data continuously and share it with the fog node. The cloud layer will work as an aggregator, and the fog node can be used to perform local training. Since fog nodes have limited computing power, performing training on fog nodes is challenging. At the same time, the size of the data at the fog node keeps growing due to the continuous production of data from the edge layer. Hence, there is a need for an architecture to run federated learning on resource-constrained devices on continuously growing data. We worked on this problem to resolve the training limitation of fog nodes. So, we designed a fog-enabled distributed training architecture that can harness the computational efficiency

of the fog-cloud node and preserve privacy while training the model. The proposed architecture is designed to train an FL model on continuous data. It offers continuous training by sequential data sampling at the fog node. The sampling is done based on the computational efficiency of the respective fog node. It formulates data modeling and slicing at the fog node for training. The proposed architecture is capable of training a model in a resource-constrained IoT network [50].

In federated learning, the client does not share raw data. However, it shares locally trained models with the server for aggregation. So, the server has to wait for clients to finish their training. Once training is finished and the local model is shared with the server, the server aggregates all local models and creates a global model, which is sent back to clients for further training. So, the server and client have to synchronize with each other to continue the training. One of the major challenges in synchronous communication is waiting time at the server. Here, the server is forced to wait for all the clients to finish. So if there are any stragglers or unreliable connections in the network, the server has to wait longer. In fact, if a device fails, the entire training freezes sometimes. To avoid this issue, the original work on federated learning has provided client sampling as a solution. However, partial aggregation can be inefficient for small networks with limited client size. Hence, asynchronous communication-based federated training is needed, which can minimize the waiting time and shorten overall training time. Therefore, we worked to create a fully functional framework that can utilize IoT infrastructure to train a model asynchronously. The framework should work effortlessly on various devices, irrespective of their hardware configurations. To realize this, we proposed a fog-integrated federated learning framework (FIDEL) for model training with both synchronous and asynchronous communication. The FIDEL proposes time-bound asynchronous federated training to handle stragglers. It runs on heterogeneous devices with continuously growing datasets. The proposed framework is developed with a lightweight containerization approach that can run on any hardware that supports Docker [52].

In an IoT network, devices are connected to each other through a wireless medium such as Wifi, Bluetooth, or cellular connection. Hence, it is expected to have a delayed update due to an unreliable network. At the same time, the network contains heterogeneous devices that may have stragglers. Stragglers in a network are those devices that

do not send their updates within the time frame. These nodes may have significant delays in model sharing due to various reasons such as unreliable networks, heterogeneous computation, device unavailability/failure, battery drainage, etc. Hence, Stragglers in an IoT network create key challenges during training. Even though the FIDEL framework reduces the impact of stragglers with asynchronous aggregation, it only ensures training is continued in the worst-case scenario. Hence, the impact of a delayed update on overall training is unclear. Since stragglers lag behind in the current training, they share old updates with the server. The server aggregates such old updates to the global model during aggregation, which may cause delayed convergence or divergence sometimes. Hence, the impact of stragglers in federated learning needs to be studied. We worked in this direction to find how much impact a straggler poses in overall training. Here, there can be two types of staggers i) fixed stragglers ii) random stragglers. In fixed stragglers, a particular device is delaying its update over time. This happens due to relatively low resources in the network compared to others in a heterogeneous network. While in random stragglers, any node in the network may delay its update. This scenario arises when devices are connected to an unreliable network. With various experiments[51], we found that stragglers have a negative impact on overall training. It not only slows down the convergence of the model, but it can diverge the training. Therefore, we proposed a weighted aggregation scheme (FedStrag) for stragglers to address the challenges of delayed updates. FedStrag priorities latest update over the old update. It ensures model convergence with multiple challenges, such as resource constraints, stragglers, network issues, and device heterogeneity. FedStrag accommodates every local update with its aggregation module by optimizing normal and delayed updates with respect to time. As a result, it outperforms baseline federated learning (FedAvg) in all possible cases of stragglers [51].

Federated learning is collaborative learning where multiple clients participate in achieving a common goal at the end. Even though each individual model is trained in isolation with their private data, the aggregated model is expected to learn global knowledge. This is achieved by aggregating local knowledge into the global model by federated averaging (FedAvg), which utilizes a mean function over layer-wise local parameters to compute the global model. How does a simple averaging over isolated learning work fine to learn patterns even from unseen data? It has been theoretically proven that federated learning converges to global optima with few assumptions, such

as convexity, smooth function, bounded variance [45, 59]. However, most of the neural networks trained in the FL setup are deep learning models, which leads to non-convex optimization [39]. The federated learning paradigm works equally for even non-convex complex optimization problems. But why and how a simple averaging works really well is a question that needs to be answered. A simple explanation-based strategy may help to understand the working and model convergence behavior of FL. So, we worked on the explainability of federated learning to demystify how the local model contributes to global knowledge. We conducted experiments to uncover feature contributions in creating a global model from local data that converges to the global knowledge. This is done by implementing the Shapley value (SHAP) [72] of the trained model on the global model to assess feature contributions in creating global knowledge. Since a neural network is a black box model, it is hard to interpret it internally. SHAP values expose feature importance for decision making. It highlights the most relevant and important features. We took Shapley value to measure feature contributions and then traced it for multiple rounds for federated learning. With extended experiments, we analyze the trends of feature importance over model training in federated learning. This reveals the importance of features in training that can help in optimizing a reliable and transparent machine learning model. The work provides an experimental granular convergence analysis of FedAvg at the feature level.

The thesis addresses the key challenges of the implementation of federated learning over the edge-to-cloud continuum. The thesis provides practical solutions for the real-time execution of distributed machine learning on low-resourced devices. It offers solutions to deal with stragglers and continuously growing data. Overall, hypothesis of the thesis is to provide a solution for privacy-preserving distributed machine learning training on resource-constrained devices. We realized this with the following four aspects:

- We hypothesize to design a federated learning architecture for continuously growing data in an IoT network. The proposed fog-enabled FL architecture addresses training a model on continuously generated IoT streaming data over fog nodes with a sampling technique.

- We hypothesize to build a federated learning framework for IoT networks with both synchronous and asynchronous strategies. The framework is capable of

utilizing heterogeneous fog-edge devices for data processing efficiently.

- We investigated the impacts of the stragglers in the federated model convergence. With the finding of delayed convergence, we hypothesize that minimizing the stragglers' impact on model convergence should improve federated learning training.

- We hypothesized to explain the internal working of the federated learning training. By exposing the importance of features with SHAP, we analyze feature contribution trends in the creation of the global model over time. It provides a convergence analysis of federated learning with feature impact using SHAP values.

1.1 Problem statement and research goals

Machine learning models are typically trained on large-scale datasets to optimize their performance and improve generalization. Centralized training requires data to be processed at the server; hence, it takes a huge bandwidth and introduces latency. However, distributed training involves collaboration with multiple nodes at various locations to train a model. Edge analytics refers to processing data near the source of data. It brings intelligence to the edge of the network. Training a model in the user's vicinity also provides privacy and security of the data. Bringing intelligence to the edge of the network not only optimizes available resources but also supports a seamless service experience for the end user in real time. For instance, it can rapidly process generated data to extract knowledge, enabling distributed data processing across multiple nodes while ensuring that raw data remains local and is not shared with the cloud. However, training a machine learning model on the edge of the network is constrained by various factors such as resource limitations, network bottleneck, communication bottleneck, and stragglers. Hence, this requires novel approaches to address the key challenges of distributed training.

We had an extensive literature survey on various distributed training paradigms. This includes multiple computing paradigms that should align with distributed training. Federated learning is one of such decentralized training mechanisms that advocates privacy preserving using distributed training. However, IoT devices lack computational

resources for data processing. The problem aggravates when the dataset keeps increasing on the edge node. Hence, we integrated fog computing paradigm with federated learning for efficient training. Our first research goal is to build an architecture that can support federated learning on resource-constrained fog nodes. Here, the amount of data at the fog node keeps on growing due to the continuous inflow of raw data at the edge node. To address this, we propose a three-layered fog architecture to train a global model using federated learning on a continuously growing dataset. Since edge nodes do not have enough resources to train data, they provide an online training solution for streaming datasets. The architecture proposes a solution for decentralized training for an IoT network. Next, we focused on building a training framework for federated learning on fog nodes. The goal is to build a framework that addresses various challenges of IoT infrastructure of real devices and provides a detailed analysis of the framework's capabilities. We proposed a FIDEL framework for distributed training using synchronous and asynchronous federated learning. It offers seamless training for a failure-prone IoT network. The framework is designed with a lightweight Docker container, which is suitable for running on any heterogeneous device. It utilized all three layers of the network for computation, such as edge layers for data collection, fog layer for processing, cloud layer for aggregation and big data processing. In a nutshell, this work provides a practical solution for privacy-preserving distributed training on the resource-constrained IoT network.

Since IoT networks often rely on low-powered and unstable devices, they are prone to failures and connectivity issues. Therefore, training machine learning models in such environments can become a slow and difficult process. FIDEL framework addresses this by using asynchronous updates. However, the impact of stragglers needs to be investigated. As a second research goal, we worked to examine the impact of stragglers on overall training. This work examines various types of straggler scenarios that can arise during training. Our analysis reveals that stragglers adversely affect model convergence, primarily due to stale updates. Although partial aggregation is a commonly used approach, it tends to be inefficient in smaller networks due to limited participation and reduced update diversity. Hence, a weighted aggregation scheme (FedStrag) is developed to incorporate the stragglers, which improves convergence. It prioritizes latest update over stale update with respect to time. The proposed work outperforms the baseline aggregation strategy, FedAvg, for all possible scenarios.

The third research goal is to explore the explainability of the model training in federated learning. The work looked into why federated learning works with simple averaging of models' parameters. The work incorporates explainability into fog-enabled federated learning architecture to explain the behavior of the model training for bias-free and transparent model training. Here, we wanted to understand how a particular feature contributed to global knowledge creation. To this extent, we examined feature contribution on the global aggregated model over various rounds using SHAP. The integration of an explainable AI method to federate learning enhanced the understanding of FedAvg convergence. With various experiments, we found that there is a correlation between feature impact and model performance. Also, it suggests that a few features contribute most to the global model. However, the impact of the few features is nominal. Here, we presented an analysis for federated learning using feature impact. This probes into federated learning training for the trustworthiness of the paradigm.

The overall thesis work addresses key challenges of the implementation of federated learning over an edge-to-cloud continuum. It provides practical solutions for real-time execution of distributed machine learning while dealing with stragglers. The thesis work is based on three research goals and formulates four research questions, which are defined as follows.

Research Goal - 1:

Exploring and developing a framework for distributed learning with resource-constrained devices for real-time machine learning training on edge-to-cloud continuum.

Regarding research Goal 1, we have formulated two research questions. Since machine learning, specifically neural network training, needs significantly large computational and storage resources, we wanted to investigate and explore the possibility of model training in an IoT network. One of the major challenges in building such a system is the resource limitation at the edge of the network. Apart from that, in an IoT network, we expect to generate data continuously. This continuous inflow of data requires more processing power to process it. Hence, data generated at the edge node grows continuously, which leads to a new problem. So, rather than centralized training, can we utilize various IoT devices to train a decentralized model using distributed data? How to deal with the continuous dataset to extract meaningful insights? Addressing

these challenges, we formulated our first research question as follows.

Research Question - 1:

How to train a privacy-preserving decentralized model on distributed nodes on resource-constrained heterogeneous edge/fog nodes? Is it possible to train the neural network on a continuously growing dataset with resource limitations?

Edge computing paradigm introduces computation at the edge nodes. However, in an IoT network, edge layers have devices such as cameras, sensors, mobiles, etc, which do not have enough resources to process large data. Fog computing brings limited computing and storage capabilities to the edge that can be utilized for efficient processing. Due to its proximity, the Fog computing paradigm has been extensively used for distributed and real-time processing. To answer the first research question, we integrated the fog computing paradigm to federated learning with efficient use of resources at the fog layer. The key challenge is to harness low resources at the fog for complex tasks such as model training. In this direction, we introduced an online approach for efficient training on a fog node with a continuously growing dataset. We proposed an architecture for federated learning training with an online training approach. In the simulation on containers, the architecture is capable of training a model for an IIoT task.

Next, we extended the work for a more realistic use case in IoT scenarios. In federated learning, the server waits for all clients to train their local models. In IoT networks, node failures are expected due to factors such as power outages, battery limitations, intermittent availability, or straggling behavior. As a result, synchronous federated learning often becomes impractical for real-world training scenarios. To address this, asynchronous or semi-synchronous updates are needed for model aggregation. Another significant challenge in real-world fog and edge computing deployments is device heterogeneity, as clients often consist of multiple devices with varying hardware capabilities and configurations. This motivates us to form our second research question and the bigger goal of distributed training on fog/edge nodes. We formulate the second research question as follows.

Research Question - 2:

What is the feasibility of developing a framework for distributed model training in a failure-prone and heterogeneous IoT network for real-world applications? What challenges are involved in its implementation, and can such a framework achieve convergence comparable to that of centralized training?

There are multiple frameworks for federated learning, such as Tensorflow Federated (TFF)[1], OpenFL [2], FATE [66], PySyft [92], Flower [10], NVIDIA FLARE [83], IBM FL [37], Federated Scope [119], Sherpa.ai FL [91], FLSim [57], etc. TensorFlow Federated is one of the first frameworks developed by Google for federated learning research. However, we can only run a simulation-based architecture using TensorFlow. Some of the frameworks, such as NVIDIA FLARE, FATE, OpenFL, etc, are developed by the industry for model training and deployments in an industrial setup. These frameworks offer very limited customization for research. At the same time, they are closed-source. However, frameworks like Flower and PySyft are open source, which can be modified for research work. But, when it comes to training the model in resource-constrained devices, most of the framework fails because they are built for distributed computing on resourceful devices. Even though Flower can run on heterogeneous devices, it only offers synchronous updates. Hence, to address research question 2, we wanted a framework that supports asynchronous training with streaming data using resource-constrained devices. Hence, we found a gap in the available framework that motivated us to build a framework for a failure-prone IoT network. The proposed framework is extensively tested with multiple models and compared with centralized training.

Research Goal - 2:

To analyze the impact of stragglers on model training in heterogeneous IoT networks and develop effective strategies to mitigate their effects, with the aim of improving training efficiency and enhancing the model's generalization performance.

An IoT network consists of heterogeneous devices that may have unreliable connections. Hence, stragglers are inevitable in the network. Stragglers are those devices that delayed their update. This could happen because of computational limitations or network failure/congestion. Training on such a network poses a challenge because of delayed updates. In this scenario, a few of the devices are faster that finish their

update earlier than the other slower devices. To overcome this, asynchronous update is preferred over synchronous aggregation. But in an asynchronous federated learning, the server waits for a fixed amount of time before aggregation, after which the global aggregated model is created. Since stragglers delayed their update, the server combined the old update while aggregating. For example, if a straggler's node delays its update by 2 iterations, the update will be aggregated at the server when it is received. This means that the update was stale by 2 iterations and was trained on the old global model. Regarding Research Goal 2, we wanted to understand whether the stale update impacts the global model convergence or not. What kind of stragglers are possible in an IoT network? How much do they impact model accuracy? Hence, we conducted extensive experiments to determine the impact of global model convergence. Further, we wanted to avoid the negative impact of the stragglers, if any.

Generally, federated learning training avoids stragglers by partial aggregation. Here, the server waits for a minimum number of updates or goes ahead with available updates [65]. This approach can be efficient for a large network with thousands or millions of nodes. However, for a smaller network with fewer nodes, eliminating any devices may lead to overfitting, because each node may contain a separate data distribution that needs to be learnt specifically for a non-IID case. Another approach uses categorization of the nodes into fast nodes and slow nodes. Then, separate aggregation is applied to both categories [14, 60]. Similarly, cross-category and intra-category weighted aggregation is also explored for model training [21, 123]. Most existing approaches primarily focus on grouping devices based on their capabilities and performing training within these groups, which still requires large nodes for processing. For research goal 2, we wanted a solution for a smaller network, for example, a network in a housing society that has limited compute nodes. Then resolve the impact of stragglers in model training. We formulated our Research question-3 around this, which is as follows.

Research Question - 3:

To what extent do stragglers impact model training in distributed learning environments? Furthermore, can the contributions from these delayed or underperforming nodes be effectively utilized to improve the model's generalization performance?

Federated learning creates a model by aggregating multiple individual models. These models are trained on an independent dataset, which is privately owned by different owners/devices. The devices only share trained parameters to the server without any access to raw data. The server applies the FedAvg method to aggregate all locally trained models. The FedAvg method is an averaging method that combines all learning into one global learning. Though FedAvg works well and is capable of training and creating a global model, it is interesting to know why it works. To understand this, we created our third research goal to examine and explore how/why the simple averaging method, like FedAvg, converges. We formulated our research Goal 3 as follows.

Research Goal - 3:

Investigate the underlying mechanisms that contribute to the effectiveness of the Federated Averaging algorithm in distributed learning settings. Specifically, it seeks to quantify the influence of individual models and feature-level contributions on the formation and convergence behavior of the global model.

Learning in federated learning happens in isolation at multiple points with personal datasets. Since data is not shared with any party during training, the server only receives model parameters for aggregation. During aggregation, the server performs an averaging method called FedAvg to consolidate all local models. In the previous goals, we have trained multiple deep learning models with FedAvg. With a simple understanding of optimization theory, it is hard to understand how deep learning methods optimize the model due to non-convexity. Still, FedAvg method has optimized non-convex functions, which are trained on heterogeneous devices/datasets. Why does a simple aggregation method work so well on not only convex but also non-convex problems? This motivates us to investigate how the federated learning paradigm effectively enables model training in a distributed environment. One key reason lies in its ability to simplify the aggregation process by reducing variance across client updates, thereby aligning the aggregated model more closely with the global objective. That means each client learns a separate model on their data, pertaining to fit local data that may have noisy estimators for global optima. These estimators overfit models to local data, which can be far from the expected global model. But the aggregated model cancels

out overfitting by averaging the parameters. Even though it looks reasonable by visualizing the effect of FedAvg, such a simple explanation cannot guarantee the convergence of the method. Hence, multiple researchers have shown theoretical convergence of the FedAvg method [27, 36, 61]. The backbone of the convergence lies in the assumption of smoothness, bounded variance, and convexity of the function. However, deep learning model training is a non-convex optimization problem. Arguably, federated learning converges because it performs stochastic gradient descent (SGD) at the client level. Here, each client performs local SGD and the server aggregates. This is also seen as performing batch SGD over distributed nodes. The theoretical analysis suggests that the algorithm converges to the global optimum. However, how the aggregated model converges to the optimum global model is interesting to know. How does each feature in the dataset impact the global model over time? This calls for an importance-based analysis to quantify each feature's contribution to global convergence. Motivated by this, our work explores the extent to which individual features drive the global optimization process. Accordingly, we formulated the following research question.

Research Question - 4:

How significantly does each individual feature contribute to the construction of the global model during federated learning? Furthermore, what patterns emerge in feature importance over the course of training rounds?

1.2 Research methodology

To achieve the research goals, we proposed four research methodologies to address all four research questions mentioned above.

To address the research question 1, we worked on the implementation of the federated learning method on resource-constrained devices. The focus is to propose a practical solution for federated learning training on distributed nodes. We investigated the feasibility of neural network training in resource-constrained environments such as edge/fog nodes. Since edge nodes do not have enough computations for ML training, we focused on the fog computing paradigm for efficient training at the edge of the network. This serves two aspects of the training: (i) Distributed training aims to train

on independent nodes that serve real-time processing, (ii) Privacy-preserving training that assures users that their data is safe in their own vicinity by not sharing it with the server. We propose a fog-enabled distributed architecture that integrates the fog computing paradigm with federated learning to enable efficient and scalable model training. A central challenge in this integration lies in ensuring real-time data processing at the fog nodes, which is critical for continuous learning in dynamic environments. Because in the IoT network, the edge nodes generate data continuously, say 24×7, making it hard to train on the fog node. Since a neural network training needs a significantly large amount of data to fine-tune the model, we proposed an online training scheme to handle continuously generated data. The training process is formulated using a data modeling approach that incrementally slices the incoming raw data across training rounds, enabling real-time learning. Unlike conventional machine learning, we do not process complete datasets for model training. Rather, training is performed on part of the dataset that keeps changing with time. So, for a given round, a set of datasets is used for training, while in another round, another set of data is used for further fine-tuning the model. This process is repeated over multiple fog nodes over time. Here, slicing of data is established by an assumption that edge nodes associated with fog nodes are producing 60 frames per minute, ie, one frame per second. Hence, at the fog node, we trained the model with 60 frames only. Since the dataset is reasonably small, fog nodes can train/fine-tune the model. However, the same dataset will not be used again because in the next round, a new dataset will be generated. Since every data point is looked at once, and it is changing on every round, the proposed architecture is the best suited for real-time machine learning training. We set up the methodology and implemented it for a real-world use case. For experimentation, we created a Docker container-based application for the simulation of the proposed architecture. Docker containers are lightweight and can be considered as fog nodes for distributed training. In addition, we have used gRPC remote procedure call for efficient communication between the cloud node and fog nodes. gGPC facilitates communication between cloud and fog nodes for model transfer, and produces calls for model training at fog nodes. To test the learning capability of the architecture, we used radar data to train safe position classification in a human-robot workspace.

To address Research Question 2, we extended the proposed fog architecture to develop a federated learning framework for the edge-fog-cloud continuum. This extension

enables collaborative model training at the network edge while accounting for the limitations in computation, memory, and energy typical of such environments. Simulation results show that a fog-enabled federated learning architecture is capable of training a neural network. However, implementation of the architectures on real devices has 3 major challenges: i) Device heterogeneity ii) Connection/device failure iii) Stragglers. A distributed training in the fog network has to address these additional challenges. We have taken these challenges into account and built a framework, FIDEL, that supports online training in an IoT network. FIDEL is developed with a lightweight containerization approach that can run on a device that supports Docker. The Docker engine provides a consistent runtime environment for executing complete packages or images, effectively addressing hardware heterogeneity. By bundling all necessary dependencies within the container, Docker ensures the smooth execution of functions, making it possible to run applications seamlessly across diverse devices. FIDEL framework is designed to have three layers: i) Cloud layer is responsible for aggregation, visualization, and reporting, ii) Fog layer is responsible for local model training and data storage, iii) Edge layer only generates data and shares it with the respective fog node. All nodes in each layer perform their tasks in parallel and communicate vertically over the network. The implementation of the FIDEL has to deal with the continuous inflow of data from edge nodes. Hence, we designed the methodology for online training in two stages: i) All raw data generated at the edge layer will be stored at the respective fog node. The fog node will always stay inside the user's vicinity to ensure privacy of the data. ii) Fog nodes selectively retrieve training data from local storage for processing. As a result, even if the edge layer generates data at a rate that exceeds the available computational capacity, the training process remains uninterrupted, avoiding system freezes and ensuring continuous learning. The selection of training data is constrained to the computational capability of the individual fog node. So, different fog nodes may have different training data sizes at a particular round. FIDEL also provides a mathematical formulation for continuous data processing based on underlying resources.

A key challenge in federated learning arises from the communication between the cloud server and fog nodes over IoT networks, where the server has to wait for updates. However, in real-world IoT environments, timely participation from all nodes cannot be guaranteed. This may happen due to device failure, network congestion, compute constraints, or stragglers node, which are inevitable in an IoT network. While answering

research question 1, we addressed computational constraints by online training, however, the training was conducted in a synchronous way. To address the key challenge in model implementation, the FIDEL framework is designed with asynchronous federated learning. The server does not need to wait for every client to complete training; it proceeds with available updates. Basically, it waits for a certain amount of time, then applies aggregation for the next round. We used MQTT for asynchronous communication between the cloud and the fog node. The asynchronous communication is achieved on two topics: i) train ii) aggregate. The fog node starts training when there is a call on the train topic and the server receives the locally trained model on the aggregate topic. Hence, the FIDEL framework deals with network/device failure, resource constraints, and device heterogeneity, which makes it best suited for distributed training on an edge-to-cloud continuum. To check the capability of the FIDEL framework, we experimented with shallow, deep, and convolutional neural networks over RADAR data for safe position detection. All experiments were conducted on Raspberry Pi devices with limited computational capabilities. The framework demonstrated performance comparable to centralized training in terms of model accuracy. Additionally, a comprehensive system analysis was carried out to evaluate the framework's feasibility for deployment in real-world IoT networks. The results indicate minimal resource consumption, with the overall memory usage remaining under 4% on a 4GB RAM Raspberry Pi.

To address Research Question 3, we began by analyzing the impact of straggler nodes on the model training in federated learning paradigm. As discussed earlier, FIDEL addresses key challenges of training in a failure-prone network. It performs asynchronous aggregation for the stragglers. However, in this process, the server aggregates stale updates into the global model. We investigated the impact of stragglers' updates and proposed a solution to improve model convergence. Stragglers are those nodes that delay their update. This can happen due to device constraints or unreliable networks. The methodology for this research is structured in two phases. The first phase investigates whether stragglers have a significant impact on model convergence in federated learning. For this, we set up various types of stragglers possible in an IoT network. We experimented with random stragglers and fixed stragglers. For the completeness of possible stragglers, we consider i) Fixed straggler, ii) Random stragglers. In the case of Fixed stragglers, a particular node delays its update every time. This implies a device issue or computational inefficiency of the particular device. However,

in Random stragglers, a random device delays the update. This pertains to network congestion or failure. We experimented with these two possible stragglers and found that stragglers have a negative impact on model convergence. In fact, as we increase the delayed time, stragglers diverge from the global model. The primary reason stragglers affect model convergence is that they perform local training on outdated versions of the global model, leading to stale updates that may hinder the overall learning process. This suggests that stale updates are slowing down the convergence, which motivates us to address the problem. In the second phase, we propose a solution to deal with stragglers so that the learning of these nodes can also be accommodated in the global model for better generalization. In the conventional FL paradigm, delayed updates are discarded during partial aggregation. However, for smaller networks, discarding updates impacts model generalization. Because every node learns a unique pattern from its data, which needs to be included in the global model. The proposed solution, FedStrag, is a weighted averaging that prioritizes the latest update over stale updates. The core idea is to incorporate every local update into the global model, ensuring that no learning is discarded. This inclusive approach promotes better generalization by leveraging the diversity of information from all participating nodes. We formulated FedStrag as a penalty-based weighted aggregation on the staleness of the model. The staleness is determined by the number of rounds a device has delayed its update. Based on this delay, FedStrag assigns a weight to each model update, which is then used in the final aggregation process to ensure that outdated updates have a reduced impact on the global model. For example, if a node delays its update by 2 rounds, then a penalty of 1/3 is forced during aggregation. The penalty-based aggregation scheme for model training ensures that every participant's contribution is considered in the final aggregation, while simultaneously preventing the discarding of any updates, thereby preserving valuable learning from all devices. To assess the convergence of FedStrag, we conducted multiple experiments under straggler scenarios, using both IID and non-IID versions of the MNIST dataset. The results demonstrate that FedStrag consistently outperforms the baseline FedAvg across all tested use cases.

We addressed Research Question 4 by incorporating SHAP-based explainability into the federated learning training. Interpretability of a black box model is one of the key areas that needs to be advanced to gain trustworthiness in the system. Federated learning trains a model in collaboration with multiple stakeholders. As discussed in

the previous section, a simple weighted averaging method converges toward the global optimum. Existing research suggests that this convergence is a result of FedAvg functioning similarly to stochastic gradient descent (SGD) across clients, with theoretical guarantees supporting its effectiveness. However, federated learning is an iterative process where the global model is created over time with clients' contributions. It starts with a weak model, which is fine-tuned at various clients and gets stronger over multiple rounds. Hence, as a weak model converts into a strong model over time/rounds. We wanted to understand how a particular feature contributes to the process. To analyze this, we incorporated a Shapley value explanation for each feature. The SHAP value provides the importance of each feature when it makes a final decision. In a nutshell, it calculates the behaviors of the model with and without the feature, then it calculates the contributions of the particular feature. We integrated the SHAP value in federated learning training to know individual feature contributions in the final decisions over time. The feature-based contribution reveals trends for each feature in the global model during aggregation. A detailed analysis of each feature and model contributions is presented to support explainable federated learning. We experimented with two different models with Iris and Predictive maintenance data sets to know how feature converges along with model convergence. This enhances bias-free training that improves trustworthiness in the federated learning paradigm.

In summary, the thesis addresses key challenges of federated learning in the edge-to-cloud continuum. Firstly, we looked into the feasibility of training federated learning on resource-constrained devices. In this work, we address the key challenge of continuous training on rapidly growing datasets. In this regard, we proposed a fog-enabled federated learning architecture for model training on distributed nodes. The architecture uses online training that enables resource-constrained devices to train the model on a large dataset. Secondly, we extended the work and proposed the FIDEL framework for implementation on real devices such as Raspberry Pi. The proposed work addresses key challenges of device heterogeneity, stragglers, and network/device failures. FIDEL offers synchronous and asynchronous federated learning that ensures seamless execution of the server aggregation. We have also shown a mathematical model for handling streaming datasets. Thirdly, we looked into the impact of stragglers on model convergence. With various experiments, we found that stragglers have a negative impact on global model convergence due to staleness of the update. To resolve it, we proposed

a FedStrag method that prioritizes the latest updates over old updates. FedStrag is a penalty-based weighted aggregation strategy that minimizes the stragglers' impact and improves the model's generalization. The results show that FedStrag outperforms the FedAvg method on every possible straggler scenario on both the IID and Non-IID data. Finally, we integrated an explainability module into the federated learning paradigm, with a focus on investigating the contribution of individual features to the global model over multiple training rounds. We analyzed the trends in feature impact during the aggregation process and provided an in-depth examination of how each feature contributes to the creation of global knowledge. This analysis not only enhances the transparency of the federated learning process but also improves its trustworthiness by offering insight into the influence of specific features on model convergence.

1.3 Contributions of the thesis

The thesis worked towards three research goals that answer four research questions. The focus of the thesis is to address various challenges in distributed training and to provide practical solutions for them. It also looked into explaining the federated learning paradigm for the trustworthiness of the paradigm.

Contribution 1: In this work, we looked into the feasibility of federated learning on a continuously growing/changing dataset over a resource-constrained device. We proposed a fog enabled distributed training architecture for machine learning tasks. A hybrid of fog computing and the federated learning paradigm is used for the model training. To address the demands of continuous data streams, we introduced an online training scheme capable of handling continuously generated data without interrupting the learning process. The training includes only recent periodic data for modelling. The system assures privacy by restricting the raw data to the fog level. In addition, by not sharing raw data directly to the server, the system optimizes network bandwidth and congestion. We simulated the proposed architecture with Docker container. To test the learning capability of the model, we used radar data to train safe position classification in the Human Robot (HR) workspace, which achieves an accuracy of 99%.

Contribution 2: In this work, we designed a fog-integrated federated learning framework, named FIDEL, for distributed data processing on continuous and dynamically growing datasets. FIDEL efficiently utilizes resource-constrained edge devices to

collaboratively train neural networks with the cloud, providing a robust federated learning architecture that is specifically designed for real-world distributed data processing applications. A key contribution of the framework is the design of an asynchronous federated learning (AFL) strategy, specifically developed to address challenges associated with straggler nodes and unreliable network conditions. The framework also includes a formal mathematical formulation that enables resource-aware processing of large-scale data in distributed environments. The overall system integrates a edge–fog–cloud architecture with federated learning, efficiently orchestrating data processing and machine learning tasks using lightweight Docker containers deployed on Raspberry Pi devices. FIDEL was implemented and evaluated using both synchronous and asynchronous federated learning strategies on continuous datasets. The containerized design ensures portability and facilitates seamless operation of the framework across diverse, heterogeneous device environments, regardless of underlying hardware or software configurations. Finally, we applied FIDEL to a real-world use case: training a model for human-safe position detection in a human–robot (HR) collaborative workspace, a critical problem in industrial automation. Experimental results demonstrate that FIDEL achieves comparable accuracy to centralized and existing federated learning frameworks, even when deployed entirely on Raspberry Pis. This validates its computational intelligence, efficiency, and practicality in real-world industrial applications.

Contribution 3: This work focuses on analyzing the impact of stragglers in federated learning. Through extensive experimentation, we observed that stragglers adversely affect model convergence, primarily due to delayed or stale updates. To address this, we proposed a stragglers-aware weighted averaging scheme, FedStrag, to deal with stale updates. The proposed scheme is trained on a continuous inflow of data with online training on resource-constrained devices for various possible straggler scenarios. We implemented a time-bounded asynchronous federated learning approach that efficiently handles stragglers and scalability. Finally, we trained an image recognition task with FedStrag over Raspberry Pis with IID and non-IID datasets. The empirical results show that the FedStrag performs better than the baseline FedAvg for both IID and non-IID scenarios

Contribution 4: In this work, we integrated an explainability module into the federated learning paradigm to enhance the trustworthiness of the system. The proposed system utilizes the Shapley value to calculate the contribution of every feature during

global model creation. We analyze how each feature's influence evolves over successive training rounds, capturing trends in their relative importance and contribution to model performance. Training was conducted on the IRIS and Predictive maintenance dataset, which suggests that some features are dominating in decision making while others have a significantly low contribution to the global model. The result also explained model drift due to changes in feature impact. The global model follows local model behaviors at the granular level. Hence, any changes in the local model impact overall training. These insights lay the groundwork for developing bias-aware and trustworthy federated learning systems in the future.

1.4 Structure of the thesis

This section presents the overall structure and organization of the thesis, outlining the chapter-wise breakdown and flow of the work.

Chapter 1 introduces core problem statements, outlines the research goals, and presents the research questions that guide the thesis. It described methodologies to solve these research questions in detail. We have also discussed the contributions of the thesis in this chapter.

Chapter 2 presents a comprehensive literature survey and an overview of state-of-the-art research relevant to the thesis. It has a detailed discussion about the essential background of the key concepts.

Chapter 3 talks about research goal 1 that offers a solution for distributed training for IoT networks. It discusses solving research questions 1 and 2, which offer a practical solution for federated learning on fog nodes using both synchronous and asynchronous updates.

Chapter 4 answered research question 3 that offers a solution for stragglers in an IoT network. It discusses the impact of staggers and proposes the FedStrag method to mitigate the negative effects of stale updates and enhance overall model convergence.

Chapter 5 explores the explainability aspect of the federated learning paradigm, with a particular focus on analyzing feature contributions within the FedAvg-based aggregation. It examines how individual features influence global model formation over training rounds, providing insights into model behavior and interpretability.

Chapter 6 concludes the thesis by summarizing the key findings and contributions. It also outlines potential directions for future research and highlights opportunities for extending the current work.

1.5 List of publications

1. Aditya Kumar, S. N. Srirama: **Fog enabled distributed training architecture for federated learning**, 9th International Virtual Conference on Big Data Analytics (BDA 2021), Prayagraj, India, December 15-18, 2021, pp. 78-92. Springer. DOI: https://doi.org/10.1007/978-3-030-93620-4_7 (Scopus-Indexed)

2. A. Kumar, S. N. Srirama: **FIDEL: Fog integrated federated learning framework to train neural networks**, Software: Practice and Experience, ISSN: 1097-024X, Volume: 54, Issue: 2, pp. 186-207, February, 2024. Wiley. DOI:10.1002/spe.3265 (SCIE, Q2, IF-2.6)

3. A. Kumar, S. N. Srirama: **FedStrag: Straggler-aware federated learning for low resource devices**, Digital Communications and Networks, ISSN: 2352-8648 Volume: 11, Issue: 4, pp. 1214-1224, August, 2025. KeAi. DOI: 10.1016/j.dcan.2024.12.004. (SCIE, Q1, IF-7.5)

Chapter 2

Background & Literature Survey

This chapter provides a comprehensive overview of the key concepts, terminologies, and related literature used in the thesis. Firstly, it provides details about the IoT computing paradigm, which serves as the foundational layer for modern distributed systems. An IoT computing paradigm consists of multiple layers that offer distributed computing at various layers of the network. Then we delve into the principles of distributed computing and federated learning. Particular emphasis is given on the communication strategies, aggregation mechanisms, and challenges such as system heterogeneity and stragglers that influence the performance of federated training in the edge-to-cloud continuum. Then it discusses model interpretability and explainable machine learning, which is increasingly important for trust and transparency in AI systems. Finally, the chapter provides an extensive state-of-the-art literature survey aligned with the thesis objectives. We reviewed significant contributions by various researchers in the domains of IoT-based computing, federated learning frameworks, and explainability techniques.

2.1 Background

This section defines fundamentals and key concepts used in this thesis. We will have a short discussion on key aspects of each terminology relevant to the thesis work.

2.1.1 Internet of things

Today, IoT has become a ubiquitous technology that connects everything. It refers to a network of interconnected physical devices equipped with sensors, software, etc,

that coordinate with each other over the internet [15, 71]. With the advent of various hardware and software technologies, the concept of IoT is materialized with edge-to-cloud that ensures data generation for modern business applications. However, the foundation of IoT lies way back in the 1980s when the idea of the first interconnected appliances was proposed at Carnegie Mellon University (CMU). The researcher at the CMU wanted to monitor the status of the Coca-Cola vending machine over the Internet (ARPANET). This modest innovation demonstrated how physical objects could be monitored remotely, laying conceptual groundwork for the IoT. However, the term Internet of Things was first introduced in 1999 by Kevin Ashton, who was MIT's Executive Director of Auto-ID Labs. He describes IoT as a system in which a physical object with RFID (Radio Frequency Identification) should be connected to the internet to monitor it [8]. His vision was to make machines capable of observing and collecting data about the environment without human intervention, primarily to improve supply chain management[96, 115].

The core idea of the IoT is to gather information from the surrounding environment to enable control, facilitate analysis, and execute appropriate actions. An IoT system is composed of smart objects or devices that can be identified, located, addressed, and controlled remotely through the internet using technologies like RFID, wireless LAN, or other communication protocols. These smart objects act autonomously to interact with the physical world, enabling real-time decision-making and automation across a wide range of applications. To achieve seamless data exchange, IoT leverages the internet through interconnected technologies, such as Wireless Sensor Network (WSN), Bluetooth, ZigBee, Wi-Fi, cellular networks, and RFID. A widely accepted definition of the IoT is provided by the Cluster of European Research Projects (CERP), which states that "The Internet of Things allows people and things to be connected anytime, anywhere, with anything and anyone, ideally using any path/network and any service" [100]. The term "Internet" is used as a global communication infrastructure that provides connectivity among devices, and "Things" refers to a broad spectrum of physical entities, including sensors, smart devices, objects, etc. Though there have been multiple proposed architectures, we are focusing on a four-layer IoT architecture as shown in Fig 2.1. This enables various applications such as smart homes, smart grid, industry automation, smart healthcare, etc.

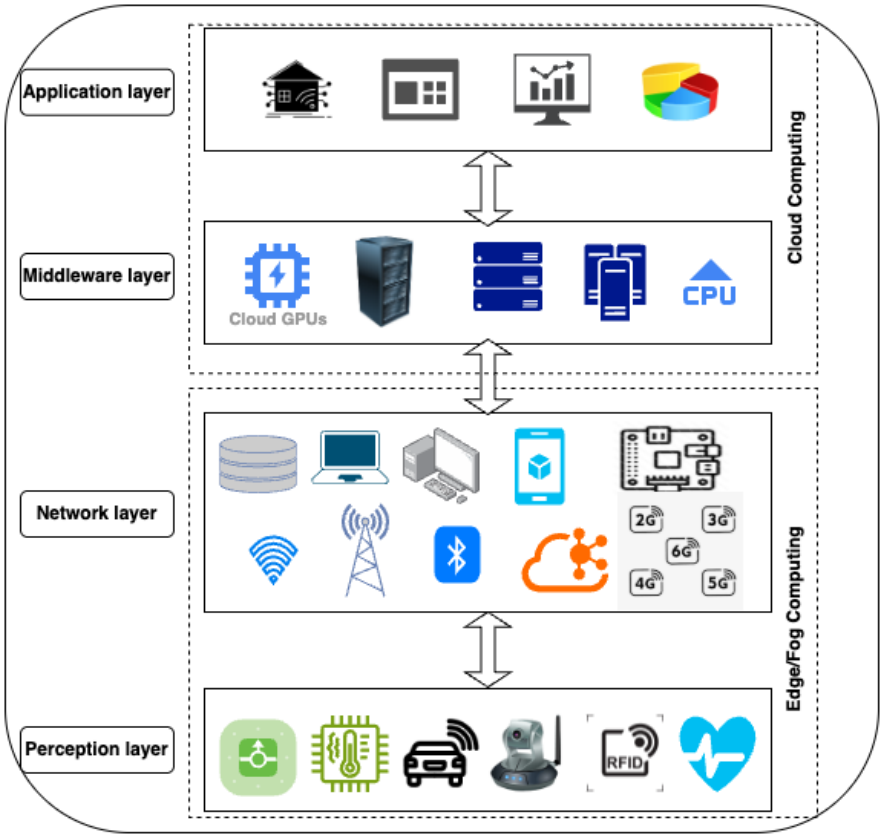


Figure 2.1: IoT Architecture

The architecture consists of 4 connected layers that manage data generation, transformation, communication, and processing. Each layer is intended to perform specific tasks to solve a business application.

The **perception layer** contains physical hardware in an IoT system, such as sensors, RFIDs, cameras, etc. Depending on the application, these sensors collect contextual data (location, temperature, humidity), which is then transmitted to the network layer for further processing. So it works as a data collection layer at the end of the network.

The **network layer** consists of various data transmission devices that ensure connectivity to the perception layer. It has limited computing capability to process data as a fog/edge node. It ensures the secure transmission of data from sensor devices to processing units using both wired and wireless technologies, such as ZigBee, Wi-Fi, Bluetooth, infrared, and cellular networks (3G/4G/5G). This layer facilitates distribu-

tion and preliminary processing of IoT data at the network's edge before forwarding it to the middleware layer.

The **middleware layer** manages the storage and processing of data received from the network layer. It has infinitely large compute/storage resources such as HPC, cloud node, etc. This layer supports advanced data analytics, ubiquitous computation, and automated decision-making. It is designed for high-performance computing and big data storage, making it essential for processing and analyzing large volumes of IoT data within integrated cloud environments.

Finally, the **application layer** serves as a user interface that enables users to monitor, control, and interact with the system. It provides a holistic view of the system, where data flows across multiple layers from sensing and communication to processing and application, ultimately delivering meaningful outcomes and services to the end user. The IoT system is built on heterogeneous devices with various communication protocols between layers. Hence, an IoT architecture serves as a blueprint for designing and developing a wide range of applications. Common IoT applications include smart homes, smart cities, smart healthcare, smart farming, smart retail, and intelligent supply chain systems [29, 87].

Though IoT architecture offers a wide range of applications, it has various practical challenges during implementation. The entire architecture heavily relies on connectivity and compute power of the underlying network. Hence, resource constraints are a key challenge in applying complex tasks like machine learning. Since it has a hierarchical structure, the data is moved from one layer to another to address some of these concerns. However, connectivity of cross devices/layers is mostly dependent on the internet or a local network. Data transfer of these connected devices is subject to the availability of the network. At the same time, devices at edge/fog layers often have a battery or an unstable electricity supply, which makes them prone to intermittent availability and unexpected failure. Hence, IoT devices are prone to failure. These forced the network to create a straggler node that creates a hurdle in seamless data processing. Stragglers are nodes that do not process their data on time. We have found that the stragglers influence a model training negatively. So, how to deal with such an inevitable situation for distributed data processing is a challenge that is discussed in chapter 4.

2.1.2 Distributed computing in IoT

Over the decades, the computing paradigm has evolved significantly, moving from centralized mainframe computing to decentralized personal computing that enabled diverse compute models. These computing models serve various users' demands. Traditionally, all data is sent to the cloud node, which has vast computational power and storage to process it. Such a centralized paradigm consumes huge bandwidth and floods the network during data transfer. It is also inefficient for real-time and responsive applications. At the same time, it hardly utilized the underlying distributed resources.

However, distributed computing offers collaborative learning across multiple interconnected nodes. The task is divided into multiple subtasks that can be executed by geographically dispersed devices. Each participating node executes the given task independently to achieve a common goal [49]. In the context of IoT, distributed computing enables data processing at various levels, such as edge computing, fog computing, and cloud computing. Since data is generated and transferred through multiple layers in the network, the distributed computing paradigm can be applied to optimize the resources and meet application requirements. We adopted distributed computing to solve the research questions 1 and 2 in the edge-to-cloud continuum. The continuum is dispersed into three layers of edge, cloud, and fog as shown in Fig 2.1. We have briefly discussed these layers and how they contribute to machine learning training by utilizing the underlying heterogeneous resources.

2.1.2.1 Cloud Computing

Cloud computing offers on-demand services over the internet that include storage, computing, networking, software, databases, etc. It is a pay-as-you-go model with virtually infinite resources. It is equipped with servers, HPCs, and GPUs that make it suitable for complex tasks such as machine learning and big data processing. Cloud computing as a paradigm plays a central role in the facilitation of data processing, storage, and seamless integration of various tools without worrying about hardware or software infrastructure. As shown in Fig 2.1, the cloud nodes receive data from underlying layers and offer an application/visualization-based final output to the user. In the context of the thesis work, the cloud layer works as an aggregation for decentralized training.

At the same time, the cloud node will be used for a resource-intensive task, report generation, and application placement.

2.1.2.2 Fog Computing

To address limitations of cloud computing, the Fog computing paradigm was introduced, which brings computational resources closer to edge of the network. The idea is to extend cloud computing capabilities to the data sources so that it can enable low-latency processing and improve real-time applications. The term was first introduced by Cisco in 2012 as a solution to the growing demand for a computational model that could handle massive volumes of data generated by IoT. It is defined as "a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional cloud computing data centers". Since it is placed close to the edge, it reduces latency, improves QoS, and results in better user experiences. The fog computing paradigm has enabled various applications such as video streaming, smart cities, industrial IoT, autonomous vehicular networks, etc.

The fog layer is equipped with enough resources that can be used for machine learning tasks. Given the compute and storage resources, a fog node can be utilized as a distributed server for model training. Although fog computing is constrained by limited resources, it enables decentralized training with real-time processing capabilities. Additionally, when a fog node is placed in the user's vicinity, it enhances trust for data privacy and security. We used the Fog computing paradigm in federated learning as a client that stores historical data and provides local training.

2.1.2.3 Edge Computing

Edge computing brings computations to the edge of the network. It offers processing units at or near data sources such as sensors, actuators, or cameras. The goal of edge computing is to facilitate real-time application that needs quick responses. The edge computing paradigm minimizes latency and improves the responsiveness of the application. The edge layer mostly consists of data-generating devices that have tiny resources to process it. Hence, the data has to be sent to the upper layer for further processing. Since in the broad picture, edge/fog computing serves the same purpose, some of the work uses both interchangeably as shown in Fog 2.1. However, at the core of the architecture, edge nodes pertaining to devices at edge layers that are data sources

and fog layer offer a cloud-like infrastructure for processing. Hence, as an edge-to-cloud compute continuum, we have used the same notation as edge layer process data, and fog and cloud layer process it.

2.1.3 Machine learning

Machine learning (ML) is a field of study that learns patterns from data without being explicitly programmed. It has a set of algorithms that are used to train a model to uncover hidden patterns/relationships from data. ML algorithms are used to solve complex tasks like classification, regression, clustering, recommendation, summarization, generation, etc. Based on the nature of task, ML algorithms are categorized as supervised, unsupervised, and reinforcement learning. Supervised algorithm needs input data and labels to learn their relationship. Unsupervised algorithms are applied on unlabeled data to organize it into various clusters. Reinforcement algorithms are agent-based learning that make decisions by optimizing rewards/penalties through actions. Based on these, multiple ML algorithms like decision trees, support vector machines, and k-nearest neighbors, logistic/logistic regression, k-means, DBSCAN, Q-learning, etc, are proposed to solve specific problems. Most of the traditional ML algorithms work well for structured and small-scale datasets, which are based on hand-engineered feature selection. Such algorithms often struggle with unstructured data (videos, images, sensor streams) with large amounts of data that have high dimensionality.

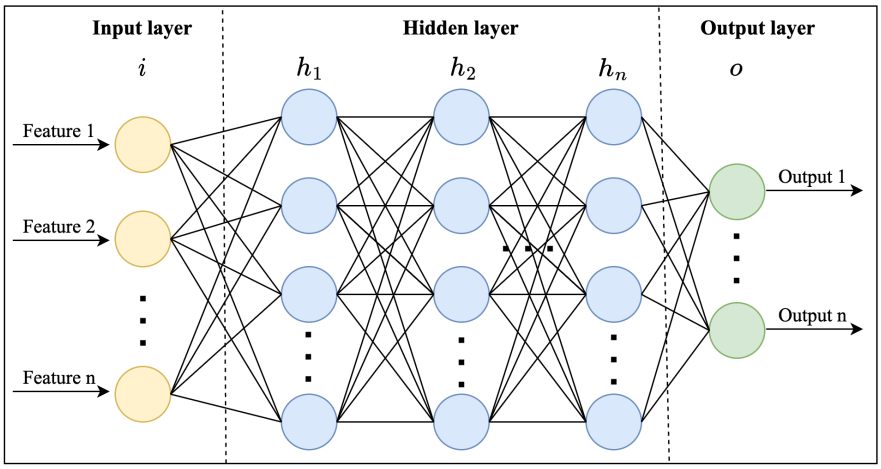


Figure 2.2: Neural network architecture

Neural network training is inspired by the human brain to learn patterns in the data. Similar to the brain, it consists of interconnected neurons that carry weights and biases. The main advantage of neural networks is their ability to learn from unstructured data without the need for explicit feature engineering. As shown in Fig 2.2, a neural network has an input layer used for input data. The number of nodes in the layer is decided by the data dimensions. The nodes in the output layer are used for the final prediction. It is decided by nature of the task. For example, a regression task may have one neuron outputting a prediction value. However, a classification or clustering task may have multiple neurons. Between two layers, hidden layers may have any number of neurons/layers based on the complexity of the network.

A network with one hidden layer is considered a shallow network, and more than one layers are called a deep neural network. For a specific task, such as image recognition, a convolutional neural network is designed that learns spatial features from the data. Every hidden layer learns some abstract/pattern from data, which is finally used across the network to predict the task. For example, in the task of human detection, the initial hidden layer of a neural network may learn to extract low-level features such as textures or edges, the next layer may learns more abstract features such as shape, then the further layer may understand some body part, and so on. Then the Final layer will predict it as a human.

The training is done with the backpropagation algorithm to find a global optimal value. It starts with a random initialization of weights and biases, then it performs a forward pass with input samples. Thereafter, the gradient of the loss function is calculated using the chain rule with respect to each parameter in the network. Finally, the weights of the network are updated by an optimization algorithm such as SGD, Adam, RMSprop, etc. These optimizers adjust weights in the direction that reduces loss of the network. Mathematically, SGD updates weights as shown in Equation 2.1.

$$w^{(new)} = w^{(old)} - \eta \frac{\partial L}{\partial w} \tag{2.1}$$

The training of the network is an iterative process that fine-tunes network parameters over a large number of samples. Neural networks have proved to work exceptionally well on unstructured datasets like images and videos. It can learn non-linearity in the

complex dataset with a varying model. Due to its architecture, it can grow dense and vectorized for efficient implementation.

2.1.4 Federated learning

Federated learning is a distributed machine learning paradigm that enables model training on geographically distributed data without accessing raw data. It is a privacy-preserving decentralized machine learning approach that learn a global model collaboratively over multiple compute nodes. In traditional machine learning, data is collected first on the server then a model is trained on it. However, FL advocates training on a distributed node without accessing raw data. Federated learning was introduced by Google in 2016, as proposed by McMahan et al., to enable decentralized model training across multiple devices while preserving data privacy. Originally, it was implemented to predict the next word for the Google Gboard app. For this, Google trained a model on millions of its users without accessing their personal data. Every participating node has its own personal data, which was used for model training. The training was done on the client side, and only model parameters/updates were shared with the server.

Since then, FL has gained a lot of attention and has been applied to various use case scenarios. Based on a number of participating nodes, FL is categorized as cross-silo and cross-device. In cross silos two organizations participate in training a model to achieve a common goal. So in this setup, there are limited participants but high-quality data with complete resources. For example, multiple banks want to learn about customer behavior. Then they can collaboratively train a model with huge resources without sharing data. However, in cross-device, a huge number of devices (millions/billions) participate in model training. These devices are generally resource-constrained and can only be used for limited processing. All training is done on the distributed node as a client node. The server works as an aggregator that combines all locally trained models into one single global model. The global model is further sent back to clients for further training. The goal of FL is to learn a global representation by learning multiple local representations. It is represented as follows.

$$\min_{w \in \mathbb{R}^d} F(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \tag{2.2}$$

where K is total number of clients and $F_k(w)$ is the local empirical loss at client k. The local model is trained on personal data to fine-tune the model, which is represented as follows.

$$F_k(w) = \frac{1}{n_k} \sum_{i=1}^{n_k} \ell(w; x_i^k, y_i^k) \tag{2.3}$$

Here, $\ell(w; x_i^k, y_i^k)$ is the loss function for the model w on data sample (x_i^k, y_i^k) from client k. Every client performs local training on local data as follows.

$$w_k^{(t+1)} = w^{(t)} - \eta \nabla F_k(w^{(t)}) \tag{2.4}$$

Once the model is fine-tuned locally, it is shared with the server for aggregation. Finally, the server applies the aggregation method on available updates and creates a global model, which is sent back to clients for further training. This completes one round of the training. The server aggregates the locally updated models using averaging over the data samples it is trained on:

$$w^{(t+1)} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{(t)} \tag{2.5}$$

The goal of the training program is to learn global optimal parameters from locally trained models without accessing raw data. Here, every node is assumed to have the same feature space that trains a common model throughout rounds. However, based on requirements, multiple variations of federated learning are proposed. There are broadly three types of federated learning implementation. **Horizontal Federated** learning has the same feature space but different sample space over all clients. Here, one common model is trained on all clients to create a global model. In **Vertical Federated learning**, the dataset has a different feature space but a common sample space. However, **Federated transfer learning** is used when both the sample and feature space are different and there is limited overlap. Hence, transfer learning is used for the adaptation to a new domain. For the thesis, we explicitly focused on horizontal federated learning that has a common feature space. Additionally, our focus is on cross-device learning with a limited number of clients. However, there are multiple challenges that we have addressed in this thesis, which are discussed in later chapters.

2.1.5 Explainable machine learning

Today, AI is also used in critical sectors like the automobile, health sector, energy, etc. One of the key concerns in the adoption of AI models in such sectors is explainability. A machine learning algorithm learns hidden patterns/relationships from given data. Once the model is trained, it predicts/classifies and makes a decision about the new, unknown dataset. However, a model can be biased towards a particular class while making a decision. Hence, a model has to explain why a certain decision has been taken. For example, in the banking loan application, if the loan is rejected, the model should tell about the variables and thresholds that it has considered for this decision. Explainable AI systems can ensure transparency, accountability, fairness, and trustworthiness. Moreover, it helped in rectifying overfitting, bias, and compliance with local laws like GDPR.

Since machine learning comprises a set of algorithms, there are multiple ways to incorporate explainability into it. Few algorithms, such as decision tree, linear regression, are simple and intrinsically interpretable by their design. However, for complex algorithms like Deep Neural Networks, it is hard to interpret. Therefore, post-hoc methods such as SHAP and LIME are often used to explain the reasoning behind their decisions. Explainability in AI models is still an emerging area that continues to evolve, with numerous researchers actively proposing various methods to achieve it. For example, Model-specific methods are proposed to provide rules/regions of interest. These methods generate logical reasoning inherently from their architecture, like decision trees, attention mechanisms, etc. Model-agnostic methods such as SHAP and LIME are proposed to find rules/regions of interest post-model creation. However, counterfactuals are another way to add explainability to the system.

Though SHAP and LIME are the two most popular methods that are used for post hoc interpretability, LIME works with local interpretability. It creates a local surrogate model (usually linear) around a prediction to explain it. However, the SHAP method is based on game theory that assigns importance to each feature for a particular prediction. Basically, it calculates the importance of a particular feature with respect to a final decision. SHAP provides both local and global explanations of the model. Hence, for this thesis, we chose SHAP as a tool to find the importance of a particular feature during model training. It has then extended to a federated learning system to

understand the feature importance during the global model. The details are discussed in chapter 5. SHAP (Shapley) value is an importance value in decision making for every feature. It was introduced in cooperative game theory by Lloyd Shapley in 1953 to calculate fair distribution of total gains in a game. So it calculates the impact of a particular player in the given game. The concept is used in a machine to find feature importance while making a decision. The core idea of the SHAP value is to calculate how much impact a particular feature has when it is used for prediction or not. Hence, it calculates the importance by subtracting the model prediction with or without the feature. Mathematically, it is shown as follows.

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]$$

Where: N is the set of all features, S is a subset of features not containing i,$f_S(x_S)$ is the model prediction using only the features in subset S, $f_{S \cup \{i\}}(x_{S \cup \{i\}})$ is the prediction using the subset S along with feature i. ϕ_i is calculated for all features that are calculated for all permutations of the feature. Hence, it is computationally expensive and may not be suitable for large feature spaces. However, we adopted SHAP to understand how FedAvg converges to global optima and what features contribute to it.

2.2 Literature Review

This section provides a comprehensive survey of recent studies in federated learning with particular emphasis on developments in the edge-to-cloud continuum. It also discusses state-of-the-art around stragglers mitigation and explainability on federated learning.

2.2.1 Distributed computing at edge-to–cloud architectures

Executing data processing and machine learning tasks demands substantial computational power and storage capacity. With the growing proliferation of IoT devices, an enormous volume of data is being continuously generated. One of the effective approaches to manage such tasks is by leveraging cloud computing services. With virtually unlimited resources, cloud platforms are well-suited for handling complex model training on large-scale datasets. The cloud-IoT compute paradigm faces various challenges

such as large bandwidth consumption, longer latency, intermittent devices, heterogeneity, and security [15]. IoT devices are typically limited in terms of computational and storage capabilities. These devices are connected to the cloud via wireless networks. These shortcomings obstruct the smooth execution of the various tasks, specifically the real-time processing. Fog computing emerges as an alternative to the traditional cloud model by positioning computational resources closer to the IoT devices. This has enabled efficient data processing at the fog layer and opened the door to applications such as smart cars, traffic control, smart buildings, real-time security surveillance, smart grid, and many more [104]. The proliferation of distributed IoT data needs real-time processing frameworks and sophisticated algorithms to analyze it. The fog layer has enough resources to store and process raw data for processing. This gives an advantage over a cloud for processing and decision-making locally [128]. Multiple researchers have discussed the role of fog computing in IoT and its applications. Due to computing localization, it has improved the usability of delay-sensitive and real-time applications. Fog computing responds in fractions of seconds, whereas cloud typically takes few seconds. Thus, processing and analyzing raw data distributively improve the applications that provide seamless user experiences. It ensures low latency, network congestion, efficiency, agility and security [20]. This enabled efficient data processing at the fog layer and opens door to various applications such as smart cars, traffic control, smart buildings, real time security surveillance, smart grid, and many more. Since fog layer brings computation close to the edge of the network, this gives an advantage over the cloud for processing and decision making locally.

Bonomi et al. [12] have discussed about the role of fog computing in IoT and its applications. Fog computing provides localization that acted as a milestone in delay sensitive and real time applications. Data analytics on real time data have various applications based on context. Some of them, such as detection or controlling, need quick response typically in milliseconds or sub seconds, whereas other applications like report generation, global data mining tasks are long term tasks. Fog computing and cloud computing can interplay operations to achieve big data solutions. Fog responds to the real time processing task, which can be geographically distributed, and cloud computing proceed with big data analysis or knowledge consolidation. Due to proximity, fog computing is beneficial for delay sensitive applications, but it may lose importance when it gets congested. The number of jobs received at a particular fog

node at a specific time may be higher, which cannot be processed quickly due to limited resources. Al-khafajiy et al. [5] have proposed a fog load balancing algorithm to request offloading that can potentially improve network efficiency and minimize latency of the services. The fog nodes communicate with each other to share their load optimally, which improves the quality of services of the network. Similarly, Srirama et al. [106], have studied utilizing fog nodes efficiently with distributed execution frameworks such as Akka, based on Actor programming model. The framework leverages Akka's concurrency and fault-tolerance features to partition tasks across fog nodes that minimizes latency and network congestion. In this context, it is also worth mentioning that scheduling of applications/tasks in cloud and fog has been studied extensively in the last few years, which is summarized in the related work of Hazra et al. [30].

The cloud–fog computing paradigm offers a hierarchical infrastructure that enhances scalability and computational efficiency for distributed machine learning tasks. However, to meet the latency and privacy requirements of real-time applications, an edge-centric computing framework becomes essential that enables localized processing and decision-making closer to data sources. Munusamy et al. [80] have designed a blockchain-enabled edge centric framework to analyze the real time data in Maritime transportation systems. The framework ensures security and privacy of the network and exhibits low latency and power consumption. Distributed machine learning training offers parallel data processing over the edge of the network. Kamath et al. [44] propose a decentralized stochastic gradient descent method to learn linear regression on the edge of the network. The work utilizes distributed environment to train regression model using SGD. The method process data at device level and avoids sending it to the cloud. A secure and platform-independent integration of cloud-fog IoT framework named FogBus [113] is proposed for the execution of various latency-sensitive applications. The framework is lightweight, which is more responsive and efficient for on-demand services. It offers platform independent interfaces to IoT applications and computing instances for execution and manages multiple application on underlying resources.

Cloud-fog and IoT model is being used extensively for ML tasks such as detection, clustering, and predictions. The framework provides parallel execution environments for shared goals. Srirama and Vemuri [107] have proposed distributed fog framework(CANTO) for training neural network tasks for IoT applications. The proposed framework can efficiently train a model on various nodes in a swarm using actor model.

This enables the fog network to train a neural network model over resource-constrained devices. Taneja et al. [110] have used distributed fog framework for lameness detection in real-time. The paper provides end-to-end solutions for lameness detection in cattle. Pedometers at the edge layer produce raw data which is processed at fog layer periodically. The processed data is shared with cloud node, which analyses it to produce lameness detection in the cattle. The detection results are 87% accurate and three days early detection before lameness is visible to humans. Xu et al. [120] have applied Lie group machine learning to fog computing environment to improve the computational efficiency and robustness of the system. Abeshu et al. [3] have discussed the application of deep learning on the fog environment. The authors discuss the need for distributed machine learning for intrusion detection.

2.2.2 Federated learning in IoT environments

The existing cloud-fog IoT frameworks can be broadly classified into two types. The first focuses on centralized learning and then deployment of a specific application over distributed environments, while others offer distributed data processing/learning over the edge of the network. The thesis focuses on decentralized training on distributed data over edge of the network. McMahan et al. have proposed a decentralized machine learning algorithm named federated learning [76] that can train a machine learning model on distributed data. The cloud-IoT training model is utilized extensively to make tangible applications on distributed data. As fog computing comes into the picture, the training model is further extended to speed up and efficient machine learning tasks. The integration of fog framework and decentralized training algorithms has the potential to dig deeper into the data and provide futuristic applications. The training is further improved using hierarchical federated learning by introducing an additional layer between the cloud and computational node. This improves communication cost, which is necessary for remote server-client communication [4, 73, 81].

Federated learning is another decentralized ML technique that trains models using very large set of low resourced participating devices [47, 76]. The proposed federated method collaborates with various participating devices to create a global ML model on decentralized data. Federated learning is done on low constraint devices that need efficient training strategies for uplink and down ink communication. Konečný et al. [48], talks about efficient communication between cloud and devices. The authors have

suggested sketched and structured updates for server communication that reduce the amount of data sent to the server. Hard et al. [28] have trained a Recurrent Neural Network (RNN) language model for next-word prediction on the mobile keyboard. The model is trained with federated learning using Federated Averaging (FedAvg) algorithm.

The fog-cloud architecture is well suited for distributed machine learning training. Li et al. [62] have used cloud-fog architecture for secure and privacy preserving distributed deep learning training. The local training is given at the fog layer then it coordinates with the cloud server for aggregation. Additionally, it uses encrypted parameters and authentication of valid fog node to ensure legit updates. The central node works like a master node for information consolidation. It synchronizes the training from various devices. Due to stragglers or mobility of devices such as vehicles, drone the synchronous update creates difficulty in training. Lu et al. [70] propose asynchronous federated training for mobile edge computing. The training is done similar to federated learning, but global model aggregation is done asynchronously. To ensure the privacy and security of the shared model, it adds noise to the parameters before sending it to the server. Luo et al. [73] have proposed a hierarchical federated edge learning framework to train low latency and energy-efficient federated learning. The framework introduces a middle layer that partially offloads cloud computational work. The proposed 3-layered framework aggregates model parameters at both fog layer and cloud layer while training is done at the remote device. Fog enabled federated learning can facilitate distributed learning for delay-sensitive applications.

Jiang et al. [42] have proposed model pruning-based federated learning architecture for resource-constrained devices. It reduces complexity of the model through multistage pruning. The results show that it achieves similar accuracy in relatively less time. With federated learning training on fog environment, the paper has trained a deep learning model for cyber attack detection on fog nodes. Zhang et al. [125] have proposed IoT based platform named FedIoT for federated learning training on resource-constrained devices. The authors have proposed an adaptive optimizer-based FL training algorithm, FedDetect, with cross-round learning rate scheduler. The proposed FedDetect is trained to evaluate anomaly detection over FedIoT platform. Mills et al. [77] worked towards minimizing number of rounds and size of model parameters for communication efficient federated learning that supports wireless edge intelligence. To address this they

proposed CE-FedAvg algorithm that optimizes local training with adaptive like adam that uses momentum to converge faster. Furthermore, it proposed a novel scheme for the quantization of weights and momentum. Before sending the data to the server, the client quantized and sparsity the model parameters that reduce model weights. Hence, a combined impact of training rounds and model compression achieved 1.7 times less training time and around 3 times less communication compared to baseline FedAvg.

Training machine learning model on resource-constrained devices suffers various challenges, such as storage, computing facility, and failure-prone systems. Imteaj et al. [38] have discussed challenges in implementing the federated learning approach for low-resourced devices. This includes device heterogeneity, privacy, communication overhead, memory/compute constraints, energy efficiency, fairness and scalability. To this end, various research studies have been done that deal with such issues. The survey suggests that the state-of-the-art implementation of the mentioned issues can be used in edge federated learning training. Liu et al. [67] have proposed fog based secure federated learning approach. The framework is focused on reliability of the model's input with secure aggregation, and on request-broadcast modules to deal with dropout. Chen et al. [17] have utilized federated learning framework for bearing fault diagnosis. The framework prioritizes local parameters based on their contributions during aggregation. Pinyoanuntapong et al. [86] have tried to optimize the cost of federated learning over multihop model transfer. Single-hop cellular communication needs a high-speed internet core to manage the communication, which is more expensive and may not be feasible for general commuting. Hence, multi-hop communication is the backbone of communication in WAN too. Therefore, the authors have thought to use multi-hop federated learning that can minimize communication costs. However, it suffers from noise and interference due to communication that may lead to nomadic FL model updates. Other than that it suffers from slow convergence and prolonged updates. The paper tries to study the impact of wireless communication on the convergence rate of an FL model. It uses reinforcement learning to make routing decisions that minimize communication costs. The proposed FedAir was implemented and tested on a physical wireless mesh network. They show that using MARL-based routing greatly improves FL convergence speed compared to standard mesh routing protocols. The work cab be a significant step toward democratizing FL for use in underdeveloped regions, dis-

aster recovery, and large-scale urban networks, where traditional single-hop FL is not feasible.

Traditional FL, based on a centralized aggregation server, might suffer from a malicious user attack or failure due to physical damage, which significantly degrades the performance of FL. The aggregation server can be attacked by an outsider who is not participating in the learning process or one of the end-devices participating in the learning process. A malicious aggregation server can infer the end devices' sensitive information from their learning model parameters. Therefore, there is a need to address the privacy leakage challenge of FL. To address this, multiple researchers have worked on different implementations of Federated learning apart from centralized star topology, such as distributed, mesh topology, or gossip-based approaches [23, 32, 79, 94]. Khan et al. [46] have proposed dispersed federated learning (DFL) that supports distributed federated learning. The paper proposes two ways of aggregation. First, Centralized DFL is a kind of hierarchical FL that trains a sub-global model in the lower layer, and the central aggregator aggregates at the higher layer. However, in the second approach, Distributed DFL, it aggregates the local model on various distributed nodes locally. The idea of dispersed learning is to provide security from attacks and single-point failure. However, there is a trade-off between convergence rate and the number of sub-global iterations in DFL. It also presents a taxonomy based on the approach of aggregating sub-global models to produce a global model as a parameter, along with several future research directions related to DFL.

Geo-location data processing can also be done efficiently with fog architecture. Saha et al. [93] have proposed fog-assisted federated learning (FogFL) for distributed learning using resource-constrained devices. The edge layer and fog layer participate in federated learning, while the cloud layer steps in heuristically for global model aggregation. The entire idea is to train a distributed ml model using resource constrained devices with minimum communication overhead. Additionally, since there are multiple fog aggregators, it is resilient to single node failure. FogFL reduces energy consumption and communication overhead significantly. This training is done on geographically distributed network that optimizes communication latency by 92% and energy consumption by 85%. Stergiou et al. [108] have used federated learning for traffic sign image recognition. A LeNet model is trained collaboratively to support autonomous

vehicles. The proposed work decouples clients from training a local model and communicating with the server. The implementation suggests a container-based solution to process huge datasets.

A major advantage of the IoT architecture is its ability to support hierarchical computing where computational tasks are distributed across multiple layers. This enables hierarchical federated learning that optimizes communication, latency, energy and scalability. It also prevents model training from a single point of failure as it has multiple fog servers for aggregation. The idea is to perform model training at the edge/fog layer repeatedly, and the cloud is used periodically for global knowledge generation [22, 41, 127].

2.2.3 Asynchronous communication in FL

Federated learning follows an iterative training paradigm that assumes the consistent availability of client devices during the learning process. However, achieving such an ideal scenario is often challenging in real-world settings, especially in environments with limited computational and communication resources. In such a network, devices are bound to fail/struggle, needing different scheduling and aggregation strategies [34]. Zhu et al. [132] introduced FLACOS, an asynchronous federated learning (AFL) framework aimed at addressing the issue of straggling clients. Their proposed approach incorporates adaptive client selection to minimize training latency while ensuring client availability and promoting long-term fairness in participation. The work tackles three critical challenges (i) unknown channel state information and client computational capabilities (ii) dynamic client availability due to connectivity/energy constraints (iii) equitable participation across devices. FLACOS uses Multi-armed bandit optimization for online client selection, Lyapunov virtual queues to enforce fairness via a credit-based system, and first-come, first-served for asynchronous aggregation to eliminate communication bottleneck. In a similar way, Ma et al. [74] proposed a semi-asynchronous federated learning framework, FedSA, for distributed edge computing environments to address edge heterogeneity, non iidness, and resource constraints. The framework aggregates a subset of local updates as they arrive and proceeds with the training process for the subsequent round. To improve training accuracy on non-IID data, FedSA deploys adaptive learning rates based on their participation frequency.

Nguyen et al. [82] identified scalability challenges in federated learning as the number of participating clients increases to several hundred. Additionally, they highlighted privacy concerns associated with asynchronous aggregation, particularly in single-node aggregation scenarios. The authors introduced FedBuff, a novel asynchronous federated optimization framework that employs buffered asynchronous aggregation. The framework aims to enhance scalability while ensuring privacy against an honest-but-curious adversarial model, incorporating secure aggregation techniques alongside differential privacy mechanisms. Training in real-time is another challenge in the network. Federated learning can be utilized to process real-time data in a distributed way to improve the driving experience and serve quality. But due to high mobility and uncertainty of vehicles, it is a challenging task. Liang et al. [63] proposed a semi-synchronous federated learning framework, Semi-SynFed, to enhance performance in Internet of Vehicles environments. The framework initiates client selection based on computing capability, network bandwidth, and the learning value of local training samples. To address the heterogeneity of clients, a dynamic waiting time mechanism is introduced, allowing the server to adaptively wait during each training round. Finally, a dynamic aggregation strategy is employed to integrate client model updates in an asynchronous manner. Chen et al. [19] have proposed a federated learning-based asynchronous communication scheme for edge devices. The work addresses key challenges of resource constraints and communication issues on edge devices. It characterizes computational resources in an unstable heterogeneous network using a 0-1 knapsack strategy. Then it selects available computing nodes based on communication conditions so that the learning process can be completed more efficiently. The study also implements federated learning on edge devices using a lightweight node selection mechanism to execute learning tasks effectively.

Lu et al. [69] proposed FedAAM, an adaptive asynchronous federated learning framework designed to address client heterogeneity and unstable convergence in asynchronous settings. The method introduces a dynamic weighting algorithm that categorizes clients into fast, moderate, and slow tiers, allowing adaptive contribution control based on recent model updates. Additionally, it integrates a global momentum mechanism to mitigate the impact of outdated local updates and to accelerate convergence. They evaluated the framework using convolutional neural networks on MNIST, Fashion-MNIST, and CIFAR-10 datasets under varying client participation

scenarios. Experimental results suggest that FedAAM outperforms FedAsync and FedNova, in terms of both accuracy and training efficiency across non-IID and dynamic environments. Similarly, Zhou et al. [129] proposed an Adaptive Segmentation-enhanced Asynchronous Federated Learning (AS-AFL) framework to tackle client heterogeneity and communication inefficiencies in large-scale intelligent transportation systems. The core idea involves a meta-learning-based adaptive segmentation strategy that clusters clients based on network conditions and task similarity. Within each segment, model updates are aggregated synchronously, while across segments, an asynchronous inter-group aggregation is employed. The framework operates in a fully decentralized manner using a gossip-based protocol, eliminating the need for a central server. Additionally, a pruning-based optimization method is used to refine tasks within segments, ensuring efficient and adaptive learning across dynamic vehicular environments.

A general way to handle stragglers in federated learning through asynchronous communication and modified aggregation. Wang et al. [114] proposed a decentralized asynchronous federated learning framework for edge devices, leveraging blockchain and IPFS. It is consensus-based training that eliminates the reliance on a central server. The proposed FedDgc method introduces a dynamically growing cache mechanism to control model aggregation based on training progress. Hence, it reduces redundant gradient exchanges and improves convergence under non-IID conditions. The framework integrates staleness-aware weighting and data-volume-based adjustments during aggregation to ensure fairness and robustness. This fully decentralized design enables edge devices to participate proactively in training, enhancing scalability and resilience in dynamic and heterogeneous environments. In another approach, Zhou et al. [130] proposed an Efficient Asynchronous Federated Learning (EAFL) framework to handle heterogeneity in edge environments. Their method uses gradient similarity-based dynamic clustering to group clients with similar behaviour, enabling smoother aggregation. The framework adapts over time by dynamically re-clustering clients based on staleness and data distribution, making the training process more stable and efficient.

Most existing state-of-the-art approaches work under the assumption that either the data-generating IoT devices possess sufficient computational resources for training machine learning models or that fog nodes have access to a complete dataset. A general approach for handling stragglers is to selectively use devices that promise timely results. Another approach, explored by several researchers, involves grouping clients and

applying a hybrid model that combines both synchronous and asynchronous training. In contrast, this thesis focuses on a smaller network of resource-constrained devices that continuously generate data at the edge. In such settings, conventional on-device training becomes inefficient due to the dynamic and ever-growing nature of the dataset. Therefore, there is a need for a framework capable of leveraging limited-capacity devices for model training. The framework must address the emerging computational and distributed challenges inherent in modern IoT environments.

2.2.4 Straggler mitigation techniques in FL

Alternative to centralized training, Federated learning is introduced to train a machine learning model on the distributed device that advocates data privacy. A model training needs a huge amount of data that may be available over geospatial locations. Depending upon scale, participant, and model, the paradigm has been applied to two possible settings: cross-silo and cross-device. Cross-silos have a limited number of participants, mostly organizations with dependable infrastructure and datasets [35]. However, cross-devices come with small computing devices, such as mobiles, tablets, and PCs, but a larger number of participants (millions). So, it also has various challenges, such as heterogeneity, resource limitations, synchronization, standardization, trustworthiness, etc., that must be handled during training. Federated learning for IoT is being extensively studied with various applications such as smart homes, health care, industry 4.0, smart cities, autonomous driving, etc [16][124]. Since data is distributed over multiple nodes, on-device training over distributed nodes can potentially train a generic model.

In a classic FL setup, the server has to wait until the last update is received. Therefore, an Asynchronous Federated Learning (AFL) approach is adopted to ensure the efficiency of the training [33, 117]. However, scalability in AFL is still an issue, e.g. AFL beyond a hundred nodes produces a diminishing effect over training speed. To address this, buffered asynchronous aggregation, FedBuff is proposed that jointly optimize synchronous and asynchronous federated learning. The server processes with aggregation only if K numbers of updates are available [82]. Similarly, multiple variations of AFL have been proposed to reduce training bottlenecks [121]. Still, stragglers in the network create further bottlenecks in model convergence. The heterogeneity of the data and resources significantly impacts the training time and model accuracy. Additional, non-IIDness of the data distribution further impacts model convergence. Chai

et al. [13] have proposed a tiered-based approach, TiFL, to divide clients into multiple groups. To mitigate the stragglers, TiFL grouped the available clients by their computational efficiency and training latency. The selection of the clients is made from the same tire, reducing the wait time for the server that improved performance by 3 times over conventional FL. In [60], a Hybrid Federated Learning (HFL) is proposed to deal with stragglers impact. The paper utilizes both synchronous kernal and asynchronous updater for both non-straggler and straggler scenarios. Basically, synchronous update is performed for fast devices, and an asynchronous strategy is made for stragglers. In a similar way, FedAT[14] tries to combine the bridge between synchronous and asynchronous federated learning to minimize straggler's impact. FedAT applies intra-tier synchronous learning with cross-tier asynchronous learning with tiered-based weighted averaging during aggregation. Reisizadeh et al. [88] have proposed a straggler-resilient algorithm FLANP that adaptively selects clients to speed up the training. FLANP starts training with faster nodes, and stragglers are involved after threshold accuracy is reached. In general, grouping is one of the popular approaches to dealing with stragglers and heterogeneity [21, 123]. These works focus on categorizing devices into two groups: slow and fast. Thereafter, training is done in their groups. However, some other approaches, such as communication-based strategies have also been studied to deal with stragglers and their impacts [6].

PedFA [65] is a partial model aggregation strategy to deal with stragglers in the network. The server waits for a limited number of devices for the updates in each round. However, only fixing the number of devices is not enough, but it needs to be carefully chosen. So, FedFA has proposed a waiting strategy for an adaptive number of devices for aggregation. In the process, the stale updates are also included, but the impact of it has not been studied. Sultana et al. [109] have proposed an elastic optimization method, FedEN, to deal with stragglers and data heterogeneity in an IoT network. The proposed method is a balanced approach of lasso and ridge penalization of the local model that specifies the model updates. The paper has addressed stragglers caused by statistical heterogeneity distribution, not the computational one. EAFL [40] has used a computation offloading approach to reduce the straggler's impact. A part of the computation is offloaded to the nearest edge server in the case of a straggler's scenario. Even though stragglers are accommodated with asynchronous or tier-based approaches, the impact of stale updates needs to be studied [18].

With various experiments[51], we found that stale updates can impact training significantly if aggregation is not performed optimistically. So aggregation process needs to be modified in case of stragglers. Zhang et al. have proposed a staleness-aware asynchronous stochastic gradient descent method to minimize straggler's impact on global model convergence. The authors have modified the learning rate of the old updates by dividing it by the stale value [126]. Similarly, Zhu et al [131]. have proposed a staleness-tolerant asynchronous federated learning, STAFL, approach to deal with stragglers in the network. The STAFL first use an asynchronous update that removes server waiting time and then dynamically changes aggregation parameters based on network weight and staleness.

Liu et al. [64] proposed FedASMU, an asynchronous federated learning framework designed to address system and statistical heterogeneity by dynamic staleness-aware model updates. At the server side, they employed a dynamic weighting mechanism that adjusts the influence of each uploaded local model based on its staleness and corresponding local loss. On the client side, an adaptive local model adjustment method is used, where devices periodically request a fresher global model during local training. This allows the global and local models to remain closely aligned during training, ensuring faster convergence and improved accuracy without waiting for slow clients. The framework is validated on multiple non-IID datasets and model architectures, showing robust performance against baseline methods. Park et al. proposed Sageflow [84], a robust federated learning framework that addresses both stragglers and adversaries. It uses a staleness-aware update mechanism that retains delayed client models for future aggregation, ensuring fair participation. To defend against adversarial updates, it combines entropy-based filtering with loss-weighted averaging, leveraging a small set of trusted public data. This design enables Sageflow to achieve stable convergence and improved robustness in non-IID and attack-prone settings.

Since stragglers are critical in edge cloud model training, this topic has been extensively studied in recent years. Even though it was discarded in the initial works, it is prominent for smaller networks due to the relevance of the data and heterogeneity. Most of the work done deals with incorporating stragglers using asynchronous updates. They are mainly focused on continuing the training by addressing communication challenges with asynchronous communication. However, none of the works is focused on the impact of stale updates on model convergence and accuracy. Additionally, resource

constraints are another concern in federated learning. In this thesis, we tried to understand the impact of stragglers through stale updates. To solve the stale issue, we proposed a weighted averaging scheme, FedStrag, for the delayed updates that can optimize model training. The overall system is trained on resource-contained devices to mimic real applications. The stragglers are controlled by time-bound asynchronous learning, while stale updates are optimized with the FedStrag strategy.

2.2.5 Federated learning frameworks

Federated learning has become a popular paradigm for decentralized training. It offers not only distributed training but also a strong privacy-preserving mechanism that attracts various stakeholders to come together. Hence, there has been significant progress in the development of federated learning software frameworks. These frameworks differ in their focus, architecture, and functionalities for both research experimentation and industrial deployment. TensorFlow Federated (TFF) and Flower are two of the most versatile and widely adopted frameworks in the FL community. Both frameworks are mainly used for research and prototyping. TFF [1] was developed by Google, which provides a layered architecture with high-level APIs for standard FL workflows and lower-level APIs for custom algorithm development. It supports simulation, which makes it suitable for experimenting with new FL algorithms on a large scale. Similarly, Flower [10] is framework-agnostic and supports a wide variety of machine learning backends such as PyTorch, TensorFlow, and Scikit-learn. It supports both simulation and on-device training for real-world edge devices. Both frameworks support synchronous communication, which makes it hard to train in a heterogeneous network. However, the framework can be modified to provide asynchronous training.

NVIDIA FLARE and IBM federated learning are designed with industry and enterprise deployment in mind, which mainly focuses on security, scalability, and integration with existing enterprise infrastructure. NVIDIA FLARE [83] is a domain-agnostic, production-ready SDK that supports multiple ML frameworks. It also provides advanced privacy-preserving features such as differential privacy and homomorphic encryption. Similarly, IBM federated learning [37] is enterprise-focused and offers a framework-agnostic platform to support various learning topologies. It is highly configurable for deployment across data centers, private clouds, and edge devices. IBM FL supports various ML algorithms such as neural networks, tree-based algorithms,

K-means, etc, on both PyTorch and TensorFlow. Both frameworks are developed on an industrial-grade implementation of existing algorithms to support real-world applications.

PySyft and FATE both emphasize privacy and security, but approach it from different angles. PySyft [92] was developed by OpenMined, which focuses on privacy-preserving techniques such as secure multi-party computation (MPC), differential privacy, and encryption. It is a PyTorch-based framework for federated learning. It is particularly popular in academic and healthcare research where data confidentiality is of high priority. It supports simulation, on-device, and real-world training with asynchronous learning capabilities. FATE (Federated AI Technology Enabler) [66] is an industrial-grade framework that implements secure computation protocols based on homomorphic encryption and MPC. It is widely adopted in finance and healthcare for collaborative analytics, supporting both simulation and production deployment on edge and cloud infrastructure. It provides an ecosystem to develop and deploy an FL solution using various ML algorithms other than neural networks.

FedML and FedScale are developed by researchers who focus on bridging the gap between research and real-world deployment. It is a simulation-based approach that involves large-scale experimentation. FedML [31] provides an end-to-end platform that includes simulation tools for research and MLOps infrastructure for production. It supports a broad range of devices and ML frameworks, and is particularly adept at handling edge-to-cloud workflows with both synchronous and asynchronous aggregation. FedScale [54] is designed for benchmarking and evaluating FL algorithms at scale. It offers a comprehensive collection of FL datasets for evaluating different aspects of real FL deployments. The architecture is well-suited for studying FL performance in heterogeneous, large-scale environments.

OpenFL and FEDn are frameworks that focus on secure, flexible, and scalable deployments in sensitive and enterprise environments. OpenFL [2] was developed by Intel, which offers confidential computing to enable secure training and collaborative analytics across organizations without exposing raw data. It is used in sectors like healthcare and finance where data sensitivity is a major concern. FEDn [99] is a cloud-native, enterprise-ready FL framework designed for massive scalability and resilience. It supports popular ML libraries like PyTorch and TensorFlow. It provides

easy deployment across private cloud and edge environments. Apart from these, numerous research-driven frameworks have been proposed to address specific challenges in federated learning. As a result, several frameworks have been developed, each offers specialised solutions to the distinct needs of both research and industrial domains. Frameworks like TFF and Flower are preferred for experimentation and flexibility, while NVIDIA FLARE, IBM FL, OpenFL, and FEDn are oriented towards secure, scalable, and production-ready deployments. Privacy-centric frameworks such as PySyft and FATE address the needs of sensitive domains, and platforms like FedML and FedScale excel in large-scale and benchmarking studies. Most of the frameworks assume having full resources for training, and very few have a strategy for straggler mitigation. Hence, there is a need for a framework for the implementation of federated learning on the edge-to-cloud continuum. The framework should address key challenges of heterogeneity, stragglers, and real-time processing.

2.2.6 Convergence analysis of federated learning

In Federated learning, the FedAvg algorithm is a widely adopted aggregation method that enables a global model to merge intelligence from various data sources. As a distributed computing paradigm, it also works with partial client participation that accommodates intermittent connectivity. FedAvg performs a simple weighted averaging rule that does not require any second-order information. That makes it a simple yet powerful strategy for knowledge aggregation. However, this raises a central scientific question: under what formal conditions does FedAvg converge and at what rate? How does it converge over time? Multiple researchers have studied the working behaviors of federated learning and tried to understand its convergence.

Li et al. [59] have presented the theoretical analysis of FedAvg's convergence. Under the assumptions of L-smoothness, strong convexity, bounded variance, and uniform gradient, the authors have proved that FedAvg converges at a rate of O(1/T) with T total SGD steps. However, they showed that data heterogeneity slows convergence. A highly non-IID client data causes conflicting model updates and drifts that push the model away from the global optimum. However, for training on non-IID data, decaying learning rates are necessary for true convergence. Hence, for real-world applications, FedAvg has been further modified to address these issues. FedProx [58] is designed to tackle heterogeneity in the federated network. It adds a proximal term to each client's

objective, which stabilizes updates under heterogeneity and yields provable convergence guarantees. The result suggests FedProx achieves significantly more stable and accurate convergence than FedAvg on realistic heterogeneous datasets. Similarly, methods such as SCAFFOLD [45], FedNova [116] aim to mitigate client drift. These results suggest that FedAvg's convergence hinges on multiple factors such as the learning rate schedule, the degree of data skew, and client participation. Addressing these has been the focus of the recent works.

On the convergence of the FedAvg, data heterogeneity plays a significant role. Non-IID data in federated learning causes client drift that leads to degradation of global model performance and delayed convergence. To mitigate the client drift between local and global models, Yan et al. [122] have proposed FedCSD that provides class prototype similarity distillation to align local logits to global logits. The interplay of these factors defines the trajectory and stability of the global model. Understanding what made the convergence or what hindered it becomes crucial for optimizing FL systems in real-world deployments.

Seo et al. [101] have tried to understand the convergence behavior of federated learning on both IID and non-IID data. The authors have studied how hyperparameters such as the number of clients, local epochs, batch size, learning rates, and partial participation affect FL performance on both IID and non-IID use cases. They probe client-specific loss landscapes to explain client drift. The authors found that in the non-IID case, loss landscapes are different for every client, which makes inconsistent global loss optimization, causing client drift. Experimental findings suggest that careful tuning of batch size, learning rate, and participation can significantly influence performance, particularly under skewed distributions. All of these analysis provides theoretical guarantees of federated learning convergence under various assumptions [25, 61]. However, for real-world scenarios, many of these assumptions do not hold. Therefore, we look into explainability-based convergence analysis of the training paradigm.

2.2.7 Explainable AI method in the federated learning

Ribeiro et al. [89] have investigated explainability techniques, SHAP and LIME, in federated learning for regression models to enhance transparency and reliability in industrial settings. These techniques highlight influential variables and provide insights into AI decision-making essential for maximizing benefits. The authors tried to find

how specific variables influence a regression neural network's predictions using SHAP and LIME. Awosika et al. [9] have integrated federated learning and xAI to detect financial fraud for banking so that privacy and transparency can be ensured. The authors trained a neural network with 93% accuracy, whereas SHAP is used to provide feature importance. Tastan et al [111] have proposed ShapFed, a contribution assessment method that uses Shapley values for a granular understanding of class-specific influences of participants in FL. Here, instead of evaluating overall accuracy, ShapFed calculates cosine gradient Shapley values by focusing on the directional alignment of gradients/parameters of only the last layer of the classifier. Based on this, they have used weighted aggregation method that assigns dynamic weights to participant updates. The work aims to overcome the limitations of conventional federated averaging, specifically in non-IID and imbalanced data scenarios for better global model convergence.

A detailed survey of federated learning and explainable AI is presented in [68]. This paper points out that explainable AI is now being added more and more to federated learning methods. The goal is to make FL more secure from harmful or malicious participants. It also observes that most current work focuses on horizontal FL, and not many researchers use well-known FL libraries. Kalakoti et al. [43] have studied federated Learning and explainable AI framework for deep learning-based intrusion detection in IoT networks. The work aims to achieve both high detection performance and interpretability while preserving data privacy in horizontal FL settings. Similarly, there are multiple studies that have used SHAP and LIME on federated learning to interpret the black box model with privacy guarantees in various domains such as healthcare, IoT, financial fraud, vehicular network, and others [11, 53, 85, 90].

While multiple studies have been conducted to integrate federated learning with explainable AI. Most of the work is presented in two ways (i) xAI methods are used to interpret the model output for various application scenarios. (ii) xAI methods are deployed to find clients' contributions for better aggregation in terms of fairness or transparency. Our proposed research takes a novel approach by specifically linking Shapley values to the analysis of FL convergence behavior. Although there are multiple studies that provide convergence analysis of federated learning, they are deeply rooted in theoretical bounds and empirical observations of loss reduction and model parameter changes. They show that non-IID data can slow down convergence and that a decaying

learning rate is often necessary. However, their works do not provide a granular feature-level understanding of why the model converges in a certain way, or which features are driving or hindering the convergence across different clients or rounds. Our work aims to fill this critical gap by examining the evolution of feature importance with SHAP throughout the training process. We can gain insights about how different features contribute to the global model's performance and how their contributions change as the model converges under various heterogeneous conditions. This will reveal whether convergence is mainly driven by a small number of important features or all features are contributing equally. Also, it shows whether there is any correlation between model convergence and feature impact, and how this pattern develops during training.

2.3 Summary

In this chapter, we presented the background of the key concepts used in the thesis. It discussed various computing paradigms for processing data in IoT networks. We also examined the layered architecture of IoT systems and their computational capabilities. We discussed both centralized and decentralized machine learning approaches, highlighting how data is trained in each setting. Furthermore, we explored the concept of explainability in machine learning models, with particular emphasis on SHAP. The chapter also examined state-of-the-art research on federated learning and its deployment across the edge-to-cloud continuum. It surveyed the literature on synchronous and asynchronous federated learning, including strategies proposed by various researchers to mitigate the impact of stragglers. Finally, we discussed the convergence analysis of the federated learning paradigm and summarized how different studies have approached and evaluated it.

Chapter 3

Federated Learning Framework for Edge-to-Cloud Continuum

This chapter discusses federated learning and its implementation on the edge-to-cloud continuum. It explores the possibility of training a neural network on resource-constrained devices. The objective is to find the feasibility of training a model that can harness distributed resources, off the cloud. With this, we aim to create a framework that supports continuous learning in an IoT network. The chapter is focused on finding solutions for research goal 1. For this, we first designed an architecture for distributed model training on resource-constrained devices. Thereafter, we extended the work to create a framework for federated learning using Raspberry Pis. The framework and architecture are tested on a real-world use case for an industrial IoT scenario.

3.1 Introduction

In the era of pervasive computing, the convergence of federated learning and distributed training with low-resource devices has redefined edge analytics. With the exponential growth of connected devices and data-intensive applications, the amount of data has increased exponentially. The data are being generated through various devices, such as sensors, cameras, etc. Each individual device produces data that contains valuable and actionable insights. The data can be used to achieve a wide range of tasks such as business analytics, security monitoring, informed decision-making, and personalized recommendation systems. This distributed data is sent to the cloud for processing and

model training. Since data has to travel to the cloud, it consumes huge bandwidth, creates network congestion, and requires powerful devices to process big data. The traditional cloud-centric training is also inadequate for real-time data processing requirements. Moreover, data privacy and security are of the utmost concern in cloud-centric computing. Therefore, a user's restraint from sharing data to the cloud. Effectively processing such vast and geographically dispersed data necessitates the adoption of advanced and distributed training paradigms. In this context, machine learning serves as a powerful tool to uncover meaningful patterns and extract knowledge from the continuously growing data.

Extracting meaningful insights from large-scale datasets demands significant computational resources. However, devices placed at the edge layer are resource-constrained, which makes it difficult to train a machine-learning model with the entire dataset in one go. The fog computing paradigm offers a viable alternative by providing computational resources closer to data sources, therefore mitigating latency and bandwidth issues. Raw data is generated across geographically distributed edge nodes that should be processed for various analytical tasks, including classification, clustering, and regression. Since the nodes at the fog/edge layer are low-resourced, it is practically hard to train a model on these devices. The issue is further aggravated when the data continuously grows at the fog/edge layer. Hence, processing and analyzing this ever-expanding data in real time remains a critical research problem that needs to be solved.

Furthermore, data privacy and security concerns impose significant limitations on the data collection and processing. Federated Learning addresses these challenges through a decentralized learning paradigm that allows model training to be performed directly on client devices, thereby eliminating the need to transmit or store raw data at the central server. In an FL setup, each client trains a local model on its private data and shares only model parameters/updates with the server. The server coordinates and aggregates these parameters to construct a global model iteratively. This collaborative process continues until a generalized model or expected result is achieved. Despite its advantages, federated learning performs optimally under stable network connectivity with sufficient computational resources. However, in scenarios like resource-constrained networks or intermittently connected devices, especially in small and heterogeneous networks, several challenges arise, such as unreliable communication, the presence of stragglers, and limited processing power. Stragglers, in

particular, slow down the entire training process by delaying their updates, thereby creating synchronization bottlenecks.

A distributed computing paradigm processes data at the edge of the network without collecting extensive data at the server. Federated learning is a decentralized machine learning paradigm that trains a global model without accessing raw data at the server. Hence, we integrated these paradigms to create a framework for distributed training with fog computing, edge computing on distributed nodes. The key idea is to bring intelligence to the edge of the network that can efficiently process data at various places, harnessing underlying resources.

3.2 Motivation

Machine learning model training relies on computationally intensive infrastructures such as cloud servers, high-performance computing (HPC) clusters, and GPUs. Conventionally, data is generated at distributed nodes and then sent to the cloud for training. This causes network flooding and huge bandwidth consumption. Alternatively, the Edge/Fog computing paradigm is opted for distributed data processing. Moreover, many edge devices that generate data are inherently constrained in terms of processing power, memory, and energy capacity. These limitations raise pressing challenges and research questions, such as, Can distributed data be processed and analyzed efficiently using decentralized and parallel learning techniques? Is it feasible to train ML models collaboratively across resource-constrained devices without flooding the network or compromising performance? How can decentralized learning systems remain robust in case of unreliable connectivity, straggling clients, and intermittent node failures? How to deal with an unreliable network and failure-prone nodes? How to synchronize the training if there are stragglers? Furthermore, as data continues to be generated at an unprecedented rate, there is a growing need to address how continuous data streams can be effectively managed to extract actionable insights in real time. So, rather than centralized training, can we utilize various IoT devices to train a decentralized model using distributed data?

While centralized approaches provide high computational throughput, they do not leverage the growing capacity of edge resources. Federated Learning emerges as a promising solution by enabling decentralized model training across distributed nodes

without requiring raw data to be centralized. However, synchronous federated learning (SFL) paradigm often suffers from performance bottlenecks due to unreliable connections, slow or straggling clients, and heterogeneous computational capabilities across participating devices. Additionally, the high velocity of data generation introduces new challenges for real-time analytics, especially when operating within low-resource environments. These are some of the major challenges in low-resourced distributed learning.

Advances in IoT technologies have provided sufficient computational resources to the fog/edge layer. The computational facilities at the edge of the network can significantly impact distributed learning. Even though the centralized training paradigm is capable of efficient training, we need a distributed system to harness the efficiency of low-powered devices. Federated learning training has the capability to learn a machine learning model collaboratively using distributed datasets without sharing the data with the cloud. However, there are bottlenecks due to stragglers or unreliable networks in synchronous federated learning (SFL). Further, the high data generation rate opens new research challenges to data analytics in low-resource devices. Most existing FL frameworks assume the presence of resource-rich clients and stable network infrastructure, which limits their applicability in constrained environments. Therefore, there is a need to design and develop novel frameworks that can harness the computational potential of low-powered, heterogeneous IoT devices for continuous, privacy-preserving, and efficient decentralized learning. Furthermore, concerns surrounding data privacy and user trust impose additional constraints. Therefore, data privacy and users' trust are other dimensions that need to be examined while processing personal data. Addressing these challenges holistically provides a direction for research in distributed machine learning under constrained resource settings.

Motivated by the mentioned challenges, we worked in this direction to bring neural network training to the edge of the network. The work directly addresses key challenges of the implementation of federated learning in the edge-to-cloud continuum. We achieved this in phases by answering research questions 1 and 2. Firstly, a fog-enabled architecture is proposed to evaluate the feasibility of model training on a rapidly changing dataset within a resource-constrained environment. This provides a blueprint for federated learning in a resource-constrained environment on streaming data. Then we

extended the work to build a framework, FIDEL, for federated learning that can be executed on heterogeneous fog nodes. The framework is also developed with asynchronous strategies to overcome stragglers and intermittent nodes. Finally, we demonstrated the computational intelligence of FIDEL to solve a real-world problem. The framework is trained for human-safe position detection in human–robot (HR) workspace for industrial applications using both synchronous and asynchronous strategies. The results show that the framework can achieve similar accuracy as compared to centralized and existing frameworks using Raspberry Pis. We will discuss both contributions in the rest of the chapter.

3.3 Fog Enabled Distributed Training Architecture for Federated Learning

This section discusses the proposed architecture that explores the possibility of model training on fog devices. We will discuss methodology, architecture, and evaluation of the proposed architecture for constrained environments.

3.3.1 Fog enabled training architecture

Storing raw data on centralized servers incurs substantial bandwidth usage that leads to network congestion, making it unsuitable for real-time or latency-sensitive applications. Additionally, this approach raises significant privacy concerns due to the exposure of sensitive information. An alternative approach involves collaboratively training models on edge devices without a cloud server. However, this is constrained by the limited computational capabilities of IoT devices, which are typically insufficient for training complex machine learning models independently. To address this, we proposed a fog-enabled online training architecture to simulate a machine learning model training on streaming data by various IoT devices. The proposed architecture is shown in Fig 3.1. Edge nodes continuously generate data and transmit it to the proximate fog node. The fog node, equipped with moderate computational and storage capabilities, can perform local machine learning tasks and retain historical data produced by IoT devices. At a higher level, the cloud node works as a central aggregator, synthesizing the knowledge derived from multiple participating devices to construct a global machine learning model.

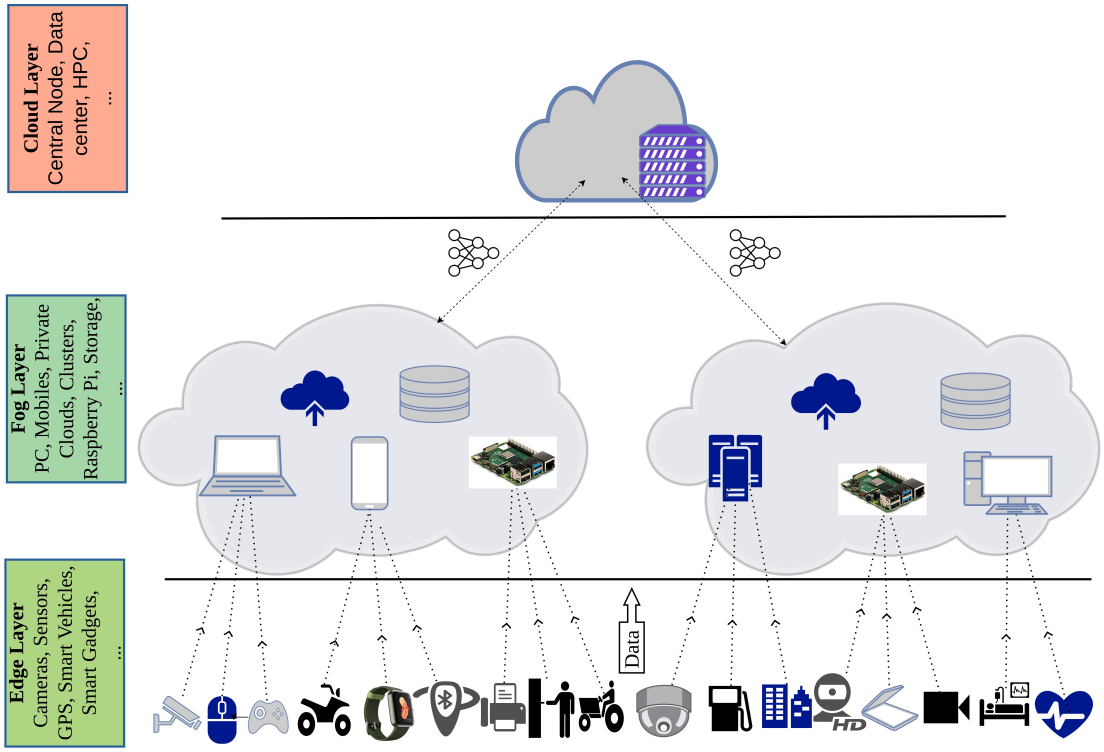


Figure 3.1: Distributed learning architecture

The Edge layer contains a large number of resources constrained IoT devices such as cameras, watches, GPS, bulbs, sensors, radars, etc. These devices continuously generate raw data by sensing the surroundings, but are limited in storage and computation. The edge layer directly connects with the fog layer and shares its data with the associated fog node. With enough computational power, a fog node trains a machine learning model in collaboration with the central server. Although a fog node stores historic data of associated devices, the training is done on recently captured periodic data frames. This simulates online training of continuously changing datasets. The Fog-Cloud layer participates in federated learning for machine learning. Once local models are trained on the fog layer, it is shared with the cloud layer. The cloud layer aggregates the local models and creates a global model. The federated learning with fog-cloud architecture continues till the convergence of the global model. The proposed architecture is used for model training on rapidly growing data in a resource-constrained setup. The fog layer is responsible for data collection and federated learning training with a cloud node. While the fog layer only shares learning parameters with the cloud for aggregations.

The raw data is stored at the fog layer, which is not shared with the central server. The proposed architecture simulates a distributed machine learning using computations of resource-constrained devices.

3.3.2 Decentralized federated learning

In decentralized federated learning, locally trained models from distributed devices are aggregated at a central server. The central node aggregates these independently learned models to construct a global model that encapsulates the collective learning of all participants. A conventional machine learning approaches collect possible data D to the server, then it trains a machine learning model M using a sophisticated algorithm. In contrast, federated learning trains its model without collecting all possible data at the server. It is a collaborative learning paradigm where k number of participating devices train local models on their local data. Each participating device i contains its personal data D_i. The total data set $D = \sum_{i=1}^{k} D_i$. Here, D_i is a collection of input data samples $(x_j, y_j)_{j=1}^{n}$ for supervised learning. Where $x_j \in \mathbb{R}^d$ is a d dimensional input data and $y_j \in \mathbb{R}$ is the associated label for input x_j. The device data D_i is remotely generated through the edge layer.

For every participating device, it is a machine learning task where it has to train the global model using its local data D_i. It takes input data $(x_j, y_j)_{j=1}^{n}$ to compute local parameters i.e. weights and biases. The loss function at every devices i for dataset D_i is $F_i(w) = \frac{1}{|D_i|} \sum_{j=1}^{n} f(h(w, x_j), y_j)$. Where $f(h(w, x_j), y_j)$ is the loss for j^{th} sample from D_i. The participating devices optimize the loss using an optimizer to find optimal parameters. In the subsequent step, the locally trained parameters (weights and biases) are shared with the central server for global model creation. The central curator receives all k locally trained models parameters and performs aggregation operations on them. The weights and biases (W, b) of respective layers of every model are aggregated. Thus, the aggregated model contains representation from all models, which is further fine-tuned in the next iteration. The training is executed in collaboration with the central server as shown in Fig 3.2. The aggregated/updated global parameters (W, b) are pushed back to local devices for the next round of training. It is an iterative algorithm that optimizes global parameters using local model updates. The global loss function for the system is $F(w) = \frac{1}{|D|} \sum_{i=1}^{k} |D_i| \times F_i(w)$. This training cycle continues

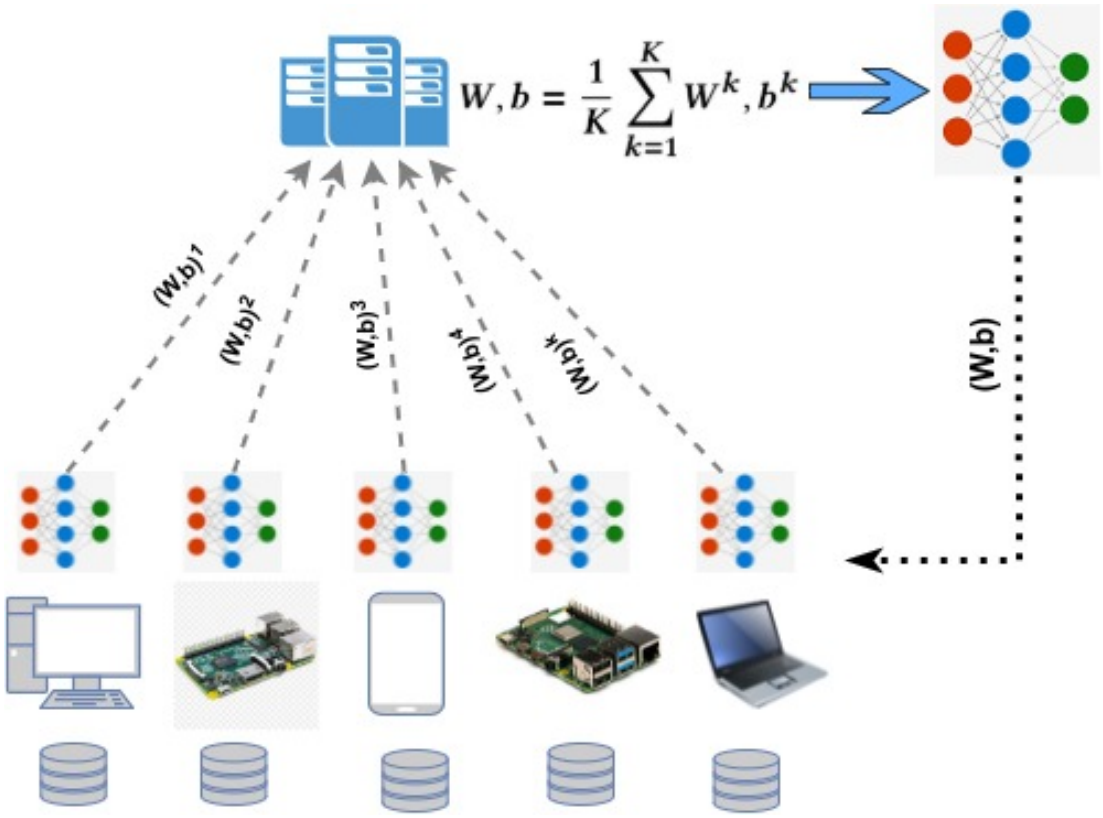


Figure 3.2: Decentralized federated learning training paradigm

until convergence without accessing raw data as shown in Fig 3.2. The goal of federated learning is to learn a global model by combining all local models.

3.3.3 Online training and data privacy

The proliferation of digital devices has resulted in an immense and continuous stream of data, creating significant challenges for machine learning training. As discussed earlier, sending raw data to the cloud would not be an efficient way to process it. Additionally, collecting, storing, and applying machine learning algorithms requires huge computational resources. As the number of devices grows, so does the volume, velocity, and variety of data, leading to the big data problem. To address this to some extent, a continuous learning approach is applied while training the model. So, instead of training the model on the complete dataset, online training is done on a subset of data. The subset can be periodic data generated from a device over a fixed

time interval. Once a device trains the model on current data, it shares the model's parameters with the central curator for global modeling. The curator aggregates all the parameters and creates a central model for all devices. In the subsequent round, each device produces a new set of current datasets. Similarly, in this round, the global model is further fine-tuned with the next set of datasets.

The approach is designed for an IoT network that consists of low-resource devices that are incapable of machine learning training on a very large dataset. However, the IoT devices, such as sensors, CCTV cameras, and radars, do not have enough computation resources to run on-device machine learning. The architecture addresses this issue by deploying a fog node near IoT devices. All data generating devices are connected to the associated fog node to share raw data. With sufficient computational and storage capabilities, a fog node stores periodic data and performs online machine learning training. The rapid growth of data accumulates a large amount on a fog node. Due to the computational limitations of a fog node, online training is done on recently captured data, leaving historic data aside. At the same time, the entire data is kept at the fog layer on backup storage, which can be reproduced in future if required. Then fog node participates in the federated learning process in collaboration with the central cloud server.

IoT devices may have highly sensitive information, including personal identifiers, location data, financial records, private communications, chats, and other forms of confidential content. Ensuring data privacy and security remains a critical challenge in cloud–IoT computing environments. With increasing concern over data sharing, users express concerns over sharing personal data, such as images, chats and messages, to the cloud due to potential misuse or breaches. However, raw data contains crucial information that can be useful for various applications such as recommender systems, security monitoring, smart home automation, safety predictions, predictive analytics, etc. Additionally, machine learning task has to follow strict data protection rules such as General Data Protection Regulation (GDPR). To address the privacy concerns, the proposed architecture used a decentralized machine learning approach for model training. In the proposed work, the data is stored at the fog layer and not shared with the cloud. The architecture ensures that raw data is never transmitted externally, thereby enhancing user privacy while enabling collaborative learning across distributed nodes.

3.3.4 Evaluation and results

This section describes evaluations and experimental results of the proposed architecture for a machine learning task. The model training is done for radar data for an IIoT setup. We simulate the proposed architecture using Docker containers. Then, training of a global model for safe distance detection is done for human position in HR workspace. To show the efficiency of the proposed work, we have trained a ANN model in distributed environments and achieved expected results.

3.3.4.1 Docker based fog enabled federated learning

We have used Docker containers to simulate resource constrained distributed machine learning. The Docker engine facilitates multiple containers to run various programs independently. A container provides the runtime environment for program execution. So, a set of containers can be created to execute tasks in parallel. With sufficient computational resources, we employed multiple Docker containers as a fog node. Every container runs a machine learning model independently with its local data. We have used the gRPC library for requests and service calls between the fog and the central node. The federated learning is done between cloud and fog nodes using Docker containers with gRPC calls. The IoT devices generate continuous data and share it with the fog node. Fog node stores historical data at backup devices and trains the model on recent data. To simulate the continuous data generation and online training, we used a fixed set of data samples for local training. Every container has its personal data, and model training is executed on fixed periodic sequential training samples.

We have simulated the proposed architecture to train an ANN-based machine learning model for human operator position detection in a human-robot workspace. The dataset is recorded from multiple Frequency-Modulated Continuous Wave (FMCW) radars. A fog node compile one minute of data from every device and complete one round of federated learning. We trained the model on 60 frames, assuming every radar is generating 1 frame/sec. The next round of training is done on the next sequence of datasets. For this experiment, we have taken 5 fog nodes for decentralized machine learning training. The fog node trains a fully connected ANN using the Tensorflow framework. The ANN model contains a single hidden layer with 64 neurons. The input

and output layers have 512 and 8 neurons, respectively, based on data dimension and output labels.

We have trained a shallow neural network with one hidden layer. The model is a fully connected dense network with 64 units in the hidden layer. Input layer has 512 input neurons which is fully connected to the only hidden layer (dimension $512 \times 64 + 64$ bias) followed by a ReLu layer. The output layer has 8 neurons, which are densely connected to the hidden layer (dimension $64 \times 8 + 8$ bias). The final output label is predicted based on the softmax activation function at the output layer. The local training on every device is executed for 5 epochs to simulate low computational resources. The network is trained by backpropogation algorithm using categorical crossentropy loss function. Further, the 'Adam' optimizer is used as an optimizer to optimize the training error. The loss value of the global model is calculated on test data. Also, we have traced the accuracy performance of the global model on both personal and unknown test datasets. The simulation is done to show the computational intelligence of the proposed architecture on continuously changing data.

3.3.4.2 FMCW radar dataset for federated learning

The proposed architecture is validated for a real world IIoT use case. The dataset is recorded from multiple Frequency-Modulated Continuous Wave (FMCW) radars in a human-robot workspace. FMCW radars are effective IIoT devices in industrial setups for environment sensing, distance measurement, etc. These radars are placed in a shared workspace of human robot to capture human position in the environment. Detection of human position in an industrial setup is crucial for worker's safety. These radars contain data to measure the distance and position of the human operator near them. Every device has its locally generated dataset. However, each participating devices have all classes samples. We used the mentioned dataset to train a machine learning model to classify human safe distance. The dataset is published by Stefano Savazzi, which can be downloaded from IEEE Dataport [97]. The radars output signals are preprocessed and converted into 512-point. The detailed methodology and collection of data are given in this paper [98].

The input sample contains 512 points with 8 labels. The labels are characterized by different distances between human operator and radar. The dataset contains a total of 32,000 samples of FFT range measurements 512 points. The dataset is already divided

Table 3.1: FMCW radars dataset

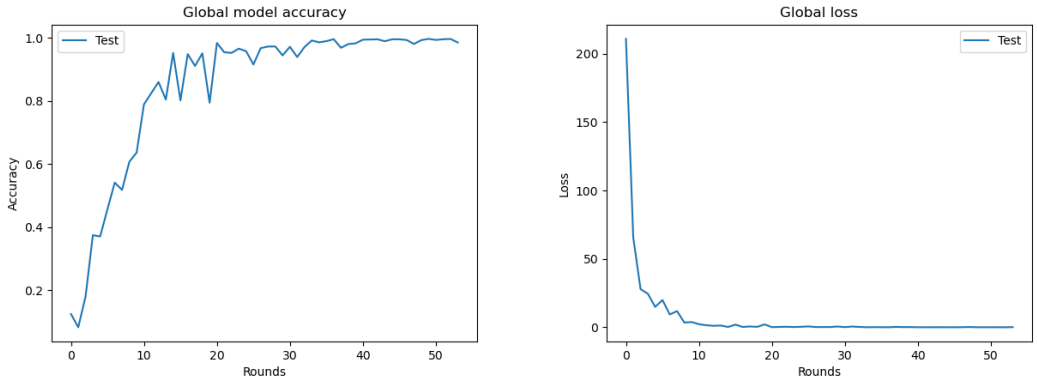
Distance(m)	Class	Critical/Safe
<0.5	1	Critical
0.5 <= d <1.0	2	Critical
1.0 <= d <1.5	3	Critical
1.5 <= d <2.0	4	Safe
2.0 <= d <2.5	5	Safe
2.5 <= d <3.0	6	Safe
3.0 <= d <3.5	7	Safe
>=3.5	0	Safe

into training and testing samples of 16,000 × 512 shape. Additionally, the data sample is also randomly distributed over various devices for federated learning simulation in the database. We have implemented the given data distribution over 5 devices for federated learning to learn ANN model for safe/unsafe position detection. The training is done for C=8 classification of the potential situation in human robot workspace. The label is an integer from 0 to 7, where Class 0 represents human distance $\geq$ 3.5m, which is also marked as safe. Class 1 is represented as critical since the distance is $\leq$ 0.5m. Other labels are marked based on different distance measures between humans and radars. Table 3.1 details about labels for various classes.

3.3.4.3 Results and Analysis

We trained a ANN model for human position classification on the given dataset. The online training is done with 60 frames at a time on 5 fog nodes. We divided training sample equally among all devices. From 16,000 training samples, every fog node receives 3200 independent samples. The local model training is done with the current 60 samples for 5 epochs only. Then central server performs aggregation of all learnt model parameters. This completes one round of federated learning. In the next round, we use next 60 samples for training by skipping previous data points. We executed 53 rounds that exhausted entire local dataset training. At every round, we assess the model's performance in terms of loss and accuracy. The global model is evaluated on the test dataset residing on the server. The training loss and accuracy of the model on test data are shown in Fig 3.3.

As training increases, the model improves its accuracy significantly. The model



(a) (b)

Figure 3.3: Global model performance on test data: (a) Loss value and (b) Accuracy over training round

optimizes loss and stabilizes the training after 30 rounds. The proposed online training performed exceptionally well as the test accuracy reached 99%. The local model is trained on multiple fog nodes in parallel. The global model is a combined learning of all local models. Fig 3.4 shows the accuracy of the global model on various local data residing in fog devices. The accuracy is the averaged learning that cancels drift on various local training data. So, even though a particular device overfit the model, the aggregation ensures it to be generalized. It clearly shows that the global model is an aggregated representation of locally trained models.

Training a machine learning model with a neural network is prone to overfitting, specifically in online training mode. Comparatively, a large model with huge parameters can memorize the data labels, which tend to perform poorly on test data. In our experiment, the amount of data passed at a particular instance is relatively small (60 frames/iteration). The ANN model quickly learnt the sample with very high accuracy (>90%) in fewer epochs. However, it failed to produce similar results on global test data. This is because of possible model overfitting on relatively small data. The federated learning consolidates these models to combine learnt information. The combined output model is better generalized and reduces possible overfitting. With the varying number of participating devices, federated learning prevents overfitting intrinsically. However, other generalization techniques such as dropout, normalization, regularization, etc, can be applied at the architectural level for better generalization.

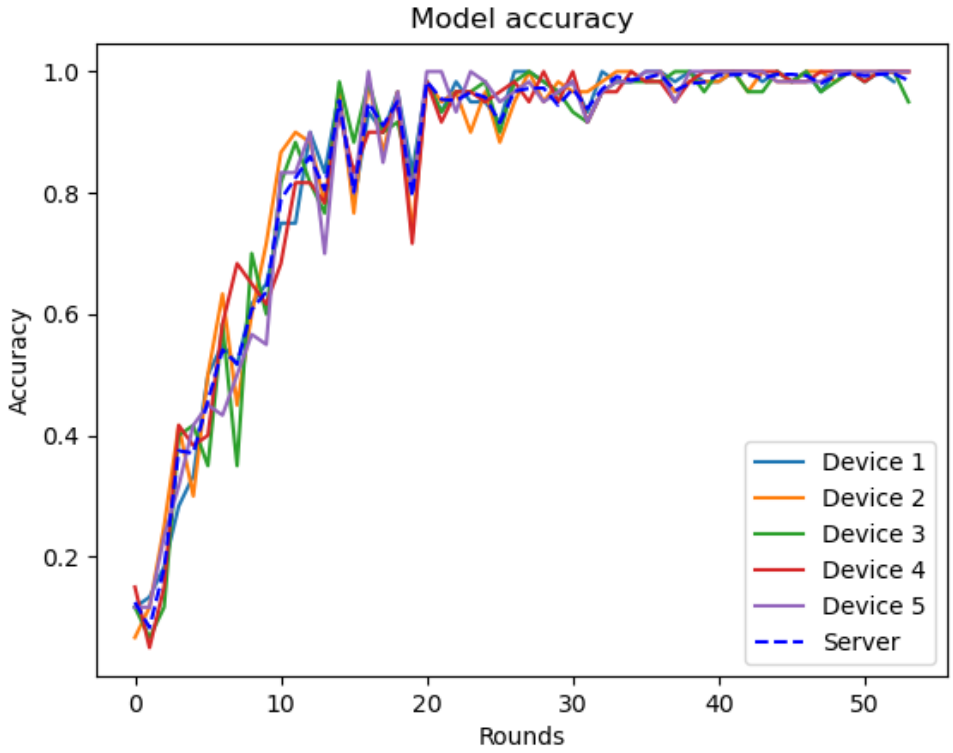


Figure 3.4: Model performance on all devices

3.3.5 Conclusion

This work focuses on machine learning model training on decentralized data. We proposed fog enabled distributed training architecture to train a neural network on rapidly changing data. The architecture suitably uses decentralized algorithms for model creation. The edge layer is responsible for data generation. The cloud layer coordinates with computational nodes on the fog layer for machine learning. Whereas, the fog layer participates in distributed machine learning training with the central server. We tested the proposed architecture for a real world IIoT use case. The simulation results show that it is capable of learning a model from a changing dataset. Hence, we find a feasible solution to train a neural network model on streaming data using resource-constrained devices. We further extended the architecture to build a distributed framework, FIDEL, that enables the edge-to-cloud continuum for model training. We will discuss this in the next section.

3.4 FIDEL: Fog integrated federated learning framework to train neural networks

In the previous section, we discussed an architecture for federated learning on resource-constrained devices. It addresses research question 1 by demonstrating the feasibility of training models at the fog/edge layer. This section provides a comprehensive discussion of the FIDEL framework, outlining its core components in detail. It further elaborates on the implementation strategies and methodological approaches in the proposed system.

3.4.1 The FIDEL Framework

Rather than centralized training, we consider a decentralized data processing framework for predictive/inferential machine learning tasks. As discussed earlier, sending raw data to the server has various concerns, such as large bandwidth consumption, network congestion, high latency, and data privacy. However, training a machine learning task with low computational devices has multiple challenges. To address this, we propose a fog-integrated cloud-fog-edge training framework named FIDEL for machine learning tasks from continuously generated data by various IoT devices. The framework can train a shared machine learning model from rapidly growing distributed datasets from multiple sources. Fig. 3.1 shows the training architecture of the FIDEL framework. It is a three-layered architecture where each layer contains various IoT devices responsible for performing individual tasks. Devices in the horizontal layer perform their tasks in parallel and then communicate with the vertical layer to learn the model collaboratively. Section 3.3.1 had some discussion about architecture and its components. The subsequent section will elaborate more on each layer of the framework, along with its components.

3.4.1.1 Edge layer

The main task of the edge layer is to generate data. This layer contains very low-resourced devices such as sensors, GPS, smart gadgets, cameras, health-related devices, etc., as shown in Fig. 3.1. Devices available in the layer can continuously generate raw data, but they do not have enough resources to store and process it at the edge layer. Hence edge layer is used as a data generating layer. Once the data is generated, it is

shared with the fog layer to the specified fog node within the vicinity of a user. The devices in this layer are directly connected to one of the processing nodes at the fog layer within the user's premises. For example, in a smart home scenario, the temperature and humidity sensors will continuously sense the environment and push data to the nearest PC tools like Kafka or MiNiFi. Hence, data acquisition is performed at the edge layer, and it is being transferred constantly to the fog node for training and persistent storage if needed.

3.4.1.2 Fog layer

The fog layer is capable of small-scale analysis and processing. The job of the fog layer is to provide computational training and persistent storage of the raw data generated at the edge layer. The storage of data at fog nodes serves two purposes (i) it ensures the privacy of the user's data (ii) the data will be available for analysis/reporting in the future. The fog layer contains relatively higher computational IoT devices such as Raspberry Pis, personal computers, private clouds, clusters, etc. This computational capability of the fog nodes can be used to process raw data. The nodes at the fog layer reside close to data-generating devices, possibly inside the user's premises, which minimizes latency and network congestion.

In the FIDEL framework, fog nodes are responsible for two tasks: (i) they provide computational resources for parallel training, and (ii) they actively store raw data for future requirements in persistent storage if needed. However, the main job of the fog node is to fine-tune the global model based on its computational capability. Every fog node selects recent periodic unseen data from the edge layer and trains a shared model for global consumption. Although a fog node stores historical data of associated devices, the training is done on the last captured periodic data frames. Hence, the fog nodes select the next set of periodic datasets in the next successive iteration to fine-tune the model, called online training. Online training is needed because we are training networks on continuous data using resource-constrained devices. Then, fog nodes collaborate with the central cloud node to create a global model. Note that the fog nodes have limited compute resources that can not process a large volume of data.

3.4.1.3 Cloud layer

The cloud node contains high-performance computing nodes such as servers, GPUs, Data centers, etc. It works as an actuator to fog nodes during global model production. The job of the cloud node is to provide the necessary support to fog nodes so that global knowledge can be created. It controls distributed training from the top of the hierarchy. It is used as a global model aggregator that accepts locally trained models from multiple fog nodes. The global model creation is done at the cloud node by inclusively aggregating all locally trained model parameters. This process combines and merges knowledge of the data patterns learned from different fog nodes on multiple datasets.

The cloud node can also be used for final report presentation, analysis, and other resource-intensive work. Since the user's data is not shared with the cloud node, a cloud node also performs various operations to authenticate the veracity of data and participating nodes. Additionally, the raw data stored at the fog nodes can be shared with the cloud node if requested for validation and complex processing. However, in the current scenario, a cloud node works as a master node to train a federated learning model.

3.4.2 Framework implementation and communication

The FIDEL framework is designed to train neural networks on continuously generated data over resource-constrained devices. The fog layer can have hybrid devices such as Desktops, Raspberry Pis, etc. So, device heterogeneity is a major concern in implementation. Hence, for seamless training/deployment of the services, we have containerized the application with Docker images for a variety of hardware specifications. We have created Docker images that are pushed to Docker Hub. The images are based on underlying hardware and os support. During training, the appropriate image is pulled, and execution starts. On the other hand, the Docker compose file in the project can create a respective image according to the underlying hardware.

Federated learning implementation has to deal with multiple communication rounds. The server supervises the entire training for the participating clients. Therefore, network connectivity and device performance play a pivotal role in training. Hence, we have implemented FIDEL using the following two communication strategies.

3.4.2.1 Synchronous update

Devices in the edge layer communicate with the fog node as continuous data streams. As soon as the edge node produces a data point, it is pushed to the associated fog node. However, communication between cloud and fog nodes is event-driven. Communication in a stable network benefits the training with low latency and high availability. For such a scenario, synchronous federated learning (SFL) is implemented using gRPC framework. gRPC is an open-source remote procedural call framework for distributed computing. The services and communication are defined in the protocol buffer. The gRPC stub can call gRPC server's methods remotely. The server initiates the training by pushing the initial model to clients. Then it waits for all clients to complete their execution. Thereafter, it performs an aggregation operation. Fig. 3.5 shows the SFL model design.

The server starts the execution by creating an initial model, and then it invokes the clients' initialization method to initialize indices and other metrics on participating clients. This is executed once to create/reset relevant files on the server and clients. Thereafter, it invokes the train method remotely, which executes training modules on the clients. Every client fetches recent data for online training and executes neural network training on it. Once training is completed, it returns model parameters to the server. At the same time, the server waits for model collection. Once every client returns their model, the server executes aggregation method to consolidate all models. The final aggregated model is updated to all clients, and then the train method is called for the next iteration. The communication is done between gRPC stubs and servers using dedicated channels. Fig 3.6 shows execution flow for the FL paradigm. Firstly, the server calls Initialize() method on the server side that creates the initial model, then it invokes clients' Initalize() method through an RPC call that initializes indices and metrics files on clients. The local method are shown in highlighted text, and the RPC remote methods are shown in non-highlighted text. Thereafter, it invokes Train() method for local model training. Once all clients return their models, the server executes Aggregate() method to consolidate all models. The final aggregated model is sent again to all clients by calling Train() method for the next round of training.

Since FIDEL considers resource-constrained devices for training that could go unresponsive due to battery/electricity shortages, device issues, internet dropout, etc.

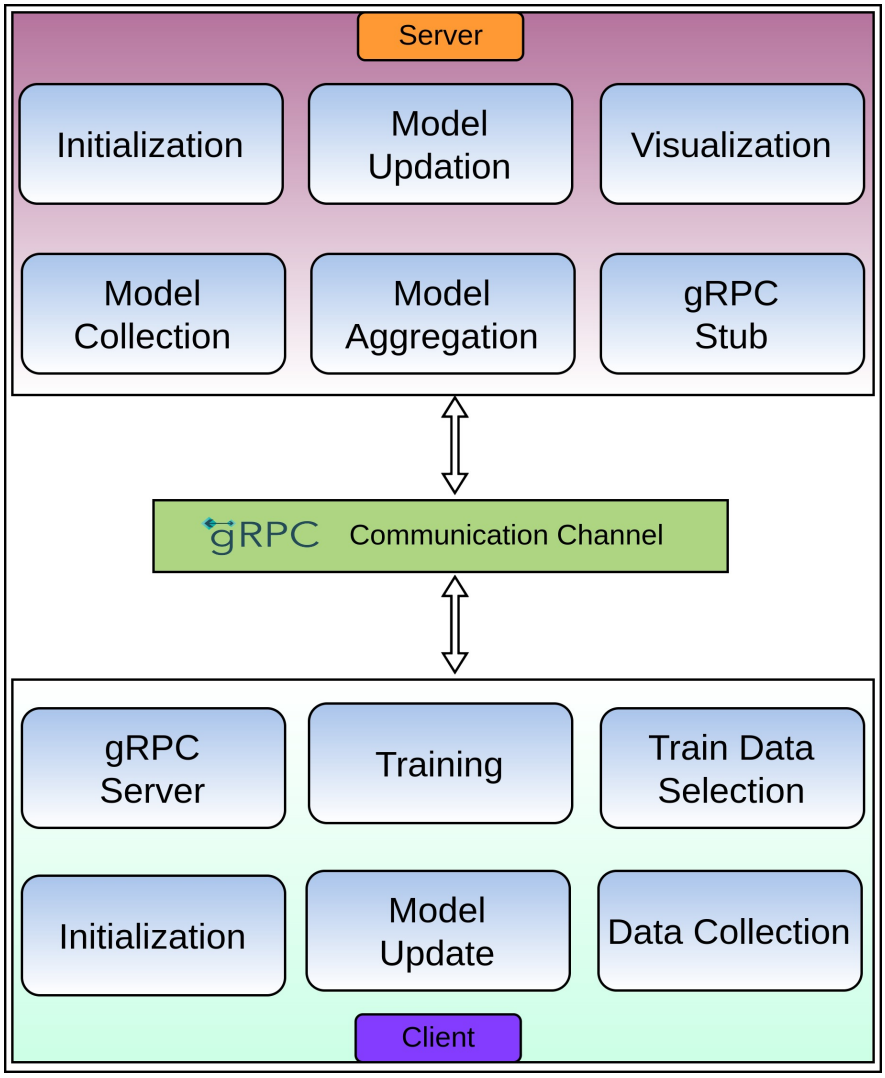


Figure 3.5: Synchronous federated learning model

Similarly, a straggler or unreliable network equally hampers the training. Statistical methods like sampling can deal with a few stragglers, but this is inefficient for the smaller network. Hence, the synchronous update is efficient only for a reliable and larger network.

3.4.2.2 Asynchronous update

In asynchronous communication, the server does not wait for every update, rather it continues to aggregate as soon as it receives a local model. Hence, the communica-

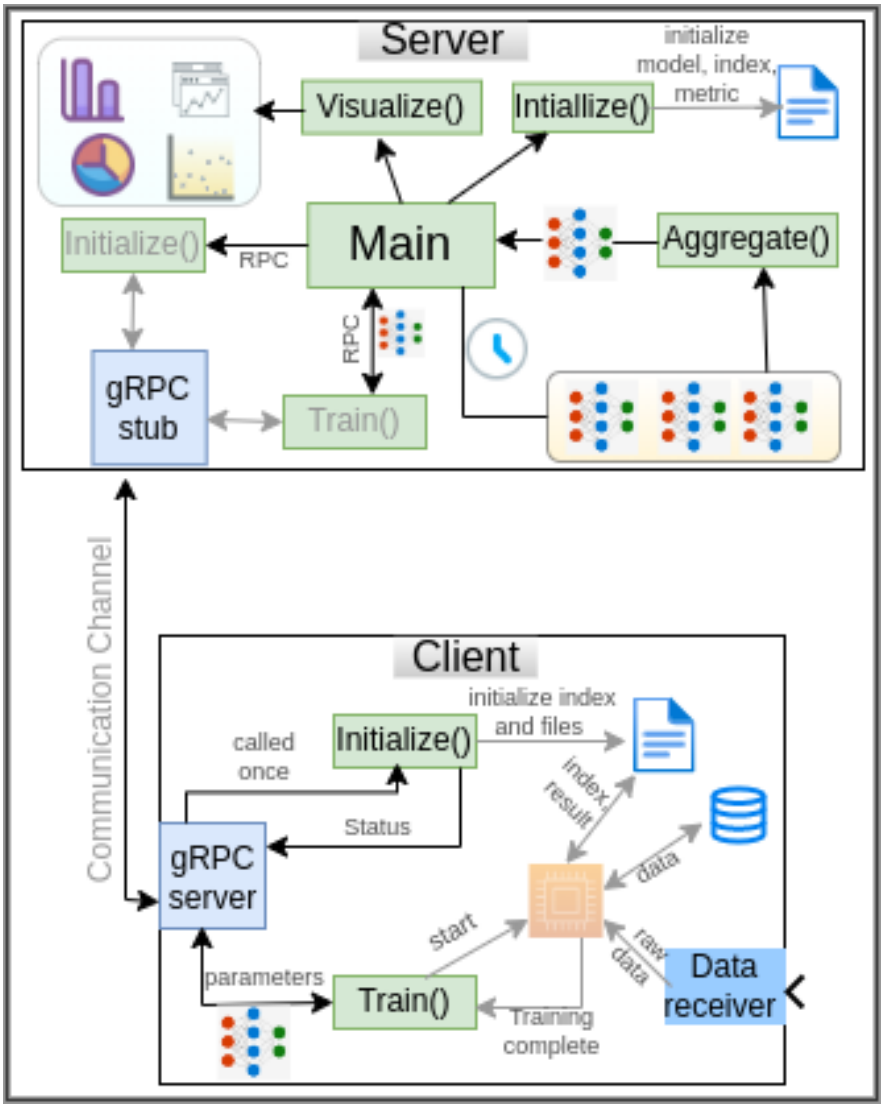


Figure 3.6: Execution of synchronous federated learning model

tion bottleneck can be eliminated with asynchronous updates between two layers. We have extended the federated learning implementation to time-bounded asynchronous updates. The server continues the execution of the framework with a fixed wait time. Time is a variable unit that can be set externally based on underlying hardware constraints. Thereafter, it continues with aggregation tasks with available updates. The implementation of Asynchronous Federated Learning (AFL) is done using Message Queuing Telemetry Transport (MQTT). MQTT is a lightweight publish-subscriber

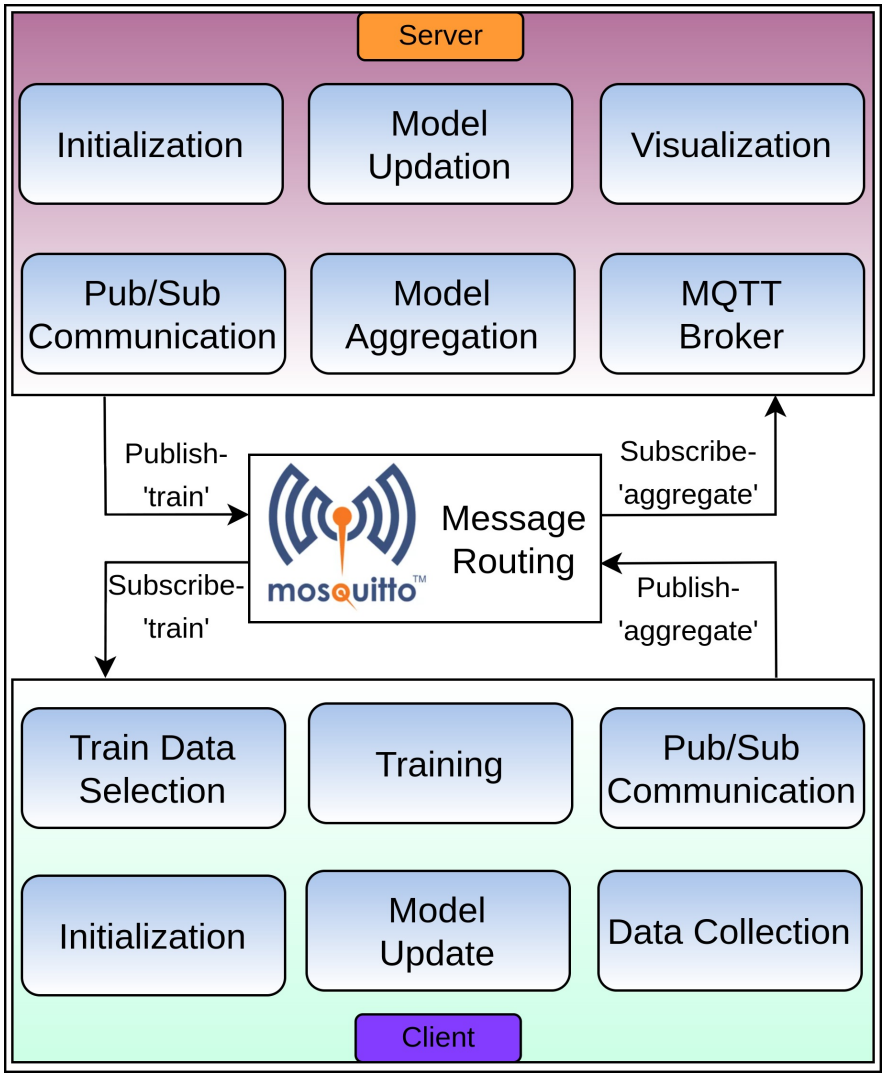


Figure 3.7: Asynchronous federated learning model

based message queuing service protocol. The communication is handled by a broker that maintains topics and connected clients. All machines need to subscribe to the relevant topic. Then the broker pushes the message to all subscribers if someone publishes on that topic. The AFL implementation design is shown in Fig. 3.7. In this implementation, we have used mosquitto broker and advertised two topics, 'train' and 'aggregate'. Here, fog nodes subscribe to the 'train' topic, and cloud nodes subscribe to the 'aggregate' topic. So whenever the server publishes on 'train', all clients get notified and start training. Once an individual client finishes its training, it publishes

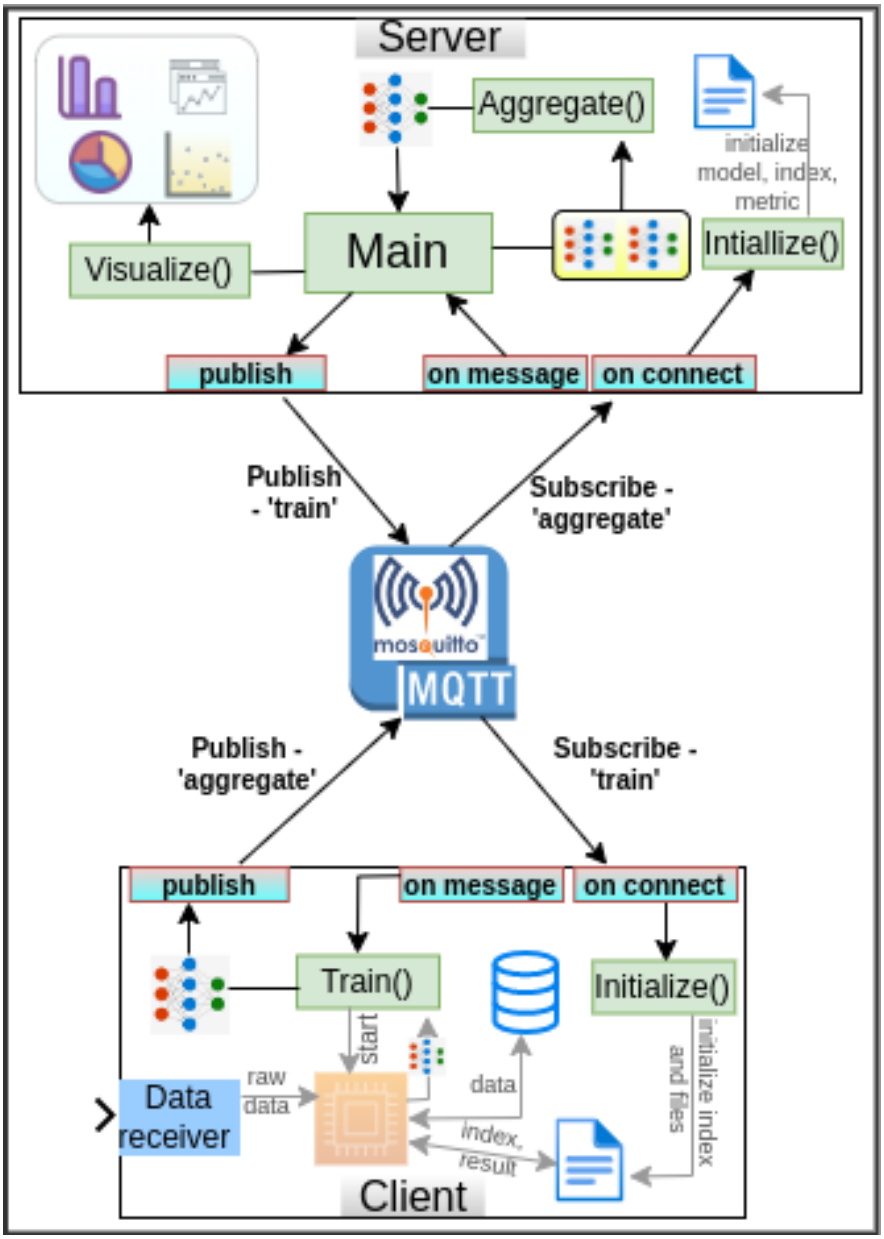


Figure 3.8: Execution of asynchronous federated learning model

the result on 'aggregate' topic, which is subscribed by the server. The server waits for the specified time and then proceeds with the currently available updates.

Both server and clients subscribe to relevant topics to the broker. Upon the successful connection, the server creates the initial model, and all clients initialize their indices. Training is initiated by the server by publishing on the 'train' topic that invokes

training on fog nodes. Afterward, the locally trained model is published to 'aggregate' topic for aggregation. The server incorporates the local models into the global model and pushes back the global model for further training. If a node stops working or goes unresponsive at any iteration, the server continues the process with available nodes. At the same time, a new authenticated device can subscribe at any time and contribute to the training. The asynchronous strategy resolves bottlenecks caused by device/network failure. It also deals with delays posed by stragglers nodes. So even if there are delays in the network due to stragglers or slow connection, the aggregation process continues to progress with other models. The execution flow of the asynchronous updates is shown in Fig 3.8.

3.4.3 System model and distributed training

In this section, we describe the system architecture and the distributed learning techniques used to train neural networks. We also explain how online training is formulated and integrated into the overall framework.

3.4.3.1 System model

We consider an IoT network of $\mathcal{K}$ edge nodes, $\mathcal{F}$ fog nodes and a central server in the distributed training paradigm. Every $\mathcal{K}$ edge node is connected to one of the associated $\mathcal{F}$ fog node ($\mathcal{K} > \mathcal{F}$). A node $\mathcal{K}$ in the edge layer communicates with the respective fog node $\mathcal{F}$ to transfer raw data. Table 4.1 outlines the formal symbols and their definitions as used in the proposed methodology. An edge node continuously generates the data by sensing the environment and shares it with the designated fog node. All the raw data is stored at the only fog node to ensure privacy and future analysis. However, the online training is done with the latest datasets; hence the raw data can be discarded if it is of no use. Federated learning is done between the cloud node and fog nodes, as shown in Fig. 3.9. The framework executes model aggregation at the server and local training at every fog node.

With limited computational resources, the fog node trains a neural network on its local dataset. It contains a historical dataset in its storage, but in view of limited resources, the training is done with the recently generated periodic dataset. Here, every participating fog node contains a total of D_h datasets, whereas, at a particular iteration, the selected data for training is based on fog node computability. This can

Table 3.2: Symbol table

Symbol	Meaning
$\mathcal{K}$	Number of clients in the edge layer
$\mathcal{F}$	Number of clients in the fog layer
D_h	Historically stored data at a fog node
D_i	Current training data at i^{th} fog node
τ	Training dataset threshold value
d_i	i^{th} data frame consisting of (X_i, y_i)
t_p	Previous frame index for data selection
X_i	i^{th} input data frame
y_i	i^{th} output label
θ_i	Model parameters at i^{th} node consist of (W_i, b_i)
θ_g	Global model parameters at server
r	Number of federated learning round
E	Number of epochs for training at a fog node

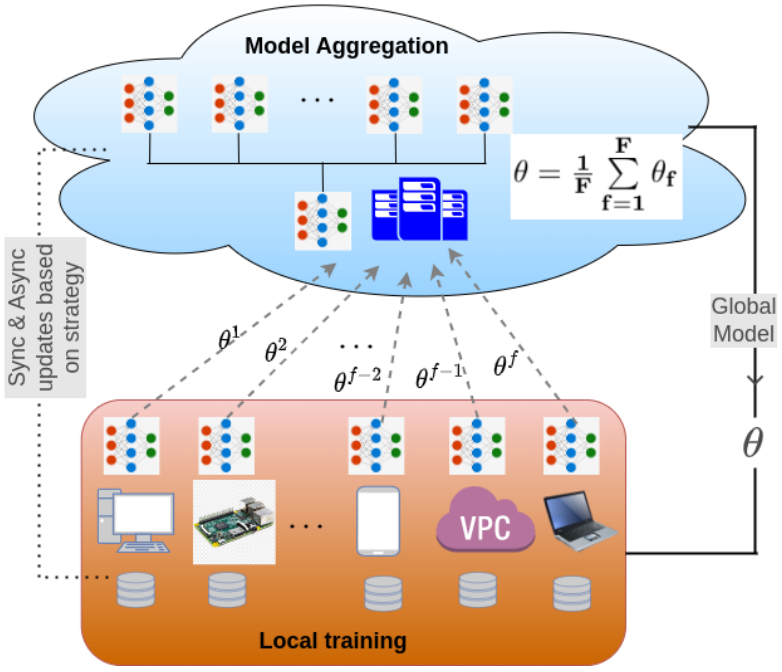


Figure 3.9: Decentralized federated learning training paradigm

be done according to time constrained or the number of samples a fog node can easily process based on its computation resources. So a fog node i shortlists its training

samples D_i from historic dataset D_h based on the following strategy.

$$D_i = \bigcup_{j=t_p}^{t_p+\tau} (X_j, y_j) \tag{3.1}$$

Here, t_p is the previous index of sequential data initially set to 0 at round 1. In the subsequent rounds, D_i is populated with new set of data and value of $t_p \in [0, min\{|D_h|, t_p + \tau\}]$ is incremented by $t_p + \tau$. Whereas (X_j, y_j) is i^{th} data frame.

3.4.3.2 Local execution

Each participating fog node i contains its local data D_i for machine learning task at a specific round. Here, D_i is a collection of input data samples $(X_j, y_j)_{j=1}^{n}$ for supervised learning. Where $X_j \in \mathbb{R}^d$ is a d dimensional input data and $y_j \in \mathbb{R}$ is the corresponding label for input X_j. A fog node chooses a subset of data frames from the historical dataset using strategy shown in Equation. 3.1. So every fog node has its personal data set D_i. Hence, the current global data set for training is the collection of all training data at round r, as follows.

$$D_{global} = \sum_{i=1}^{\mathcal{F}} D_i \tag{3.2}$$

Whereas the total generated dataset for machine learning training is:

$$D_{total} = \sum_{i=1}^{\mathcal{F}} D_h^i, \ where \ D_h^i = \sum_{t=0}^{T} (X, y) \tag{3.3}$$

Here, D_h^i is the historical data frames collected on a fog node i since time $T \in [0, Now]$. For every fog node, it is a machine-learning task to fine-tune the global model with its local data D_i. The goal of the machine learning task is to train a model on local data that should fit the current global data set D_{global}. At the same time, it is expected that the model will equally fit and generalize on the cumulatively generated dataset D_{total}. And eventually, it should produce similar results on unseen datasets. The training is done on locally selected periodic data set $D_i = (X_j, y_j)_{j=1}^{n}$. The loss function of fog device i for dataset D_i is

$$L_i(\theta) = \frac{1}{|D_i|} \sum_{j=1}^{n} g(h(X_j, y_j; \theta)) \tag{3.4}$$

Where $g(h(X_j, y_j; \theta))$ is the loss for j^{th} sample from data D_i. The ML model takes input data $(X_j, y_j)_{j=1}^{n}$ to compute local parameters $\theta = [W, b]$ i.e. weights and biases. Each fog node $\mathcal{F}$ updates its local parameters as follows:

$$\theta_f = \theta_f + \eta \nabla L_i(\theta) \tag{3.5}$$

Once the local parameters θs are calculated, it is shared with the cloud for global execution. This process continues with the next set of data till we achieve a model with the expected accuracy. The execution flow of the fog node is shown in Algorithm 1, which describes local training process.

Algorithm 1 Local Execution at i^{th} fog node

Input: Current global parameters θ_g, E, t_p, τ
Output: Updated parameters θ_i
  Select current local data D_i from historical dataset D_h using Equation 3.1
  for `e = 1,2,3,...,E` **do** ▷ ML training
    Calculate local update θ_i on data D_i using equation 3.5
  end for
  $t_p \leftarrow t_p + \tau$
  return Final parameter θ_i to the server

At every round, a fog node selects the dataset and processes it to fine-tune model parameter θ shown in Algorithm. 1. Thereafter final parameters θ are sent to the global model aggregator. The cloud node performs model aggregation based on Algorithm. 2.

3.4.3.3 Global execution

Every fog node optimizes the loss using an optimizer to find optimal parameters. In the subsequent step, the locally trained parameters are shared with the central server for global model creation. The central curator receives all $\mathcal{F}$ locally trained models' parameters and performs aggregation operations on them. The value of $\mathcal{F}$ may change based on communication strategy. All the parameters θ of respective layers of every model are aggregated. Thus, the aggregated model contains representation from all

models, which is further fine-tuned in the next round. Fig. 3.9 shows the collaborative learning paradigm between fog and cloud layer. The aggregated/updated global parameters θ are pushed back to local devices for the next round of training. It is an iterative process that optimizes global parameters using local model updates. The idea is to train multiple models on a smaller dataset D_i and then accumulate the knowledge which should work on the larger dataset combined, D_{global}. Hence, we try to optimize the global loss value of the current global dataset. The global loss function for the system is :

$$L(\theta) = \frac{1}{|D|} \sum_{i=1}^{\mathcal{F}} |D_i| \times L_i(\theta) \tag{3.6}$$

The goal of federated learning is to learn a global model by combining all local models that produce similar results on total dataset D_{total}. This training cycle continues till convergence without accessing raw data, as shown in Fig 3.9. The server execution for global model aggregation is shown in Algorithm. 2.

Algorithm 2 Global Execution at server node

Input: Locally trained parameters $\theta_1, \theta_2, ..., \theta_{\mathcal{F}}$
Output: New global parameter θ_g
  Initialize empty parameter space for θ_g
  for `f = 1,2,3,...,`$\mathcal{F}$ **do** ▷ Model Aggregation
    for Each layer l of the parameter space **do**
      $\theta_g^l = \theta_g^l + \theta_f^l$ ▷ Summation of associated neurons
    end for
  end for
  $\theta_g = \frac{\theta_g}{\mathcal{F}}$
  return Send back global parameters(θ_g) to fog layer for further training

The cloud node acts as a central curator that aggregates existing knowledge extracted from multiple fog nodes. It performs aggregation operation on every neural network unit as discussed in Algorithm. 2. For aggregation, it performs the FedAvg method. The new model is created using the aggregated parameter, which is tested for unseen datasets. Local and global execution combined complete one round of federated learning. The training continues in online mode with new datasets till the model converges or the desired accuracy is achieved on unseen test data. The prototype model is implemented with both synchronous and asynchronous updates.

3.4.4 Evaluation and results

This section describes experimental setup, evaluations, and results of the FIDEL. The model training is done on radar data for IIoT setup. We have evaluated the proposed architecture by training multiple neural network models using lightweight Docker containers. To show the learning capabilities of the FIDEL, three models (shallow network, deep network, CNN) are trained using both synchronous and asynchronous updates in distributed environments with volatile datasets. Then we compared FIDEL training with other synchronous and asynchronous frameworks. The code is available at Cloud and Smart lab's GitHub[1] [2] page.

3.4.4.1 Prototype and execution setup

In the prototype setup, we deployed five Raspberry Pi devices, each equipped with a quad-core Cortex-A72 (ARM) 64-bit SoC running at 1.5 GHz and 4GB of RAM, to serve as resource-constrained fog nodes. A high-performance desktop machine with a 16-core Intel Xeon CPU operating at 3.7GHz and 32GB of RAM was used as the central cloud node. The experimental setup is shown in Fig 3.10. All the devices are connected to the same WiFi network, but they can be on different networks. The network is trained with both synchronous and asynchronous strategies with rapidly changing datasets. Synchronous updates are done using gRPC framework. However, communication between the fog node and the cloud node is established using MQTT for asynchronous updates.

We have used lightweight Docker containers for deployment over Raspberry Pi in a distributed setting. Docker engine facilitates containers to run a complete package independently. A container provides a run-time environment for program execution that enables the program to overcome hardware heterogeneity. It enables seamless execution on heterogeneous devices. On every device, a container runs machine learning tasks independently with its local data. The Docker image can be deployed on any machine, which is an additional advantage from a software point of view. The IoT devices generate continuous data and share it with the fog node. Fog nodes participate in federated learning with local datasets synchronously/asynchronously.

[1] https://github.com/cloud-and-smart-labs/FIDEL-Async
[2] https://github.com/cloud-and-smart-labs/FIDEL-Sync

Figure 3.10: Distributed training execution prototype

3.4.4.2 Machine learning model

In this experiment, we have trained three models i) Shallow network ii) Deep Network iii) Convolutional neural network models for human operator position detection. For this experiment, we have taken 5 Raspberry Pi as fog nodes for decentralized machine learning training using TensorFlow. However, any other framework can be used for model training. TensorFlow is a general-purpose, open-source framework designed for building and training neural networks. The input and output layers have 512 and 8 neurons, respectively, based on data dimension and output labels.

	Shallow Network	Deep Network	CNN
Model	Input ↓ FC(64) ↓ Relu ↓ FC(8) ↓ Softmax	Input ↓ FC(32) ↓ Relu ↓ FC(16) ↓ Relu ↓ FC(8) ↓ Softmax	Input ↓ 1D Conv(8) ↓ Relu ↓ MaxPolling ↓ 1D Conv(4) ↓ Relu ↓ Flatten ↓ FC(8) ↓ softmax
Trainable Prameters	33,352	17,080	8,332

Table 3.3: Model details with trainable parameters

The shallow network is a fully connected neural network with one hidden layer of 64 units. Similarly, the dense network contains 2 hidden layers with 32 and 16 units, respectively. Both networks are densely connected, followed by relu activation function. However, the CNN model has two 1D convolutional layers with 8 and 4 units, followed by maxpooling and Relu activation, respectively. Table. 3.3 shows model details with trainable parameters. The final output label is predicted based on the softmax activation function at the output layer. The network is trained by backpropagation algorithm using categorical crossentropy loss function. Further, the 'Adam' optimizer is used as an optimizer to optimize the training error. The loss value of the global model is calculated on test data. Also, we have traced the accuracy performance of the global model on both personal and unknown test datasets. The experiment shows computational intelligence of the proposed FIDEL framework on continuously changing data.

3.4.4.3 FMCW radar dataset for federated learning

For the experiment, we used the same radar dataset for human position detection. The details of this dataset have already been discussed earlier in Section 3.3.4.2, as summarized in Table 3.1.

3.4.4.4 Results and analysis

We trained all models for human position classification in a shared HR workspace use case. The online training is done with 60 frames simultaneously with 5 fog nodes using both strategies. Every device receives 3200 independent samples that execute local training. The central server performs synchronous and asynchronous aggregation of all learned model parameters. The fog nodes compile one minute of data and perform training that completes one round of federating learning with the cloud node. We trained the model on 60 frames, assuming every radar is generating 1 frame/sec. The next round of training is done on the next sequence of datasets. This completes one round of federated learning. In further rounds, we use next 60 samples for training by skipping previous data points. We executed such 54 rounds that exhausted the entire local dataset. At every round, we assess the model's performance in terms of loss and accuracy. The global model is evaluated on the test dataset. To compare the FIDEL's efficiency, we have also implemented recently published SFL and AFL frameworks for

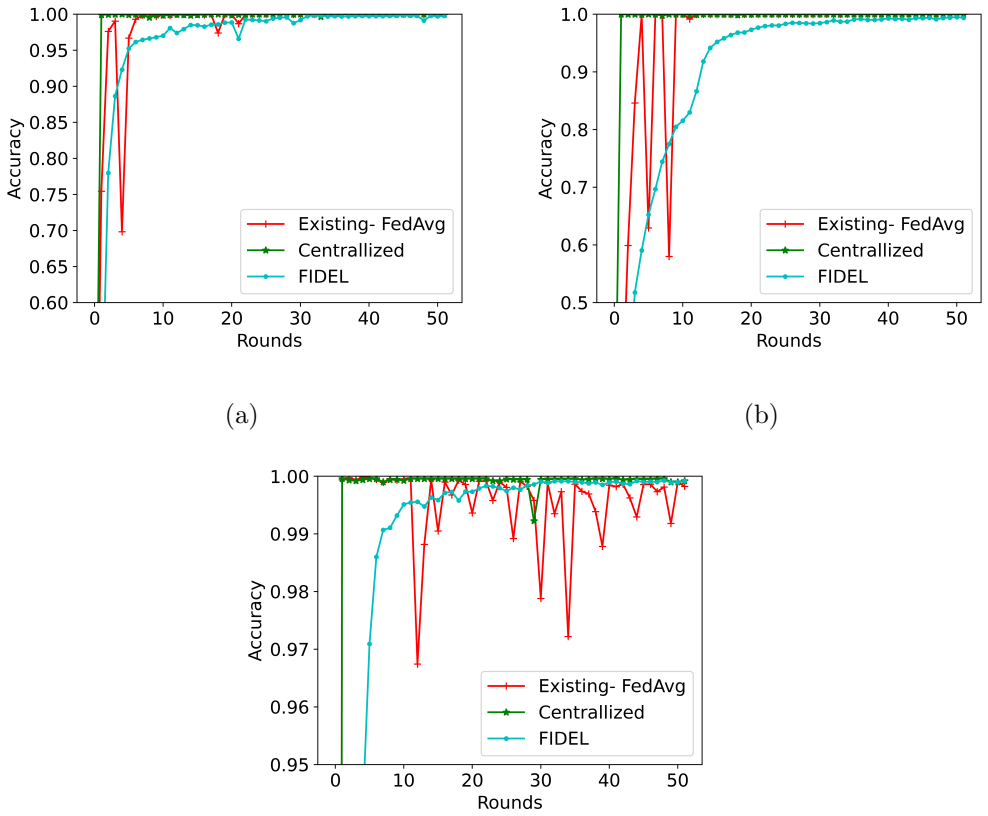


Figure 3.11: Training comparison on (a) shallow network (b) deep network (c) convolutional neural network model with existing-FedAvg, centralized training and FIDEL using synchronous aggregation strategies

training a neural network. We compared the proposed framework with a recently published framework and centralized training. The state-of-the-art implementation for SFL for neural network training is taken from a paper published by Stergiou et al. We implemented the existing framework [108] for neural network training with fedAvg implementation. Fig. 3.11 shows training comparison on all three models for synchronous aggregation.

As shown in Fig. 3.11, the FIDEL is able to train the neural networks and achieve similar accuracy to the centralized and existing work. Even though the existing work trains the model with the complete dataset on every client. Hence, it is hard to train on the edge of the network due to the lack of high computing resources on the fog layer.

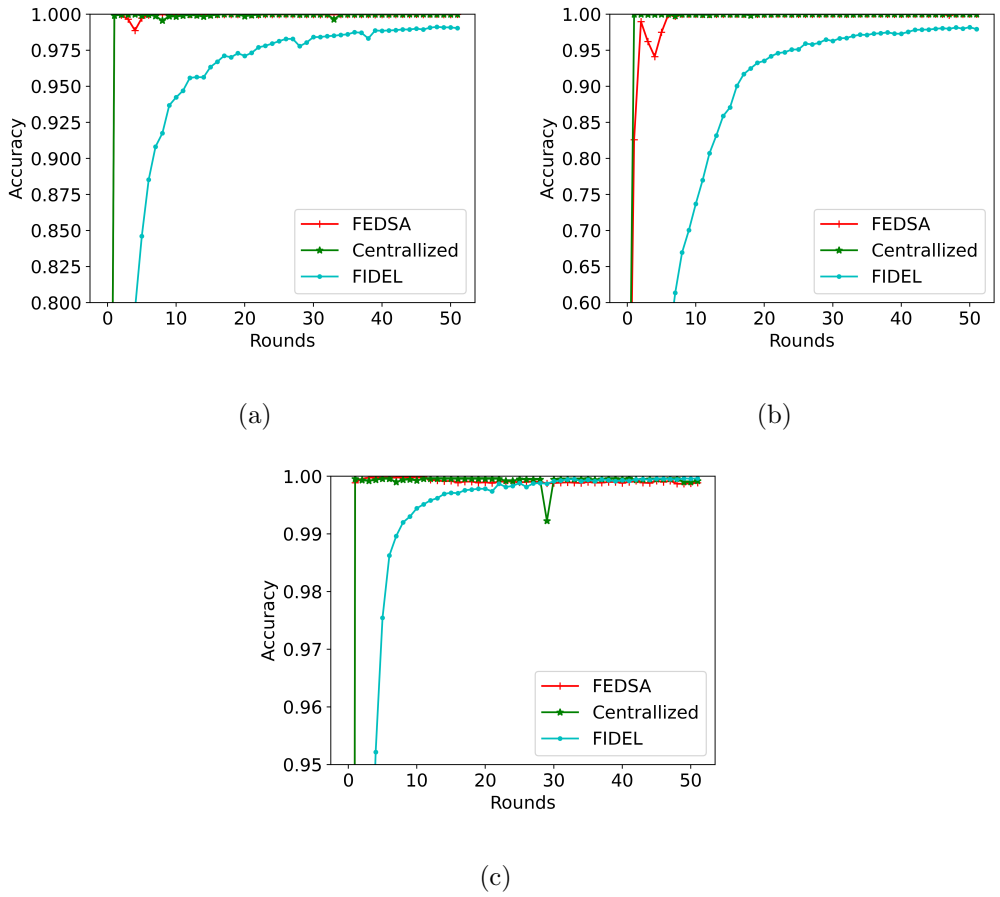


Figure 3.12: Training comparison on (a) shallow network (b) deep network (c) convolutional neural network model with FEDSA, centralized training and FIDEL using asynchronous aggregation strategies

However, the proposed online training is done on a subset of data that has achieved the same accuracy as existing work. But the SFL is subject to network and client availability. As the network/client disconnects, the entire training may freeze. At the same time, it is bounded by straggler nodes in the network. Hence, FIDEL is also trained and compared with an existing asynchronous FL framework FEDSA [74]. The training comparison of AFL is shown in Fig. 3.12.

The asynchronous online training also faced client dropout. Hence, at a particular iteration, there have been fewer updates than total clients. In spite of that, the AFL training achieves similar accuracy measures on all three models. This can be visualized by gradual learning of the model as shown in Fig. 3.12. The asynchronous strategy

also eliminated stragglers' impact on training. However, the asynchronous strategies take more time to converge because of asynchronous aggregation. That means at a particular aggregation step, there were fewer than the total participating clients. This may happen due to device computation time or client disconnection.

The result suggests that SFL is faster than AFL. However, it requires a stable network and efficient compute nodes. This can only happen in an ideal case that may be provided in a simulated environment or a well-established setup. However, resource-constrained devices are prone to failure, which makes the training insignificant. Therefore, the proposed online AFL can be a practical solution for distributed training. By doing this, we can train an equivalent machine learning model with a practical use case. Figure 3.13 presents a comparative analysis of the learning performance of both SFL and AFL across all models.

Asynchronous takes a longer time to converge because it suffers from stragglers or device failures. However, synchronous is faster and more efficient. So there is a tradeoff between time and accuracy. Asynchronous takes more time for training, but it has a low failure rate. However, the synchronous update can fail or be delayed due to client failure or network issues, but it has high accuracy and faster convergence. As training increases, the model improves its accuracy significantly. As shown in Fig. 3.13, both strategies converge to the global minimum, which leads to a similar accuracy level to centralized training. Synchronous strategy learns quicker than asynchronous, this is because of stable participants. Even though asynchronous update learns more slowly, it reaches a similar accuracy over a larger training time. Similarly, all models optimize their loss value and converge to the optimal minima, shown in the second column of Fig. 3.13. Experimental results show that all models have achieved 99% accuracy on test data. Fig. 3.14 shows the accuracy of all models on both communication strategies. It should be mentioned that the models got very high accuracy due to simplicity of the data.

All clients perform training on local data. However, testing is performed in the cloud with unseen test data. Fig. 3.14 shows the model's accuracy on test data with synchronous and asynchronous updates. We have also traced the performance of the global model on local data. Fig. 3.15 shows models' performance on participating clients' data. The accuracy of test data is averaged by cancelling the drift on various

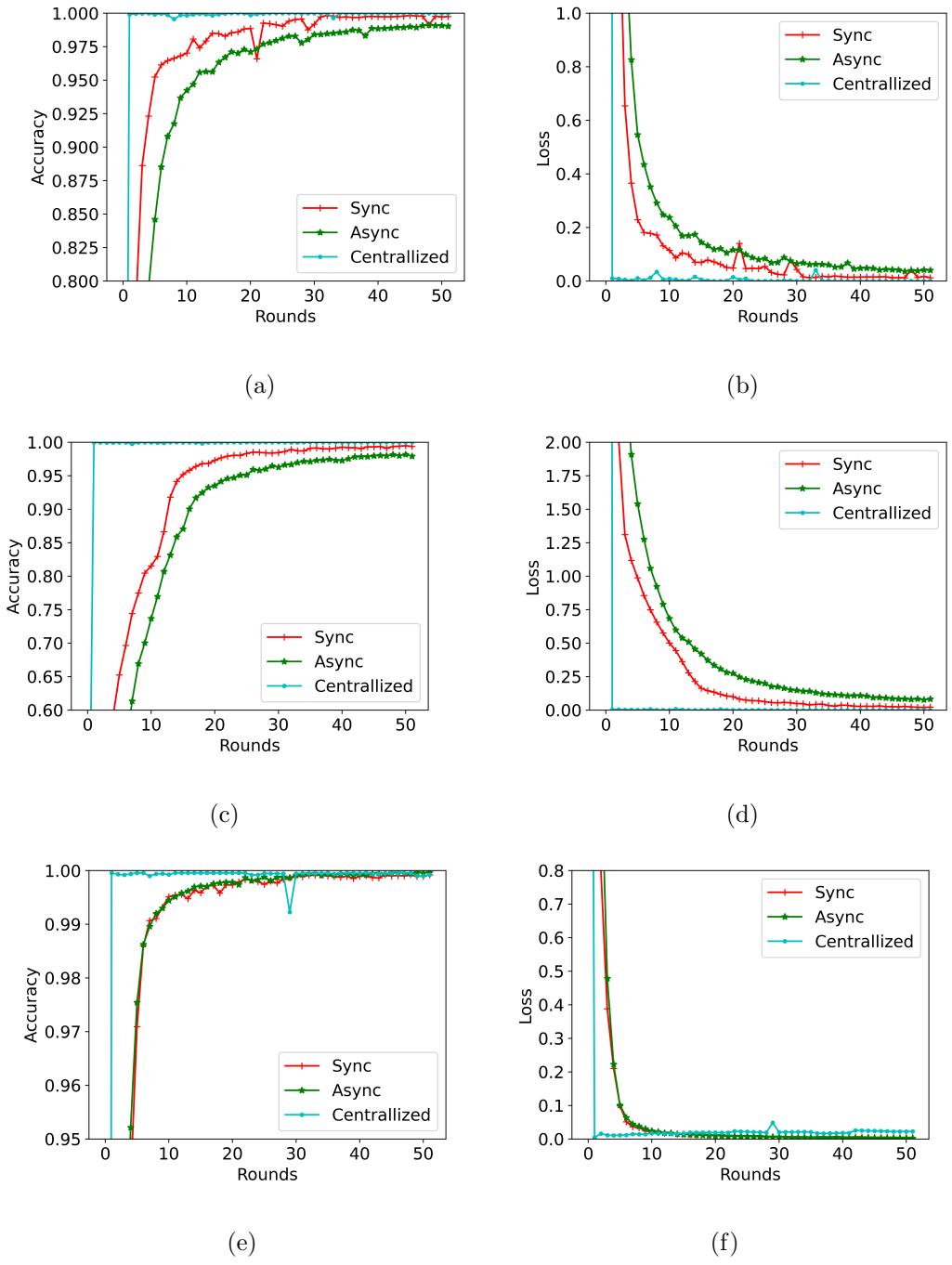


Figure 3.13: Training comparison of SFL and AFL with centralized learning. The first column indicates accuracy, and the second one is the loss value of (a), (b) Shallow network (c), (d) Deep network (e), (f) Convolutional neural network

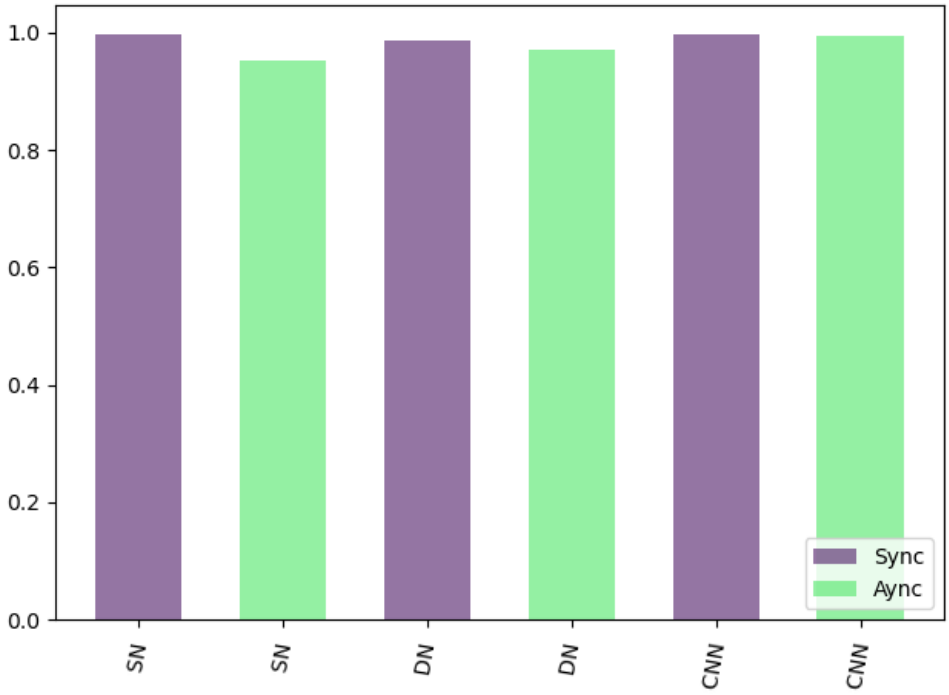


Figure 3.14: Model's accuracy on both synchronous and asynchronous updates

local training data shown in Fig. 3.15 for both synchronous and asynchronous. Hence, the proposed framework can efficiently achieve significant results on various models.

3.4.5 System performance analysis

As discussed previously, the proposed framework can train an equivalent global model compared to stable CPU/GPU-based training. Since FIDEL has used resource-constrained devices with online training, there is a need for system efficiency analysis. Therefore, we evaluated the framework's efficiency from a resource-constrained device's perspective. The framework is developed on a Raspberry Pi for system usage. We monitored memory consumption, total training time, computation time and communication time for synchronous and asynchronous strategies, while training the deep model. The size of the model is determined by its configuration. In our case, it is approximately 60 kB, which makes it lightweight and easily transferable over the network. Figure 3.16 shows a comparison between two strategies for total training time (computation + communication), computation time taken by pis while training and communication time for

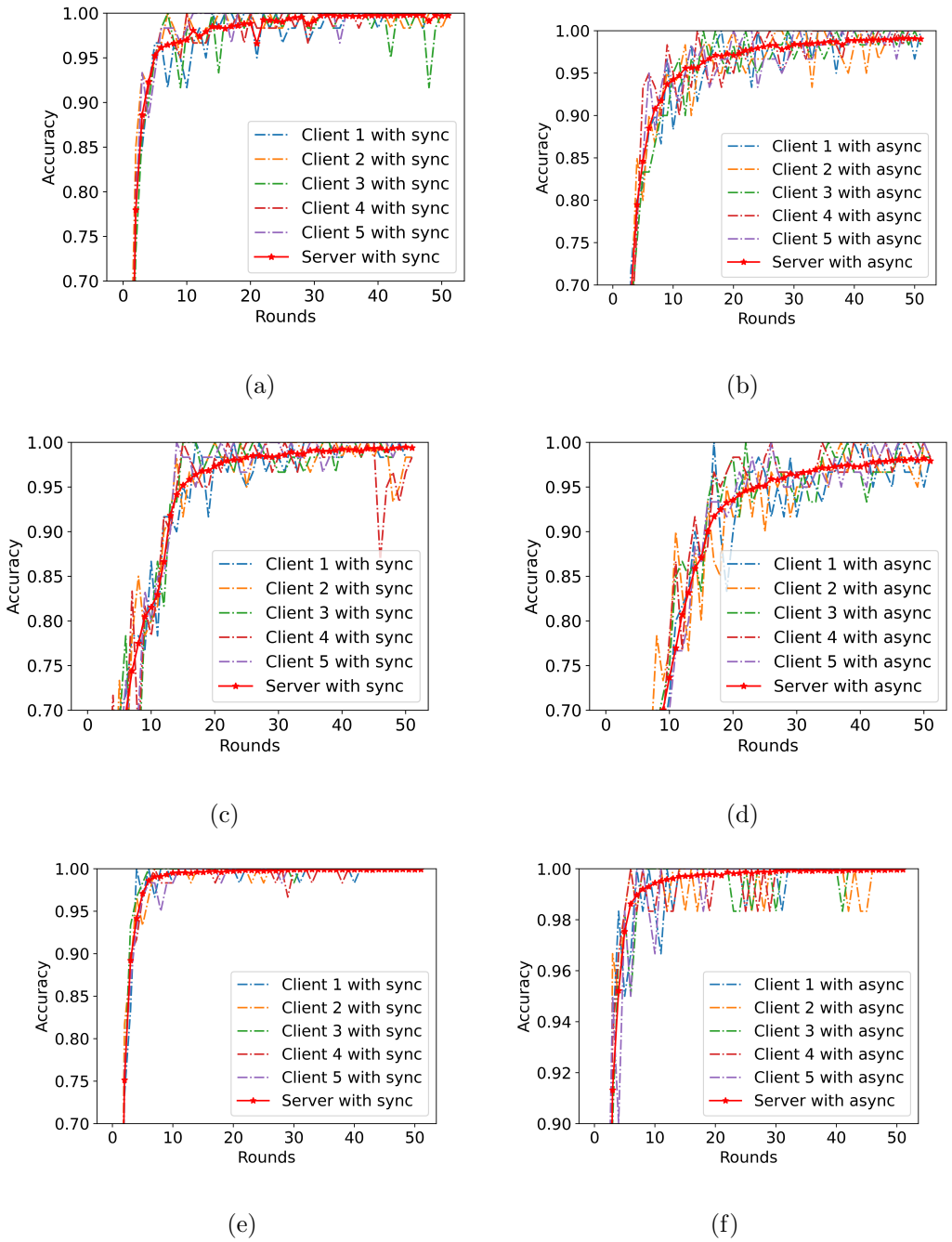


Figure 3.15: Global model performance on participating nodes using both strategies. The first and second columns are synchronous and asynchronous updates, respectively for (a), (b) Shallow network (c), (d) Deep network (e), (f) Convolutional neural network

model sharing over wifi per iteration.

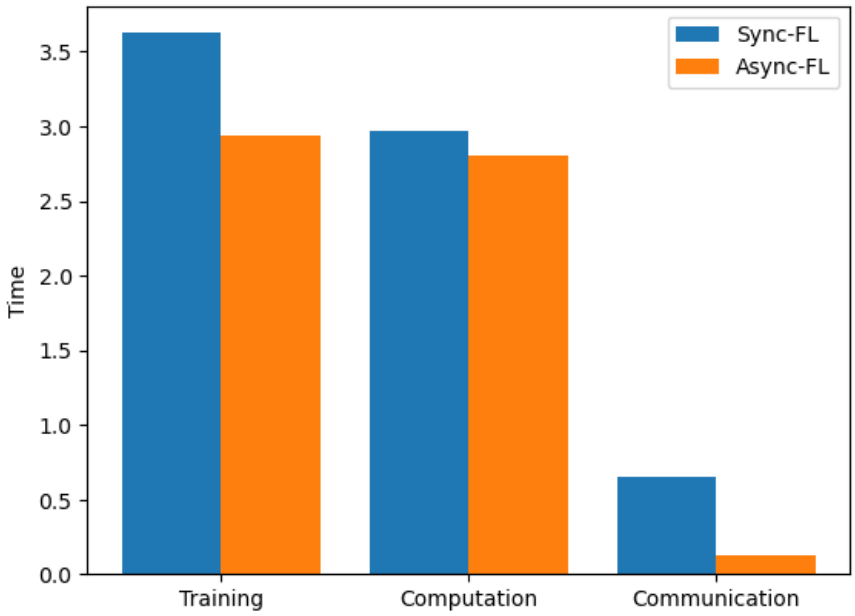
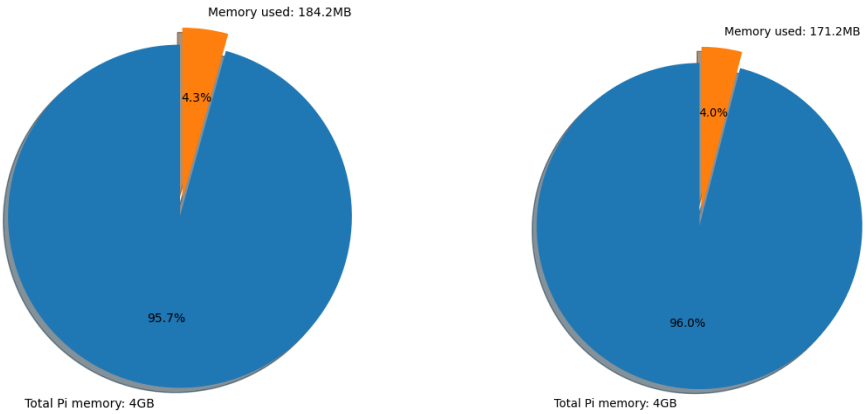


Figure 3.16: Average time taken (seconds) by the framework per round

On average, the total training, computation, and communication times are 3.62, 2.97, and 0.65 seconds, respectively, for synchronous update. However, the same metrics for asynchronous updates are 2.93, 2.80, and 0.12 seconds. As expected, the synchronous update took more time than the asynchronous update. This is because the server has to wait for all the clients to send their updates. The time difference can further shoot up when any straggler node is in the network (There were no stragglers in this experiment). The per iteration training time is ≈3s, for training mentioned deep network with 16 local epochs, which is reasonable for online training.

Next, we trace the memory consumption while training the framework shown in Figure 3.17. The memory usage of both strategies is similar and reasonable for training, even for larger batch sizes. In the current experiment, we used 60 frames that took an average of 184.2 MB in synchronous and 171.2 MB of memory out of 4GB of total memory. We have used lightweight Docker for the training, which can be easily deployed over heterogeneous hardware. The overall memory consumption was ≈ 4% of 4G RAM Raspberry Pi. The above system metric suggests that the framework is



(a) Synchronous learning (b) Asynchronous learning

Figure 3.17: Memory consumption of the framework

lightweight enough to train on resource-constrained devices that can achieve an equivalent centralized model.

3.4.6 Conclusion

In this work, we propose a fog-integrated distributed training framework for training neural networks. We focused on training a global model using resource-constrained devices on decentralized data. Federated training is done on continuously changing distributed data over different locations. The edge layer contributes to continuous data generation that has been taken care by the associated fog node. The cloud and fog nodes collaboratively trained a shared global model for classification tasks. Asynchronous federated learning is also implemented to address straggling and intermittent nodes. The FIDEL is capable of real-time data processing using resource-constrained devices such as Raspberry Pi.

We implemented both synchronous and asynchronous federated learning to showcase learning capability. The FIDEL framework is tested on multiple neural networks for human safety detection in HR workspace. The experimental results of all models on continuously changing datasets are significant. Synchronous learning is more efficient but suffers from stragglers. At the same time, asynchronous learning achieves similar accuracy in longer iterations. Finally, both strategies could learn an equivalent global model. However, FIDEL has to deal with stale updates and straggler's impact, which is addressed in future work.

3.5 Summary

This chapter discusses federated learning implementation on the edge-to-cloud continuum. It addresses the challenges of training machine learning models on resource-constrained devices with decentralized, continuously streaming data from an IoT network. To achieve this, it first proposes an architecture for federated learning on resource-constrained environments. It analyses the feasibility of training a neural network on a given IoT network. Then we extended it to build a framework that supports model training on devices like Raspberry Pi. We presented the FIDEL framework for training neural networks on the edge-to-cloud continuum with synchronous and asynchronous communication strategies. A detailed discussion of the development, methodologies, and prototyping processes has been presented. The implementation is done using Docker to achieve seamless execution on heterogeneous devices. Finally, we tested the framework for a real-world industrial IoT use case to detect human position in the HR workspace. With various experiments, we conclude that the framework converges equally to the centralized approach.

Chapter 4

Straggler Aware Federated Learning In A Heterogeneous Network

This chapter discusses about issue posed by the stragglers during processing. Stragglers are one of the key challenges in an IoT network that creates a bottleneck during model training. These nodes delay their update often due to hardware limitations, network congestion, or data distributions. This chapter examines how stragglers impact model training in federated learning. We worked in this direction to understand the impact of stragglers in an edge-to-cloud continuum. With multiple experiments, we found that it impacts model convergence negatively. To improve this, we propose a stragglers-aware weighted averaging scheme, FedStrag, that deals with stale updates and optimizes model training. Finally, the proposed method is tested and compared with baseline FedAvg on the MNIST dataset. The experiment results suggest that the FedStrag outperforms the baseline FedAvg in all possible stragglers and data distribution cases.

4.1 Introduction

Machine learning training has always been done on a centralized server with powerful computers. However, distributed machine learning trains a model across multiple machines. A general principle of training models in a distributed setup is to partition the data across multiple nodes and process it independently at each node. Federated learning executes similarly without sharing data, also called decentralized training. However, there are a few assumptions that are made in distributed machine learning

training. (i) Participating nodes are capable enough to perform training, which means it has powerful compute nodes. (ii) The network is stable enough for any communication. For example, consider multiple companies that wish to collaboratively analyze customer behavior without sharing their proprietary or sensitive data. They can train a global model with their local data without sharing with federated learning. such a cross silos training is done with powerful resources residing in their organization. At the same time, the network connectivity across the organization is considered to be reliable with high availability.

However, an IoT network has to deal with low resources and an unreliable network. The IoT ecosystem has evolved into a complex, multi-layered network that spans from highly distributed edge devices to cloud infrastructure. The edge-to-cloud continuum comprises a wide range of heterogeneous nodes that may differ significantly in both hardware capabilities and software configurations. Many of the devices within the IoT network are resource-constrained, characterized by limited computational power, memory, storage, and energy availability. For example, the network may have full-fledged computers, Raspberry Pis, mobile phones, or a Jetson Nano. These different types of devices have varying levels of computing power, which means they process data at different speeds. Therefore, they share model updates at different time intervals. Hence, in a federated learning paradigm where the server waits for clients' updates for aggregation, it creates a bottleneck during training. Furthermore, devices in the network may not be backed by a reliable power source, making them prone to failure due to battery depletion or power outages. Such devices often function as intermittent nodes, participating in the network only when power and connectivity conditions permit. Hence, the presence of stragglers poses a significant challenge to the practical implementation of federated learning.

Apart from this, communication among IoT devices often relies on wireless technologies such as Wi-Fi, cellular connection, and Bluetooth that can be affected by interference, signal strength, or other network issues. This makes the network vulnerable and unreliable for communication. For example, during frequent data transfer over the network, there is a high chance of network congestion that may lead to packet delay, and even packet losses. Training a machine learning model in such an unreliable network is challenging due to stale updates, as the server has to wait for all devices to send their latest updates. A common approach to mitigating the impact of stragglers

in federated learning is to exclude their updates during the aggregation process. While this strategy may be effective in large-scale networks with millions of nodes, it proves inefficient in smaller networks. In such scenarios, discarding updates from stragglers results in the loss of new information, as every node captures local patterns from its data. Since every node has learns a hidden pattern in the network, elimination is an inefficient way to deal with stragglers. As discussed in chapter 3, FIDEL addresses this issue by asynchronous update. So, rather than waiting for every update, the server waits for a fixed time before aggregation. Then the server creates a new global aggregated model by combining available updates and proceeds with the next round of training.

Though FIDEL ensures seamless training in the edge-to-cloud network, the impact of stragglers is still unknown. Here, we aggregate stale updates during aggregation without accessing their impact. Federated learning has enabled numerous applications with distributed training. However, stragglers are a critical bottleneck in the network. Simulation results for federated learning in cloud-fog are promising, but when it comes to deployment, it suffers from various challenges such as network dropout, device dropout, stragglers, etc. The impact of stale updates and straggler nodes on model convergence needs to be known. With analysis, we found that stale updates impact model training and delay convergence significantly if aggregation is not performed optimistically. So the aggregation process needs to be modified in case of stragglers. The analysis reveals that the main reason for the negative impact is the aggregation of the method. Since we apply the averaging method during aggregation, the stale update slows down the training process. This happens because stale updates are trained on an old model that lacks the latest knowledge. Hence, it delays the training over time and even diverges sometimes. Additionally, training machine learning models at the fog layer presents significant challenges due to inherent resource constraints and the continuous inflow of streaming data.

To this extent, we propose a novel federated weighted averaging approach, FedStrag, to resolve stragglers' issues. It is a penalty-based aggregation strategy that incorporates both stragglers and non-stragglers' updates. FedStrag prioritizes model updates based on their staleness values and accommodates stragglers during the aggregation process. Basically, it penalizes the stale update. It accommodates delayed updates in an asynchronous mode within the system, which has a positive impact on

model training in terms of time and accuracy. This approach does not discard training updates from any nodes, but stale updates are given less priority than the latest ones. Practically, the FedStrag prioritizes the latest update over old updates with respect to delayed time. With FedStrag, we propose an end-to-end federated learning paradigm in an unreliable fog network for image detection tasks. The training is designed to handle stragglers and optimize their impact in the network. The implementation is done to be scalable and trainable on heterogeneous nodes. The paradigm is trained on resource-contained devices with a containerization approach to deal with hardware and software heterogeneity. The core idea of the work is to accommodate stragglers' contributions in the learning process rather than discarding them. FedStrag optimizes overall learning towards generalization of the model in a smaller network. The paradigm enables the network to learn continuously over a larger dataset with heterogeneous resource-constrained devices. We added the FesdStrag strategy into the FIDEL framework to learn a machine learning model for multiple straggler scenarios that test the effectiveness of the proposed method. We will discuss the methodology, training, implementation, and results in the rest of the chapter.

4.2 Training architecture and model training

This section will explain network architecture for distributed training on the edge-to-cloud continuum. It will also formulate data flow over the IoT network for efficient model training.

4.2.1 System model

Consider an edge-fog-cloud network employed for training a machine learning model on continuous data. Federated learning training is being performed between the cloud and fog/edge layer. The fog network may have stragglers here, and a cloud node works as an aggregator. Assume there are $\mathcal{F}$ fog nodes, $\mathcal{K}$ edge nodes, and a central server in the network. Table 4.1 describes symbols used in the chapter. Fog nodes are resource-constrained devices with limited computational capabilities. They are connected to the cloud node over the wireless medium. Considering resource-constrained devices at the fog node with wireless connectivity, there is a high chance of stragglers and device failure, making the network notoriously hard to train in synchronous federated learning.

Therefore, we have developed time-bound asynchronous federated learning to ensure the continuity of the training.

Table 4.1: Symbol table

Symbol	Meaning
$\mathcal{K}$	Number of nodes in edge layer
$\mathcal{F}$	Number of nodes in fog layer
D_f	Total historical data available at a fog node
D_c	Data subset for current training data at a fog node
D_{tot}	Complete dataset from all nodes
τ	Number of data points a fog node can process
T_d	Server waiting time
p	Last frame index for data selection at a fog node
X_i	i^{th} input data for training
y_i	i^{th} output label for training
L_c	Loss function at a fog node with current data D_c
L	Global loss function
θ_i	Model parameters at i^{th} node consist of (W_i, b_i)
θ_g	Global model parameters at server
θ^*	Optimal global parameter
t_g	Current global iteration number
t_l^f	Iteration number on which the model is trained
D	Vector containing number of training samples
W	Weight vector for every model at server
E	Number of epochs for training at a fog node

However, due to resource limitations in the fog layer, there is not much room for large data training, specifically for continuously generated data that leads to big data problem. Therefore, we have adopted an online federated learning paradigm for continuous data flow [52]. Every fog node contains total historical data $D_f = \bigcup_{i=0}^{now}(X_i, y_i)$ for local training. However, due to resource constraints, the fog node selects a subset of data from historical data and trains on it. Similarly, in the next iteration, the fog node selects the next subset of data for further training. Therefore, the model is only trained on a subset of locally selected data, not on complete datasets D_f. The selection of data subsets is contingent upon the computational capabilities of each fog node. The criteria for data selection are based on the computational capacity of the fog node. Specifically, each fog node selects recently captured datasets denoted as D_c, referred to as the current training dataset. The selection of data is made as

$$D_c = \bigcup_{i=p}^{p+\tau}(X_i, y_i) \tag{4.1}$$

Where p is the last index of data selection, and τ is the number of data points a fog node can process in one iteration. Considering supervised learning, each data point has input samples X_i with label y_i. Let $l_i(\theta_g) = l(\theta_g, X_i, y_i)$ represent the loss function for i^{th} sample at the fog node of local data D_c. So, for a fog node loss function for the current training is

$$L_c(\theta_g) = \frac{1}{|D_c|} \sum_{j \in D_c} l_j(\theta_g) \tag{4.2}$$

The loss function is optimized with the gradient descent method over the input samples in D_c. The local parameter θ_i is fine-tuned with the local data as

$$\theta_i = \theta_g + \eta \nabla L_c(\theta_g) \tag{4.3}$$

The framework is trained with subsets of the data D_c, with online training, which is expected to learn hidden patterns in the complete historical dataset D_f. Therefore, the global loss function for the current training is defined as

$$L(\theta_g) = \frac{1}{|D_{tot}|} \sum_{c \in f} |D_c| L_c(\theta_g) \tag{4.4}$$

Where $D_{tot} = \cup_{c \in f} D_c$ is the collection of all local training datasets. The goal of the training is to learn a model that can establish input-output relations from current datasets D_c that should work equally well on unseen datasets. So, the system has to learn a model that can work well on the historical dataset D_f. Hence, overall system has to optimize the θ^* for all possible data $D_{tot} = \sum_{f=1}^{F} D_f$.

4.2.2 Distributed training

Collecting raw data at the central node and then processing it at the cloud may not be efficient for various big data and continuous data. It needs huge bandwidth, a large centralized processing unit, and storage. Additionally, due to privacy concerns, data owners refrain from sharing data. The federated learning paradigm enables decentralized learning over distributed nodes. The edge-fog-cloud continuum is well-suited for federated training. The network contains a cloud layer, a fog layer and an edge layer that has heterogeneous devices for data processing. The proposed FedStrag is implemented on edge-to-cloud architecture as shown in Fig. 4.1. The architecture consists

of three distinct layers, with each layer comprising various IoT devices assigned to perform specific tasks. In the horizontal layer, devices operate in parallel to execute their respective task. These devices then interact with the vertical layer to collaboratively learn and fine-tune the model. Although a similar architectural framework was introduced in Chapter 3, we briefly revisit it here to establish a more comprehensive context for the contributions of the present work.

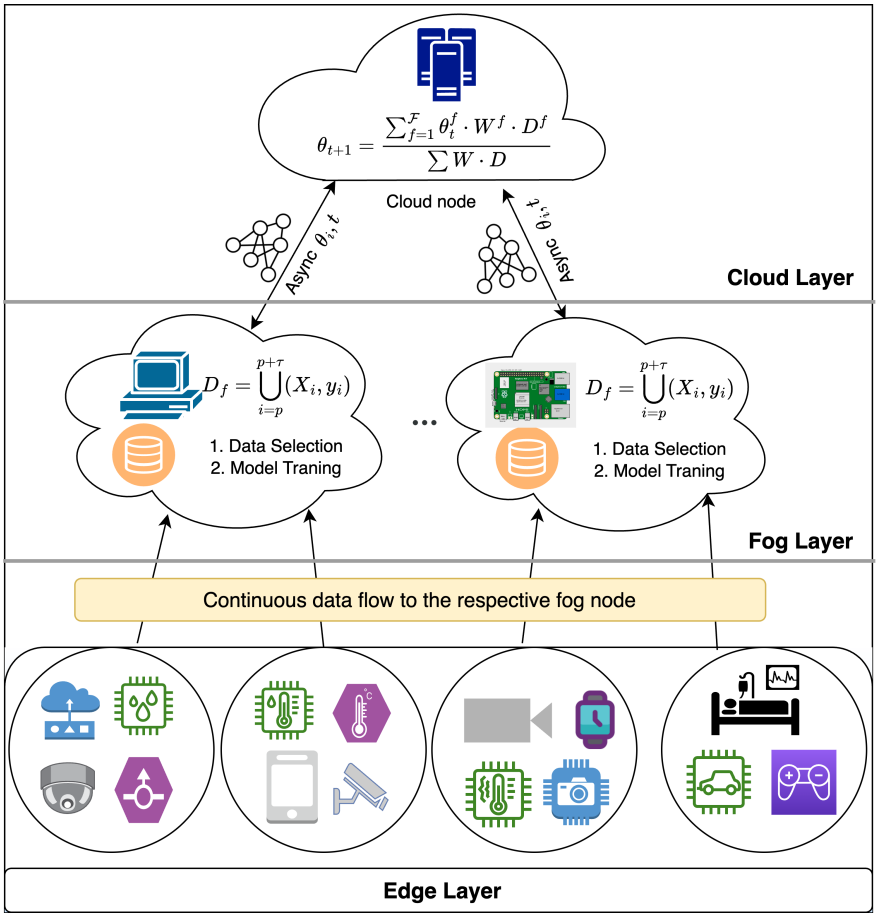


Figure 4.1: Overall architecture for model training with FedStrag

4.2.2.1 Cloud Layer

The main task of the cloud layer is to work as a central aggregator during global model creation. Nodes in the cloud layer are stable and computationally efficient. It coordinates with the connected fog nodes and instructs for model training. Once

the model is trained at the fog node, the cloud node aggregates it with the proposed FedStrag strategy and sends back the global model to the fog for further training. Cloud nodes are also responsible for the authentication of genuine contributors. Since it has high memory and processing, it can be used for other resource-intensive work like report generation or visualization. It works as an actuator for overall federated learning training.

4.2.2.2 Fog Layer

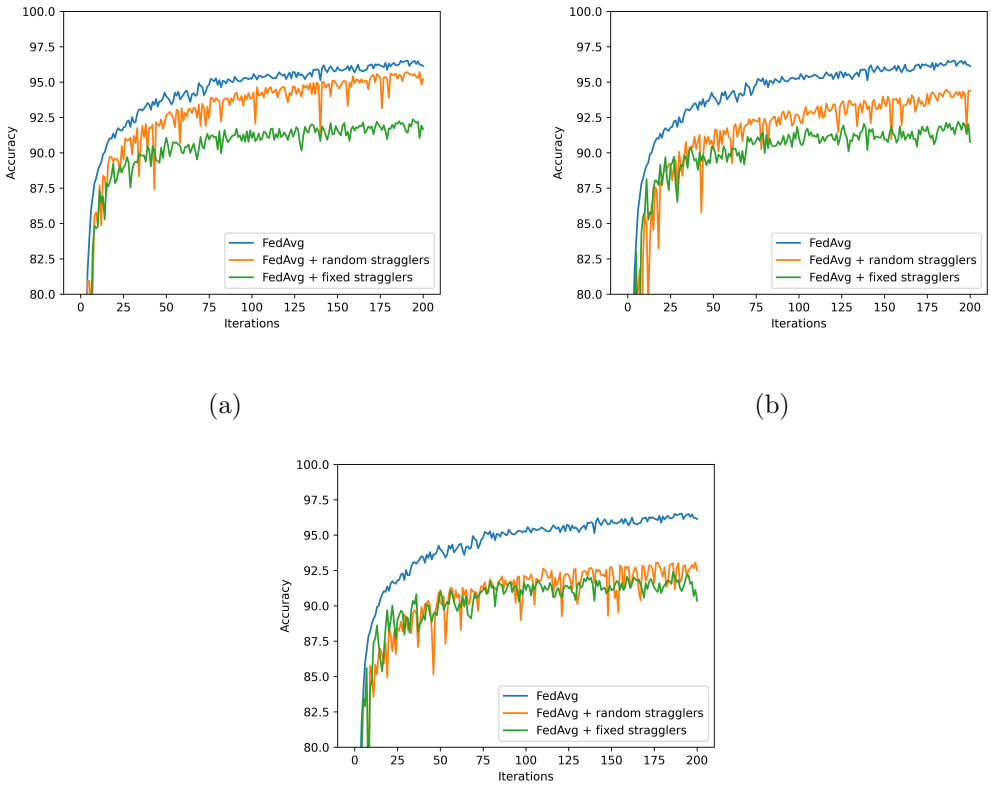
Fog layer contains sufficient processing devices such as desktops, laptops, Raspberry Pis, mobiles, etc. Nodes in this layer may not process huge amounts of data. However, they can process a limited amount of data. It works as an intermediate layer for processing the raw data. A fog node receives continuous data from the edge layer, which is stored in persistent storage. With the proposed FedStrag strategy, we process the continuous data by selecting fewer samples at a time. The main job of the fog layer is to train the local model on a selected dataset and thereafter coordinate with the cloud node for model aggregation. The nodes of the fog layer are connected to edge nodes to prevent any data leaks. It can be set up in the data owners' vicinity to protect against data breaches. This provides users' trust in the overall training network.

4.2.2.3 Edge Layer

Nodes in the edge layer are mainly data-generating devices such as sensors, GPS, cameras, smart gadgets, etc. These devices are not efficient for any machine learning task, however they can produce continuous data for monitoring and processing. Hence, edge nodes are responsible for data acquisition by sensing the environment. The data is sent to its connected fog node for storage and processing. Each edge node is tightly connected to its data owner's fog node.

4.3 Stragglers and their impact in model training

Not all devices in an IoT network will have equal computational efficiency. Hence, some devices will produce results earlier than others. Similarly, an IoT network may face challenges like network failure, power/battery shutdown, etc. In such a scenario, the client will delay the update (stale update), and the server has to wait for a longer or



(a) (b)

(c)

Figure 4.2: Straggler's impact on model training with random and fixed straggler of (a) 10 sec, (b) 15 sec, (c) 20 sec

infinite amount of time. We have tackled the issue with time-bounded asynchronous updates [52]. However, the impact of stale updates on federated learning needs to be investigated. Hence, we analyzed the impact of stragglers on model convergence. We took two possible scenarios for stragglers: (i) Fixed stragglers (ii) Random stragglers. In fixed stragglers, a predefined particular node produces stale updates. However, a random nodes produce stale updates in the random straggler scenario. Details about stragglers setup are explained in section 4.5.3. We trained a model with fixed and random stragglers to assess the impact of stale updates on training. Details about the model and dataset are presented in section 4.5.2. We set a time delay of 10 sec, 15 sec, and 20 sec for stragglers in these three experiments. The results are shown in Fig 4.2.

As shown in Fig 4.2, it is evident that the struggles impact the training negatively.

The model was trained with the time-bounded asynchronous approach on 5 devices. With increasing delay time, the training gap increases in both fixed and random cases. Stale training (Orange and Green) has a degrading effect with varying stragglers' times. In fact, the baseline FedAvg algorithm (Blue line) performs better in all cases. This signifies that the stale updates are pushing back the global model performance. According to convergence theory, with more iterations, the training is expected to converge to the global solution. However, with stragglers, the training either slows down or diverges over time. This is happening because the stale updates are trained on the old model.

Moreover, Fig 4.2 suggests that fixed stragglers are more prone to divergence of the model. This happened because the same node is producing stale updates on every iteration, so the global model suffers during aggregation. However, in the random stragglers case, a random device has submitted stale updates. At all other times, it submits the latest updates. Therefore, it averts divergence and slows down the training of the network Fig 4.2(a,b). However, if the time is increased to 20 sec, even the random straggler's model starts decreasing its performance like the fixed one. Hence, it is clear that even with asynchronous federated learning, the stale updates impact the performance significantly. Therefore, in this paper, we address the issue of staleness by weighted averaging.

4.4 Proposed FedStrag method for federated learning

We consider federated learning problem for optimizing a global model as

$$\theta^* = \underset{\theta_g}{\operatorname{argmin}} \sum_{f=1}^{\mathcal{F}} \frac{|D_f|}{|D|} L_f(\theta_g) \tag{4.5}$$

Where $\mathcal{F}$ is the number of fog nodes, $|D_f|$ is the number of data samples at node $\mathcal{F}$. So $|D_{tot}| = \sum_{f=1}^{\mathcal{F}} |D_f|$ is the total number of data samples over all fog nodes in the network. The optimal parameter θ^* is produced iteratively with the federated averaging algorithm like FedAvg.

4.4.1 Straggler-aware federated learning

In the fog-cloud network, we expect some nodes to be stragglers. Stragglers are those nodes that tend to delay submission due to various reasons, such as limited computa-

tional capability and network fluctuation. It is also expected that a node may fail due to power/battery outage, communication failure, etc, which may reconnect after some time. At the same time, a new device may connect at any time for training. Hence, synchronous federated learning is not a suitable paradigm for the training. Therefore, we developed time-bound asynchronous federated learning for the training. Here, time-bound asynchronous learning implies that the server will wait only for a fixed time interval T_d before aggregation at every iteration. The server waits and accumulates all locally trained models for T_d units of time. Thereafter, all available updates are aggregated for the next round as a global model. Since every node in the network contains crucial data that needs to be mined to learn generic patterns. Hence, this approach accommodates stragglers' contributions too. This is crucial for the smaller network with limited nodes/data, where we can not ignore a node's learning based on staleness.

However, the training of the straggler is based on old updates, which significantly impact global model convergence. At the same time, eliminating stragglers' contributions can fail to learn generic patterns in the data, specifically in a smaller network with limited data. Hence, we must jointly optimize both straggler's and non-straggler's training. In the time-bounded asynchronous FL updates, the server waits for T_d time before consolidation of the updates. Hence, in the T_d time frame, there are updates from both normal and straggler nodes. The baseline FedAvg aggregates every update with normal averaging that eventually slows down model convergence. This happens due to assigning equal priorities to every node, including stragglers. Since stragglers are a few iterations behind the current iteration, it drags back the optimal parameter θ^* by its delayed factor. Therefore, in order to accommodate stale updates, we have taken a weighted averaging approach for efficient training. The idea is to penalize stale updates but accommodate their contribution during aggregation.

The proposed FedStrag is a staleness-based federated averaging scheme for prioritizing updates before aggregation. The main idea of the FedStrag is to assign less priority to the stale update and more priority to the latest one. It prioritizes an update based on its delay factor. To achieve this, the server needs to keep track of the time of each update. So, rather than sending only parameters θ_g to the client nodes, the server also sends time t along with parameter θ_g. The client trains the model with current local data and returns the updated model, along with the time, to the server for aggregation. During aggregation, the server aggregates available models based on

the weighted factor of every update. Here, we assign weights to every update based on the staleness of the update, which is calculated as

$$w^f = \frac{1}{t_g - t_l^f + 1} \tag{4.6}$$

Where t_g is the current global iteration and t_l^f is the iteration on which the model is trained on f^{th} fog node. Equation 4.6 ensures that weights for the latest parameters will be high, and stale updates will be low during aggregation. The final equation for FedStrag training for model aggregation is as follows.

$$\theta_{t+1} = \frac{\sum_{f=1}^{\mathcal{F}} \theta_t^f \cdot W^f \cdot D^f}{\sum W \cdot D} \tag{4.7}$$

Where D is a vector containing the total number of samples on which the respective model is trained, and W is the weight vector for every model calculated with Equation 4.6. FedStrag training is given between cloud and fog nodes with stale updates. Figure 4.3 illustrates the training loop and the aggregation process.

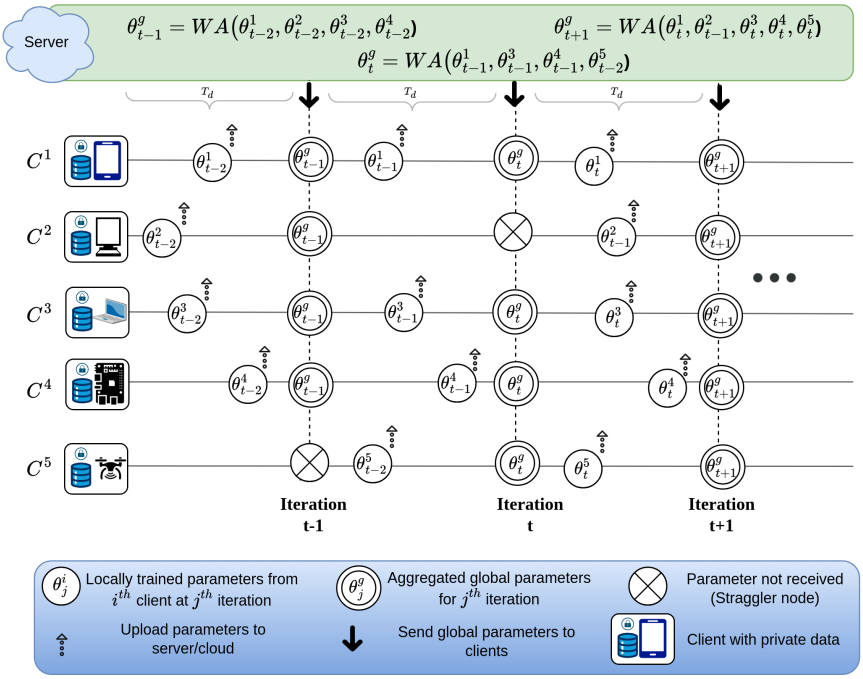


Figure 4.3: FedStrag based federated learning training

An i^{th} client sends its locally trained parameters independently. The server performs

proposed weighted aggregation on available parameters based on Equation 4.7 as shown in Fig 4.3. It waits for T_d units of time (shown as an arrow) and then processes weighted aggregation with available updates. If an update is received in the next iteration, it is processed at the next round with a reduced weight. The algorithm for the FedStrag is described in Algorithm 3.

Algorithm 3 Federated learning with FedStrag

1: Initialize empty global parameter θ_g and global iteration $t_g = 0$
2: **for** $t_g = 1$ to N **do**
3: **for** for every client node **do**
4: **Call** Client Execute in parallel
5: **end for**
6: wait for T_d unit of time
7: **Call** Server Execute for model aggregation
8: **end for**
Server Execute
9: **Input:** Locally trained parameters, iteration, no of samples $(\theta_1, t_1, d_1), (\theta_2, t_2, d_2), ..., (\theta_\mathcal{F}, t_\mathcal{F}, d_\mathcal{F})$
10: **Output:** New global parameter θ_g
11: **for** `i = 1,2,3,...,`$\mathcal{F}$ **do**
12: $W[i] = \frac{1}{t_g - t_i + 1}$ ▷ Weight Computation
13: $D[i] = d_i$ ▷ No of training sample
14: $\theta_g = \theta_i \cdot W[i] \cdot D[i]$
15: **end for**
16: $\theta_g = \frac{\theta_g}{W \cdot D}$
17: **return** Global parameter θ_g and t_g to fog clients for further training
Client Execute
18: **Input:** Current global parameters θ_g, E, p, τ, t_g
19: **Output:** Locally fine-tuned parameters θ_l
20: Select current local data D_c from historical dataset D_f using Equation 4.1
21: **for** `e = 1,2,3,...,E` **do** ▷ ML training
22: Update θ_l on data D_c with SGD using equation 4.3
23: **end for**
24: $p = p + \tau$
25: **return** Final parameter θ_l and local t_l to the server

As stated in Algorithm 3, FedStrag has two modules (i) server execute (ii) client execute. 'Client execute' is performed by the fog nodes in parallel. The main functionality of the 'client execute' is to select the data and perform training. The 'server execute' is performed by the cloud node which is responsible for model consolidation with the FedStrag strategy. The execution starts from the server by asking for training to available clients. At first, each client fetches the latest data for training (line 20), then it trains the current global model with its local data (lines 21-23) and sends it to the server. To avoid straggler's impact on training, it only waits for T_d unit of time (line

6). Thereafter, it calls 'server execute' for aggregation of currently available updates. The server calculates staleness and data sample vector for weighted averaging (lines 12-13). Thereafter, weighted averaging is applied on all available updates that create new global θ_g (lines 14-16). In the subsequent step, the current global θ_g and current iteration t_g are sent for further training rounds.

4.4.2 How does FedStrag work?

The central idea of FedStrag is to utilize everyone's contribution by incorporating their learning into global knowledge creation. It prioritizes client's update based on the stale factor shown in Equation 4.6. An update with high staleness is given low value in the weighted averaging, as stale updates are trained on the older models. However, it carries some information/pattern that needs to be included during aggregation. Unlike the standard FedAvg algorithm, the proposed FedStrag gives a smaller weight to stale updates and a higher weight to the latest update in global model creation as discussed in Algorithm 3.

Suppose, at 5^{th} global iteration, three updates are available for the aggregation. Assuming each node is being trained on a different set of data D_i i.e D = [60, 20, 80]. Out of the three updates, assume only two of them are latest ones. So there is no delay between them as iteration numbers of both updates $t_l^1 \& t_l^2$ are 4. However, last update is delayed by one iteration i.e. it arrived with $t_l^3 = 3$. So, the difference between global iteration and update is 2. Hence, at 5^{th} global iteration, weight factor W will be calculated as W = [1, 1, 0.5] using Equation 4.6.

Given the above scenario, the FedStrag will calculate the next global θ_6 as

$$\theta_6 = \frac{1 \cdot \theta_4^1 \cdot 60 + 1 \cdot \theta_4^2 \cdot 20 + 0.5 \cdot \theta_3^3 \cdot 80}{60 \cdot 1 + 20 \cdot 1 + 80 \cdot 0.5} \tag{4.8}$$

So during aggregation at 6^{th} iteration, the global model will have full contribution of θ^1 and θ^2. However, the contribution of θ^3 is reduced to 50% because of staleness by one iteration. Similarly, if the delay were 3, the contribution would be reduced by 1/3.

4.5 Evaluation and results

This section will discuss about experimental setup and evaluation of FedStrag. We evaluate the performance of the FedStrag on both IID and non-IID data for digit

recognition tasks. Additionally, we evaluated the approach with various straggler settings. The implementation is done with the online training scheme for large datasets on resource-constrained devices. The codes and results are available on the cloud and smart lab's GitHub[1] page.

4.5.1 Prototype and experiment setup

We created a prototype network of five Raspberry Pi devices working as a client node and one Intel Xeon 16 core CPU 3.7 GHz desktop computer with 32 GB RAM as a server node for federated training. We chose Raspberry Pi as a resource-constrained device because of its low hardware configuration. All Raspberry Pi devices have 4GB RAM, 1.5 GHz CPU with Raspbian operating system. A local network is created over WiFi that connects clients and server. The communication between clients and server is done using MQTT protocol. MQTT is a lightweight publish subscriber based message queuing protocol for device to device communication. Any authenticated device can join the network and subscribe to the relevant topic to contribute in the training anytime. This also ensures scalability of the system. Similarly, considering device heterogeneity as a challenge, we adapted containerization approach for code execution. Docker containers are deployed for program execution on various hardware. A Docker provides OS-level virtualization for software to run in isolation. It is a lightweight weight package that provides a complete execution environment for running code. The image contains all relevant packages and dependencies. Hence, it can be deployed on any hardware that supports containers.

4.5.2 Dataset & model

We conducted the experiments on publicly available MNIST digit datasets [56]. The dataset contains a total of 70,000 (60,000 training + 10,000 testing) image samples containing handwritten digits. Each image is categorically labeled as a number based on the digit written on it. The dataset has a total of 10 classes between 0 to 9. The dataset is freely available and used in various state-of-the-art research articles as proof of experiments. Additionally, the dataset is labeled and structured, which gives us more freedom to convert it into various forms for our experiments, such as IID & non-IID

[1] https://github.com/cloud-and-smart-labs/FedStrag

cases. The entire data set is distributed over 5 clients. For this experiment, we have utilized the dataset for both IID and non-IID scenarios.

For IID, the dataset is shuffled and distributed uniformly over all five clients. So, each client received 12,000 training samples. It is also ensured that each client receives all class data. However, in non-IID, every client received only two classes of data. So, for the given 10 classes (0-9), client 1 receives datasets with labels 0 and 1. Similarly, clients 2, 3, 4, and 5 have datasets with labels 2 & 3, 4 & 5, 6 & 7, 8 & 9 respectively. Therefore, every client's data distribution is different, and they train the model with isolated classes. This case is well suited for the strong client data bias.

In the experiment, we have trained a CNN architecture for the digit recognition task. The model is relatively lightweight, so it can be trained on Raspberry Pi. It has two convolutional layers that use 5×5 kernel size with 6 and 16 activation units. We have used relu activation function as a non-linear layer. Both convolutional layer is followed by 2×2 max-polling layer. Finally, we have used 120 dense units and an output layer with a softmax activation function. The total number of trainable parameters is 34,622. We trained the model with 'sgd' optimizer with categorical cross-entropy as a loss function. The training is done for both IID & non-IID data with varying straggler settings.

Since we have low-resource devices for the training, even though the model is relatively small, it can not process huge amounts of data in a single iteration. So, we have used online training that includes training a data partition only once. Assuming that the client node is producing one data point per second, and we are processing the data every minute, which utilizes 60 data points per iteration. We choose 60 data points at a time for every global iteration. In the experiment, we run 200 global iterations with 16 local epochs. This setting is based on hardware capabilities, which can be modified for different configurations.

4.5.3 Stragglers setup

An IoT network has a variety of hardware that may go out of the network at any time. So, it is expected that a few nodes will struggle for an on-time response, called straggler nodes. In the experiment, we incorporated stragglers of various kinds and then evaluated FedStrag's performance on both IID and non-IID datasets. We have taken two aspects of the stragglers (i) A particular node is a straggler. This scenario is fit for

a node that is having some issues in sending updates on time. This could be because of computational inefficiency or device failure. (ii) Random nodes are a straggler. This scenario may arise when there is an issue in the network propagation or a mix of computations that delays the delivery. In both cases, the straggler's contributions are incorporated based on the weighted averaging strategy. Similarly, device failure is inevitable in the IoT network. So, if any client gets disconnected at any point, it should reconnect and contribute seamlessly. The implementation of FedStrag can handle both device failure and stragglers at the same time.

For this experiment, we chose two strategies for the straggler devices. In the fixed strategy, we forced one device (client 3) to be a straggler. However, in a random strategy, any device will delay its updates with 0.25 probability. Hence, at a given iteration, a device is set to straggle with 25% of the time. Thereafter, we tested the learning capabilities at varying straggler times. The length of the straggler is controlled programmatically. Straggler's time of a device is predetermined with timestamps of 5 sec, 10 sec, and 15 sec. The mentioned time is suitable and aligns with multiple epochs of training time so that the delayed update is stale with at least one iteration.

4.5.4 Results and analysis

We tested the FedStrag strategy for model training and compared it with the baseline FedAvg algorithm. Datasets for every device were allocated before training based on IID and non-IID settings. First, we trained the model on IID data with a fixed straggler. Here, 1 out of 5 devices (node 3) was straggler over the entire training. The training result is shown in Fig 4.4.

As shown in Figure 4.4, with a smaller straggler delay (5 seconds), both FedStrag and FedAvg ultimately achieved comparable performance, although FedStrag exhibited better accuracy during the initial training rounds. Because 5 sec of server waiting time for asynchronous FL did not produce more delayed updates. At the same time, any update is hardly delayed by one iteration. That's why FedStrag performed marginally better in the first 100 epochs, but over longer epochs, both strategies converged equally. However, in longer straggler's time (10 sec and 15 sec), FedStrag converges faster than FedAvg. The difference in training with baseline FedAvg is significant over time. FedStrag has better and more stable performance shown in Fig 4.4.

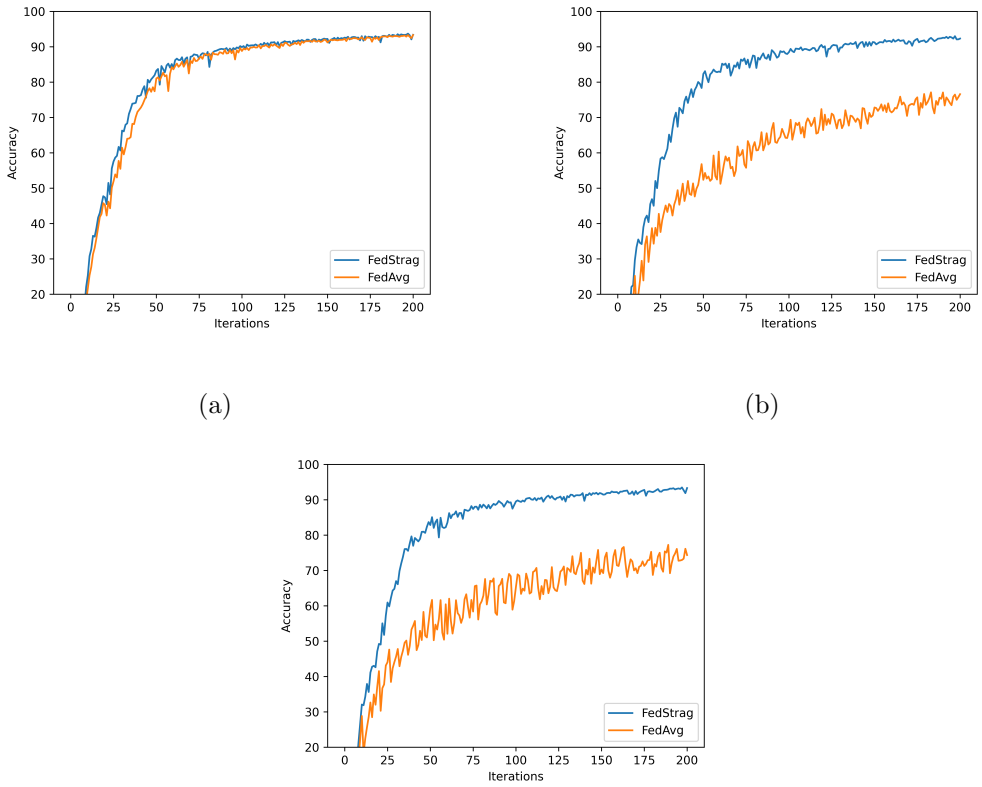


Figure 4.4: Training comparison on IID data with fixed straggler node-3 with varying straggler time

Similarly, Fig 4.5 shows FedStrag performance on non-IID data. It again converged as equal to FedAvg for a shorter stale time. We have chosen non-uniform data distribution for non-IID settings, which is statistically hard for training in federated learning. Even though the general accuracy of the model is less than IID, which is expected because of the data bias. Still, the advantage of FedStrag over FedAvg is clearly visible. It is more stable, better, and efficient for training with a larger stale time. Both results suggest that FedStrag performs better than FedAvg with a larger delay time. In an IoT network, stragglers are expected; hence, FedStrag can be an efficient strategy in loosely coupled IoT networks. However, FedAvg seems to be performing better in certain iterations, but the model is fluctuating more over two data points. It is more stable throughout the training. Although accuracy is limited, the edge of FedStrag

over FedAvg is clearly visible. Notably, the situation happens mainly for non-IID cases because of statistical heterogeneity in data, which is a known limitation of federated learning. In this case, we have strict heterogeneity with no common classes. Still, FedStrag is more consistent and stable in terms of training.

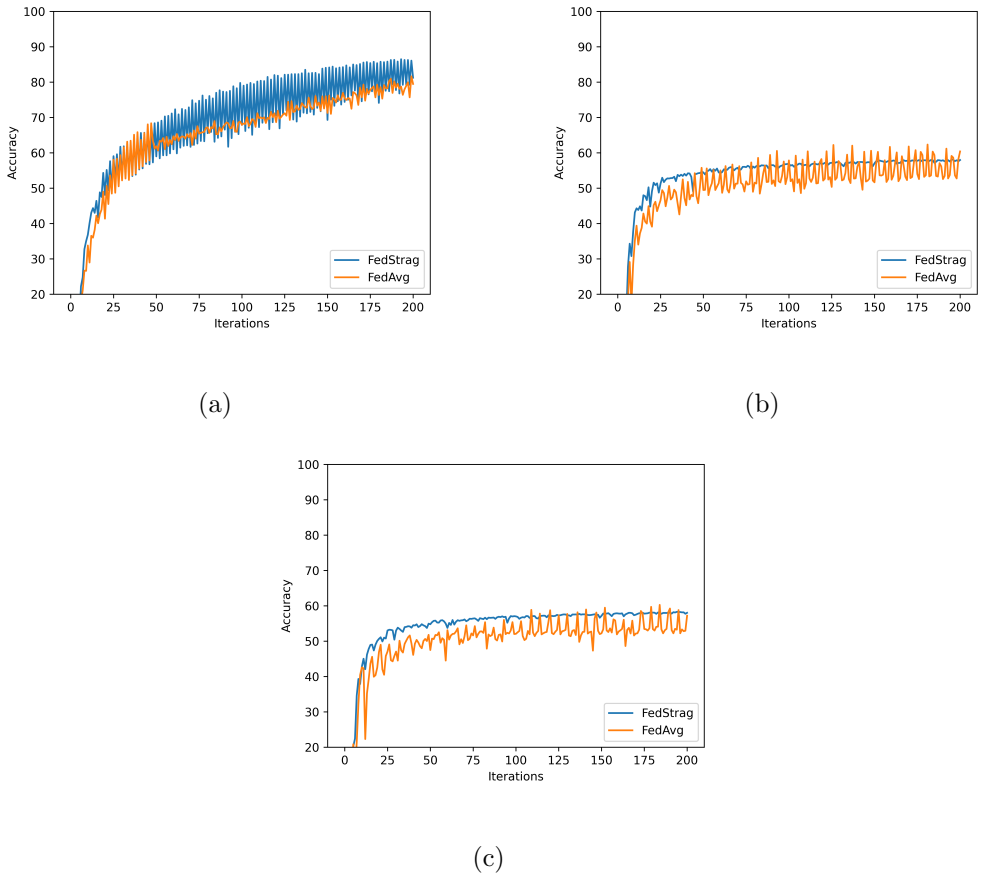
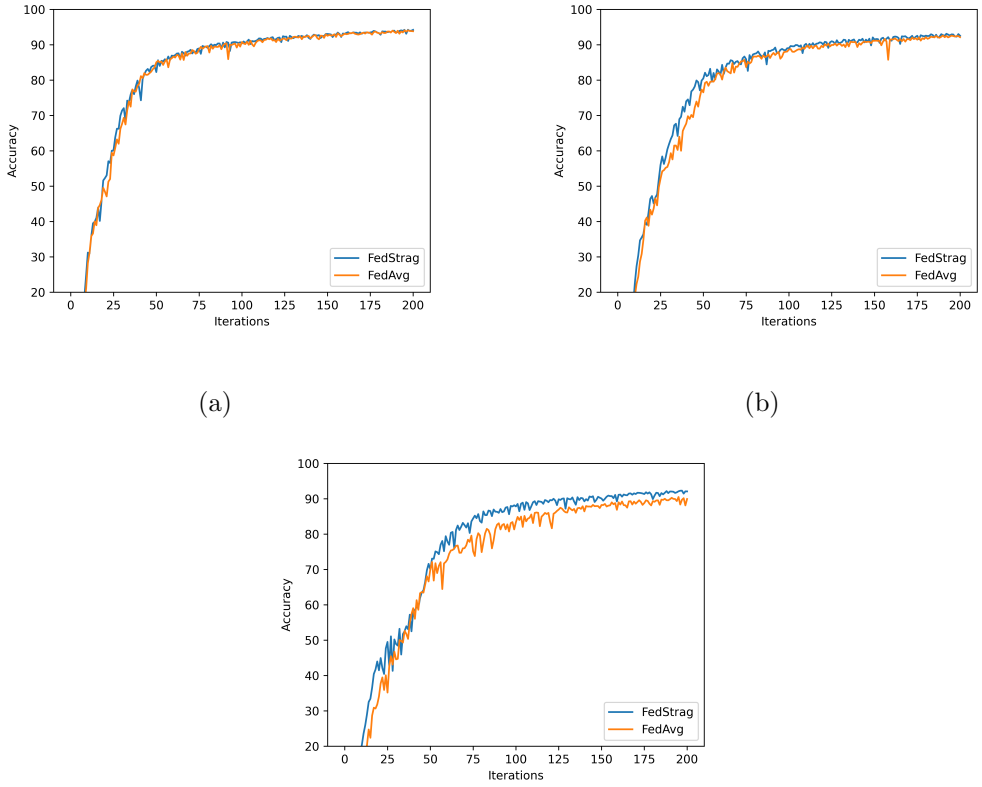


Figure 4.5: Training comparison on Non-IID data with fixed stragglers Node-3 with varying straggler time

Next, we evaluate the performance of FedStrag with a random straggler. In this setup, nodes are forced to delay the update with a probability of 0.25 on both IID and non-IID data. The training result is shown in Fig 4.6. Like fixed cases, FedStrag has performed equal to or better than baseline FedAvg. As shown in Fig 4.6, the margin of the difference is less in smaller delay times, but as the delay time increases, FedStrag starts outperforming the baseline algorithm. Since, at every iteration, straggler nodes are changing, the risk of learning drift is comparatively lesser than fixed stragglers.

This can be seen in the 5 and 10 sec staleness in Fig 4.6 (a & b). However, as delay time increases to 15 sec, FedStrag performs better than FedAvg shown in Fig 4.6 (c).

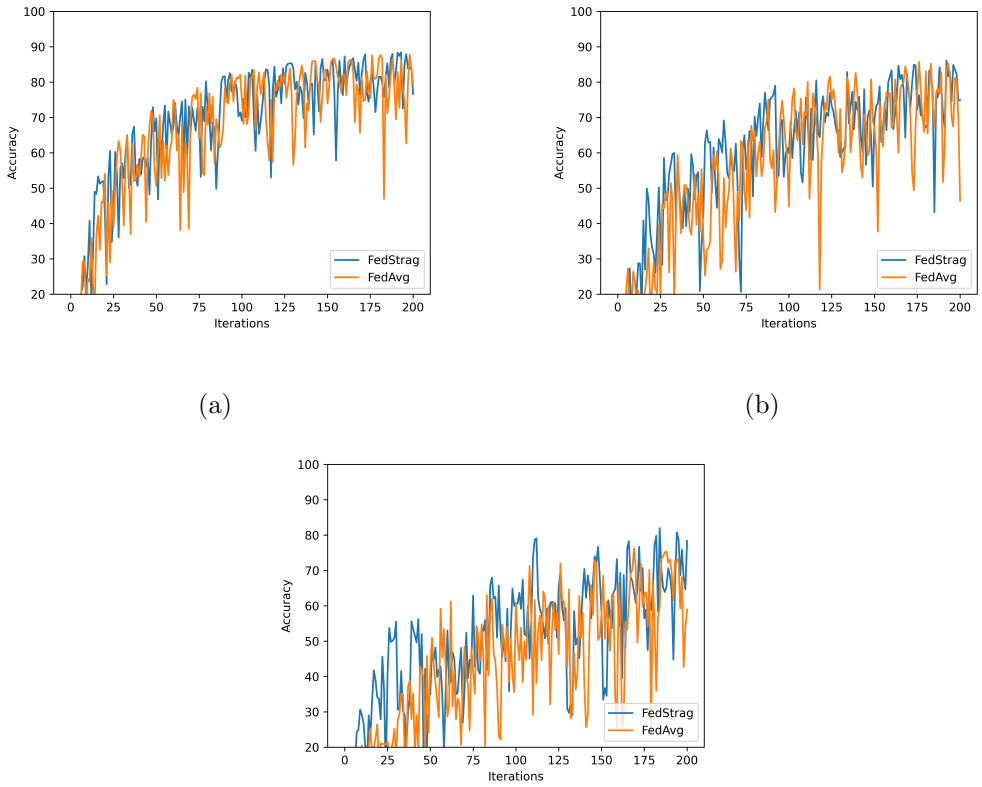


(a) (b)

(c)

Figure 4.6: Training comparison on IID data with random with 25 % stragglers

Finally, we tried to train the model on non-IID with random stragglers. We considered the same strategies for straggler nodes. The results are shown in Fig 4.7. In this scenario, the model has failed to converge properly. Although it tries to achieve accuracy but there is too much variation between the two rounds. This happened due to the combined impact of both staleness and non-IID nature of the data. Since data at each node is non-overlapping, the federated learning could not train it, which is a known limitation of the paradigm. Further staleness of the update has impacted the learning. Hence, it could not perform on test data. To address the issue, there is a need for a better strategy that can deal with non-IID data, which is the scope for future research.



(a) (b)

(c)

Figure 4.7: Training comparison on Non-IID data with random with 25 % stragglers

In this scenario, the model lags in training properly due to the strict non-IIDness of the local datasets. However, FedStrag still performed equally to FedAvg.

In real-world applications, the probability of having a straggler is high in an IoT network. Additionally, resource-constrained devices in a heterogeneous network will have delayed updates due to a mismatch of fast and slow devices. At the same time, device connectivity and model sharing add uncertainty to the network. Hence, straggler-aware federated learning can boost the learning process to build a generalized model by incorporating everyone's learning. As shown in experiments, FedStrag is capable of learning efficient models in an IoT setup. It converges faster than the baseline FedAvg algorithm. The empirical evaluation suggests that the model trained with the FedStrag strategy performs better on unseen test data. Fig 4.8 shows model accuracy on test data. FedStrag performed significantly better in fixed with larger straggler time. In-

terestingly, FedStrag has never underperformed in comparison to FedAvg in any of the experiments. It has either performed equally to or better than the baseline algorithm. Therefore, the proposed strategy can significantly boost federated learning training in an IoT setup for real world use case scenarios.

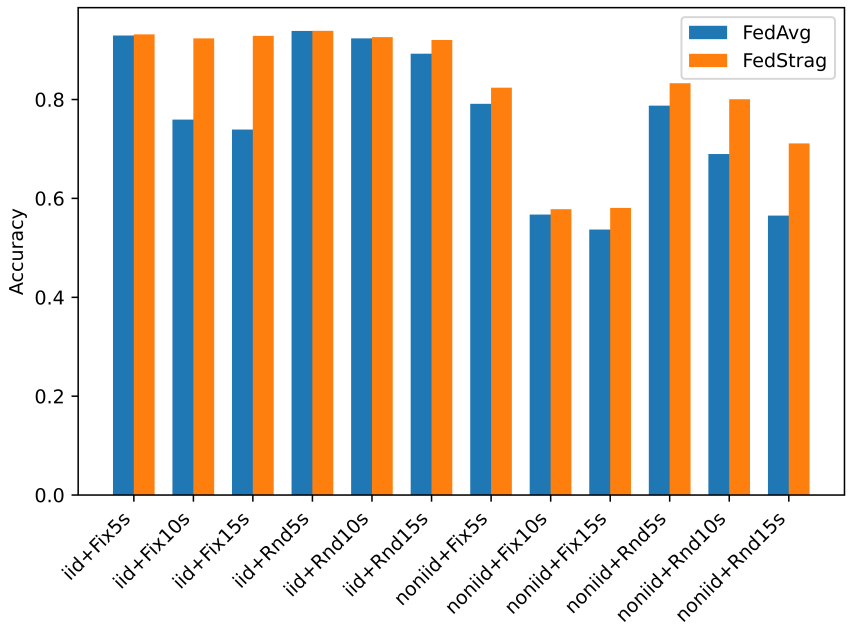


Figure 4.8: Performance comparison of both strategies on IID and non-IID data

In an IoT network, efficient resource usage is critical due to the vast number of connected devices and the often-limited computational power and energy resources available. IoT devices typically rely on constrained hardware with limited battery life, processing power, and memory, making it essential to minimize the energy consumed during data transmission and computation. The size of the model is determined by its configuration. In our case, it is approximately 162 kB, which makes it lightweight and easily transferable over the network. We conducted a few experiments to check resource usage for the proposed FedStrag, shown in Table 4.2. FedStrag is efficient than baseline FedAvg. It is approximately 1.7 seconds faster and consumes less power. Values in Table 4.2 are obtained experimentally on Raspberry Pi over a local wifi network. Here, computation time is the time taken by Raspberry Pi to train the model on 60 frames. Communication time is the time taken to send the model to the server. Total turnaround time is the time to calculate one round. Energy is calculated based on

the standard power consumption of the Raspberry Pi and WiFi. These values suggest that FedStrag is more efficient in consuming power. Even though both FedAvg and FedStrag have equal computation time, FedStrag consumes less communication and produces better results in terms of accuracy.

Table 4.2: Average resource usages

	FedAvg	FedStrag
Computation time(s)	5.279612398	5.255433941
Communication time(s)	4.187181759	3.735132936
Total turnaround time(s)	11.46911464	9.685097802
Power consumption(w)	109.158791	100.8671701

Further, we performed a statistical test to check the significance of FedStrag's model with both strategies. We performed a statistical test with the hypothesis that the FedStrag has the same performance. That means it has not had significantly different results. We performed a t-test on each trained model with 10 cross-validation on the test data. All models are passed through a t-test with FedAvg. With 95% significance, the test result is reported in Table 4.3. The p-value of all models is less than the level of significance that justifies the FedStrag to be statistically different from the FedAvg with better performance empirically. That concludes that the model trained with FedStrag has better performance than the baseline algorithm.

Table 4.3: P value of the trained model

	Fixed	Random
IID + 5s	1.38e-03	7.96e-05
ID + 10s	7.10e-16	5.54e-06
IID + 15s	2.46e-16	1.237e-11
non-IID + 5s	4.38e-05	6.35e-08
non-IID + 10s	2.81e-05	1.22e-13
non-IID + 15s	1.23e-01	6.71e-14

4.6 Conclusion

Training machine learning models in an IoT network is considerably harder due to stragglers and resource constraints. In this work, we conducted experiments that show that

stale updates have a negative impact on model training. At the same time, federated learning training with synchronous updates is inefficient due to client heterogeneity. To address the problem in this work, we propose FedStrag, a weighted averaging aggregation scheme that can efficiently deal with stale updates. For end-to-end model training with straggler nodes, we developed a time-bounded asynchronous federated learning. Considering resource-constrained devices as clients, we have adopted an online training scheme that trains the model on a subset of data sequentially over historically generated datasets.

We experimented with a prototype for the image classification task on two possible scenarios. The model is trained on both IID and non-IID datasets. The results suggest that the FedStrag scheme has better accuracy throughout all possible cases. The proposed scheme has converged faster than the baseline algorithm. However, FedStrag was unable to achieve significant accuracy on non-IID datasets. This is due to strong data bias at each client, which is a known limitation in federated learning. It addresses the key challenge of stragglers and optimizes aggregation at the server. This approach enables efficient machine learning even in small-scale networks. Overall, FedStrag improves model training in IoT networks with better generalization.

4.7 Summary

This chapter focuses on the issue of stragglers in federated learning within heterogeneous IoT networks. Stragglers are inevitable in an IoT network that needs to be addressed. This chapter assesses the impact of the stragglers and then proposes a solution to mitigate the impact on convergence. It demonstrates that stale model updates, often caused by hardware limitations or network issues, negatively impact model convergence. To address this, the chapter introduces FedStrag, a novel straggler-aware weighted averaging scheme. Instead of discarding stale updates, FedStrag penalizes them based on their staleness, giving greater priority to more recent updates. The proposed method is integrated into a time-bounded asynchronous federated learning framework and evaluated against the baseline FedAvg algorithm on the MNIST dataset, using both IID and non-IID data distributions and various straggler scenarios. The results indicate that FedStrag consistently outperforms FedAvg, leading to faster and more stable model convergence.

Chapter 5

Interpreting Federated Learning Convergence With Shapley Value

In this chapter, we explore the convergence behavior of federated learning by integrating with an explainable AI approach. Federated learning enables the training of a global model by aggregating multiple locally trained models, typically using the FedAvg algorithm. It is a simple yet effective averaging technique that combines the knowledge acquired at diverse locations into a global model. The chapter will look into why the FedAvg method converges during training and how individual features influence the convergence. To gain this understanding, we employ SHAP values, which allow us to quantify the impact of each feature throughout the training process. This chapter also presents our approach to integrating explainable AI techniques within the federated learning framework, thereby providing deeper insights into model behavior and decision-making.

5.1 Introduction

In recent years, there has been significant growth in digital data. With the advent of IoT and digital devices, data is being generated from multiple sources. Most of this data is generated from edge devices such as sensors, cameras, smart gadgets, etc., which can be accessed through the internet for processing [15, 102]. The traditional approach of sending all data to a cloud server not only leads to network congestion and bandwidth overuse but also raises concerns about privacy and security. Federated learning provides

an alternative approach as a decentralized way of model training that can create a global model without accessing raw data. Due to privacy-preserving and data security, federated learning has become a popular distributed training paradigm in recent years. It is an iterative learning process where multiple compute nodes collaborate to learn global knowledge from local data. FL has shown a path to train a global model from geographically separated distributed data without accessing it. Hence, it has emerged as a compelling area of interest for a broad spectrum of stakeholders, such as researchers aim to advance the field through novel methodologies, whereas business leaders seek to harness its potential for addressing practical challenges in data-sensitive environments. The paradigm has been implemented in various domains such as vehicular networks, hospitals, industrial networks, agriculture, etc.

The key idea of FL is to collectively learn a global knowledge with strong privacy preservation so that the data owner feels comfortable in training. Eventually, local learning is combined to create a global learning so that the global model can work equally well on the overall dataset. Hence, knowledge aggregation plays a critical role in the overall training. The vanilla federated learning performs a simple weighted averaging, FedAvg, over model parameters to create a global model. FedAvg performs layer-wise averaging to compute the aggregated value for every neuron. The global model has weighted average values of every neuron in each layer with the corresponding other models. The weights of the model are decided by the number of samples a particular model is trained on.

Training a machine learning model on the edge of the network is notoriously hard. Hence, most of the framework trains a model on a central server, and then the model is deployed on the edge nodes for inferences/predictions. The main limitation in the distributed training over an IoT network is resource limitations of the edge node. Fog computing paradigm addresses this by bringing significant resources near to the source, which works as a middle layer between the cloud and the edge node [105]. Kumar and Srirama have integrated federated learning into an edge-to-cloud continuum to train a global model on distributed fog nodes while preserving data privacy. The proposed framework, FIDEL [52], can efficiently train a model on continuously generated data from edge sources in real-time. In the framework, the edge layer is used as a data source, the fog layer is used as a distributed processing node, and the cloud layer aggregates locally trained models. To achieve continuous training on a resource-constrained IoT

network, FIDEL proposed online training that uses recently captured data for the current round of training. Overall, the framework provides a practical solution to implement federated learning on IoT networks. But how the global model converges with such a small sample of data is still unknown.

With numerous experiments, it is evident that a simple averaging method, FedAvg, works exceptionally well during aggregation. Though there are various challenges in the federated learning paradigm, such as heterogeneity, communication, non-IID data, data security, stragglers, etc. Thereafter, various research has been conducted to address these challenges, such as FedProx [58], FedNova [116], SCAFFOLD [45], FedStrag [51], Quantization method, Differential privacy, Holomorphic encryption, etc. Some of the studies are focused on the aggregation method to handle heterogeneity and non-IID. FedProx algorithm is designed to address data heterogeneity and system heterogeneity in distributed machine learning. FedNova solves the problem of objective inconsistency due to varying local update steps by normalizing updates, ensuring fairness despite client heterogeneity. FedStrag tackles straggler effects (slow or dropped clients) by adaptively selecting and aggregating client updates within deadlines. SCAFFOLD addresses client drift in non-IID settings by using control variates to correct the difference between client and server updates. Other techniques, such as quantization, differential privacy, or holomorphic computations, are used to optimize learning and secure the data. Though there have been various studies to improve the federated learning outcomes, aggregation is still the core component in overall training. Hence, it is crucial to know how/why a simple aggregation method accumulates the distributed learning and converges over time.

In a federated learning setup, the raw data is never exchanged, but trained parameters are for aggregation. The FedAvg algorithm is at the core of the aggregation that averages these updates to get a new global model. Though FedAvg works well on homogeneous data, it may not work equally well on heterogeneous data[59]. Hence, understanding why FedAvg works is a crucial question that can help in understanding FL paradigm to optimize it. There are multiple views on this question, such as statistical efficiency, optimization-based reasoning, communication, and theoretical convergence. For example, if each client has drawn data from the same distribution, then each client's local model is effectively a noisy estimator of the true global optimum. Averaging these noisy estimators reduces variance and brings the global model closer to the optimum.

Hence, FedAvg prevents overfitting to reach a global model. However, optimistically, FedAvg is also viewed as Stochastic Gradient Descent (SGD) at the client level. Here, each client performs local SGD steps on a common model while the server averages the results to reach global optima. However, the theoretical backbone of convergence lies in the assumption of a convex loss function, L-smoothness, IIDness, and bounded variance. Hence, averaging behaves like mini-batch SGD in comparison to centralized training. The theoretical analysis of the FL algorithm suggests that it has a convergence rate of $O(\frac{1}{T})$ for a strongly convex and smooth problem, where T is the number of SGDs. However, in practice, such strong assumptions do not hold. For example, data distribution at various clients is non-IID, and the loss function is non-convex to learn complex patterns. Additionally, deep learning models are a non-convex optimization problem. However, the FedAvg still converges over global optima. This brings curiosity about how/why federated learning works.

Existing work focuses on the statistical and theoretical convergence of the FedAvg method. However, it does not take the features' impact into account. Explainable AI (xAI) adds explanation to the AI model that provides reasoning for the decision. The explanation-based method helps to understand model's behaviors locally and globally. For example, it can reason out why a particular sample is classified as a true class or a false class. Similarly, an explainable model can provide global insight about feature importance. A local update on client data may have overfitted model that can prolong the convergence due to aggregation. In the same way, how a feature becomes prominent from local importance to global importance can be crucial information that can reveal how FedAvg works. A trend in feature influence on the local and global model should help to understand the model behavior. This analysis allows for monitoring concept and model drift from the local to the global model. Thus, integration of explainable AI with federated learning can be promising not only to understand FedAVG but also to make an efficient model that optimizes bias and fairness.

Every participating client contributes to the global model optimization. The FedAvg accommodates updates into the global model to enhance model performance. Hence, the contribution of every model is critical to know the convergence behaviors of FedAvg. Moreover, the contribution of each feature in global model creation that leads to convergence needs to be known. Hence, explainability-based federated learning will lead to resilient and robust training that overcomes client/feature bias. SHAP value is

the most popular explainability method for computing the importance of features. It leverages Shapley values from game theory to assign each feature an importance value for a particular prediction, quantifying the precise contribution of every feature. It can be applied globally to understand feature influence across the whole dataset and locally for individual predictions. The SHAP value is a consistent and mathematically grounded explanation for feature contributions, which is excellent for post hoc model interpretability. Hence, we integrated SHAP-based explainability to understand the behaviors of every feature on global and local model convergence. The main idea of the work is to unveil when the global model converges, and how the local model contributes to the convergence. How does a particular feature converge over time, and how does every client help in that convergence?

5.2 System architecture

Instead of relying on centralized training, we explore a decentralized data processing framework for predictive and inferential machine learning tasks. We consider an IoT network architecture designed for training a machine learning model on continuously generated data. The architecture consists of edge, fog, and cloud layers performing collaborative tasks to train a machine learning model. Here, federated learning is performed between the cloud node and fog nodes, while the edge devices supply data to the fog layer. Let us assume the network consists of F fog nodes, K edge nodes, and one central cloud server. Fog nodes in the network are typically resource-constrained devices with limited computational capacity, connected to the cloud over wireless channels. The system is running on continuous data with an online training scheme to overcome resource limitations. At the same time, the SHAP-based explanation is performed on the cloud node. Nodes in every layer execute independently in parallel. However, they are connected to the other layers via a communication channel.

5.2.1 Cloud layer

Nodes in the cloud layer have virtually infinite resources to compute aggregation, visualization, and explanation of the model. The main job of the cloud node is to perform aggregation of the locally trained model. The cloud node works as an orchestrator to manage overall training. It starts with creating a base model for the given task. It

initiates the training by sharing the base model with the connected fog nodes. The server seeks fine-tuned localized models from every connected fog node that can be aggregated to create a global model. Once the server receives locally trained models $(w_1, w_2, ..., w_f)$ from clients, it aggregates them to create a global model w_g. The aggregation is performed on weights of the local models as discussed in Chapter 2. Then the global model w_g is sent again to the fog node for further training. The training continues till the desired accuracy is achieved.

Since the model is a black box, we only focus on general metrics such as accuracy, F1 score, precision, recall, etc. Hence, even though we reach the desired accuracy, how the model w_g converges over time needs to be known. To address this, the server implements an explainability module on every federated round. The SHAP value is calculated for every locally trained model to check the importance of the feature. This determines the average feature impact of local models before aggregation, which can be seen as the feature impact of the model on the local device/data. Thereafter, the server computes feature importance on a global model to check how FedAvg summarizes the feature importance in the model. It traces out the impact of each feature over rounds on the global model to justify the feature convergence.

5.2.2 Fog layer

The fog layer is designed to perform localized analysis and processing on a smaller scale. Its primary responsibilities include providing computational training and the persistent storage of raw data generated by the edge layer. Storing data at the fog nodes serves two important purposes: (i) it helps ensure the privacy of user data by keeping it closer to the source, and (ii) it preserves the data for future analysis and reporting. The fog layer consists of IoT devices with comparatively greater computational power, such as Raspberry Pis, personal computers, private clouds, and clusters. This capability allows fog nodes to effectively process raw data before forwarding it. Positioned close to the data-generating devices, often within the user's premises, these fog nodes help reduce latency and alleviate network congestion.

Since nodes in the fog layer have limited resources, it is hard to train a machine learning model. Hence, we adopted the FIDEL approach to train the model on small data. The fog nodes continuously receive data from the edge node, which is saved in

the local storage. It selects recently captured data from the associated edge node and performs training on only that data. The selection of the data is done as follows.

$$D_i = \bigcup_{j=t_p}^{t_p+\tau} (X_j, y_j) \tag{5.1}$$

Here, τ is the number of samples a fog node can process. And t_p is the previous index initially set to 0 at round 1. In the further rounds, D_i is populated with new dataset and value of $t_p \in [0, min\{|D_f|, t_p+\tau\}]$ is incremented by $t_p+\tau$. Whereas (X_j, y_j) is i^{th} data frame.

At each round, the server asks for training every fog node fetches recent data D_i for training. Once training is completed, the fine-tuned model is shared with the server for aggregation. Even though the model is trained on local data D_i, the global model w_g is expected to learn hidden patterns in overall data.

5.2.3 Edge layer

The primary function of the edge layer is data generation. This layer comprises highly resource-constrained devices, such as sensors, GPS units, smart gadgets, cameras, and health monitoring devices. While these devices are capable of continuously producing raw data, they lack the necessary computational power and storage capacity to process or retain this data locally. Consequently, the edge layer primarily serves as a data acquisition source. Once data is generated, it is promptly transmitted to a nearby fog node. Devices in the edge layer maintain direct connections to corresponding fog layer nodes located within the user's premises. Through this, data collection is efficiently conducted at the edge, with seamless and continuous transfer to the fog node for training.

5.3 Proposed method

We consider a federated learning setup with F fog nodes as clients, each having private data continuously received from a connected edge node. As discussed in Section 5.2, data is stored at the fog node $D_f = \bigcup_{j=0}^{now} (X_j, y_j)$. The goal of the training is to collaboratively learn a global model w_g by minimizing the empirical risk over the union of local datasets without sharing raw data as follows.

$$\begin{aligned}\min_{w_g} J(w_g) &= \sum_{f=1}^{F} \frac{n_f}{|D_f|} J_f(w) \\ &= \sum_{f=1}^{F} \frac{n_f}{|D_f|} \mathbb{E}_{(x,y)\sim \mathcal{D}_f}[\ell(w; x, y)]\end{aligned} \tag{5.2}$$

where, $\ell(w; x, y)$ is the loss function, $|D_f| = \sum_{f=1}^{F} n_f$ is the total number of data points at a particular fog node F, $J_k(w)$ is the local empirical risk at client F and we want to optimize global objective $J(w_g)$ to find optimal parameter w_g.

Since fog nodes are resource-constrained, out of the total available data set D_f, we select currently generated data D_i of size n_f using Equation 5.1. Training is executed on fog nodes, and the aggregation operation is performed on the server as discussed in Algorithm 4. The server also computes SHAP values of local models to compute feature importance of the model. Since SHAP is computationally expensive hence it is being performed at the server. The SHAP values produce the feature importance of all local models, which provides local insights. Thereafter, it also computes SHAP values for the aggregated model. The analysis shows the trends and model drift at the granular level. Finally, it provides a convergence behavior of every feature on the global model. The results discuss how the impact of each feature grows or shrinks over time till the convergence. This reveals if convergence is driven by a few dominant features or if every feature is contributing uniformly. Finally, how features evolve with the training process over time. After aggregation and SHAP calculation, the server sends global parameters to fog nodes for further training. Server execution is described in Algorithm 4, the client module is shown in Algorithm 5, and SHAP value calculations are detailed in Algorithm 6.

As discussed in Algorithm 4, the cloud initializes the base model and shares it with the fog node. Every fog node trains the model and provides an updated model in parallel (steps 6-8). After training, server performs FedAvg on updated model to create next global model (step 9). Finally, it computes mean SHAP value of every model, including global model. The mean SHAP value provides feature importance on the overall data.

At the fog node ClientUpdate method performs model training as shown in Algorithm 5. It fetches recently captured data based on hardware capability. Thereafter, it

Algorithm 4 Federated learning execution at server

1: **Input:** Number of rounds T, number of clients F
2: **Output:** Feature importance on global and local models
3: Initialize global model weights w_0
4: **for** each round $t = 0, 1, \ldots, T-1$ **do**
5: Sends w_t to available fog node F
6: **for** each client $f \in F$ **in parallel do**
7: $w_{t+1}^f \leftarrow \textsc{ClientUpdate}(f, w_t)$
8: **end for**
9: Aggregation of locally trained models:
$w_{t+1} \leftarrow \sum_{f \in F} \frac{n_f}{|D_f|} w_{t+1}^f$,
where $|D_f| = \sum_{f=1}^{F} n_f$
10: Fetch standard data $\mathcal{D}_{\text{shap}}$ for SHAP values
11: **for** each client model w_{t+1}^f **do**
12: $\bar{\phi}_{t+1}^f = \text{ComputeSHAP}(w_{t+1}^f, \mathcal{D}_{\text{shap}})$
13: **end for**
14: Compute SHAP for aggregated model:
$\bar{\phi}_{t+1} = \text{ComputeSHAP}(w_{t+1}, \mathcal{D}_{\text{shap}})$
15: **end for**

divides data into chunks and performs model training (steps 4-8). Finally, the updated model is returned to the server for aggregation.

ComputeSHAP method performs feature importance of the dataset at the server as shown in Algorithm 6. The method implements the Shapley Additive exPlanations value of every data point to calculate feature importance over the data. First, it selects a representative sample from the dataset for the base prediction, then the rest of the dataset is used to compute the mean SHAP value (Step 3-4). It computes the SHAP value for every sample over every feature (steps 6-14). The SHAP produces feature importance by calculating the marginal contribution of the particular feature (step 11). Finally, the average SHAP value is calculated and returned to the server. The server stores mean SHAP as a feature importance over training rounds T.

5.4 Evaluations and results

This section discusses the experimental setup, results, and their analysis on various data. We trained two models on two different datasets to understand the convergence

Algorithm 5 ClientUpdate(f, w_g)

Input: Global weights w_g, local dataset $\mathcal{D}_f$, local epochs E, batch size B, learning rate η
Output: Fine tuned updated model w_g
Fetch latest data $D_i = \bigcup_{j=t_p}^{t_p+\tau} (X_j, y_j)$
Split $\mathcal{D}_i$ into mini-batches of size B
for each local epoch $e = 1$ to E **do**
    for each mini-batch b in $\mathcal{D}_i$ **do**
        $w_g \leftarrow w_g - \eta \nabla \ell(w_g; b)$
    end for
end for
return Updated weights w_g

behavior of the model.

5.4.1 Prototype and experiment setup

We implemented the proposed SHAP-based federated learning training on a desktop machine as a server with the configuration of an Intel Xeon 16-core CPU (3.7 GHz) with 32 GB RAM. Then, we created five containers as fog nodes, and their local data was mounted with them. Weight sharing is handled through the MQTT protocol, a lightweight, publish-subscribe messaging system designed for device-to-device communication. The communication is executed on two topics, namely 'train' and 'aggregate'. We employed containerization to execute code across different hardware platforms. Each device runs Docker containers, which offer OS-level virtualization that allows an application to execute in an isolated environment.

5.4.2 Dataset

We conducted the experiments on two publicly available datasets. The selection of the dataset is based on the number of features present that can be used for explanation. However, the method can be applied to any dataset. First, we experimented with the Iris data set [24]. The Iris dataset is a classic and widely used dataset in the fields of statistics and machine learning. The dataset consists of 150 samples from three species of the Iris flower (i) setosa, (ii) versicolor, and (iii) virginica, with 50 samples per class. Each sample has four numerical features as sepal length, sepal width, petal length, and

Algorithm 6 ComputeSHAP(w, D)

Input: Trained model f with parameter w, dataset D with N features and m instances
Output: Mean SHAP values $\bar{\phi} = [\bar{\phi}_1, \ldots, \bar{\phi}_n]$
Select a representative background data $B \subset D$ of size b
$X = D \setminus B$ is the remaining data used for SHAP computation with $|X| = m - b$
Initialize mean SHAP vector $\bar{\phi} \leftarrow [0, \ldots, 0] \in \mathbb{R}^N$
for each instance $x^{(j)} \in X$ **do**
    Initialize SHAP vector for the instance: $\phi^{(j)} \leftarrow [0, \ldots, 0]$
    for each feature $i \in \{1, 2, \ldots, N\}$ **do**
        for each subset $S \subseteq \{1, 2, \ldots, N\} \setminus \{i\}$ **do**
            Compute expected model output using background data B:

$$f_S(x_S^{(j)}) = \mathbb{E}_{z \in B}[f(z_{\bar{S}}, x_S^{(j)})]$$

$$f_{S\cup\{i\}}(x_{S\cup\{i\}}^{(j)}) = \mathbb{E}_{z \in B}[f(z_{\overline{S\cup\{i\}}}, x_{S\cup\{i\}}^{(j)})]$$

            Compute marginal contribution:

$$\Delta_i^{(j)}(S) = f_{S\cup\{i\}}(x_{S\cup\{i\}}^{(j)}) - f_S(x_S^{(j)})$$

            Compute Shapley weight:

$$w_S = \frac{|S|!\,(N - |S| - 1)!}{N!}$$

            Accumulate:

$$\phi_i^{(j)} \leftarrow \phi_i^{(j)} + w_S \cdot \Delta_i^{(j)}(S)$$

        end for
    end for
    Update mean SHAP: $\bar{\phi} \leftarrow \bar{\phi} + \phi^{(j)}$
end for
Compute final average: $\bar{\phi} \leftarrow \frac{1}{|X|}\bar{\phi}$
Return: Mean SHAP values $\bar{\phi} = [\bar{\phi}_1, \ldots, \bar{\phi}_N]$

petal width (all in centimeters). 150 samples are divided into five clients for federated training. Since the dataset is limited in this case, out of 150, 50 stratified samples were chosen for test data for the server. Further, to simulate online training, only six samples are used at a time for training. Representative samples ensure that the test data includes all class labels, while online training guarantees that training and testing occur on largely disjoint datasets

Then we experimented with Predictive maintenance (PM) data published on UCI Machine Learning Repository developed by Stephan Matzka [75]. The AI4I 2020 Predictive Maintenance Dataset is a synthetic dataset designed to simulate real-world conditions for predictive maintenance in industrial settings. The dataset contains 10,000 instances with 10 features. The target variable is binary, which indicates failure or non-failure. The dataset is also imbalanced. So we used SMOTE for oversampling of the minority class. We applied preprocessing and removed non-predictive features, such as UID, Product ID, and Type. Out of the two predicted variables, we chose Target variable as the label. This makes the five predictive features as air temperature, process temperature, rotational speed, torque, and tool wear. This dataset is widely used for benchmarking classification models and testing explainability techniques in predictive maintenance scenarios. After sampling, we divided the 80:20 dataset into train and test. The train data is distributed among fog nodes, and the test data is used at the server for SHAP calculations. Each fog node receives $approx3000$ samples, which are used for local training. Here we use 60 samples per round to ensure online training.

5.4.3 Model

Both datasets vary in shape and size, so we built two custom multi-layer neural networks for training. For the Iris dataset, the model has 3 hidden layers with (16, 8, 4) neurons, followed by a ReLU activation layer in each layer. For the Predictive maintenance dataset, the model has 3 hidden layers with the first two layers having (16, 32) neurons, followed by a ReLU and a Dropout layer with p = (0.25, 0.1) respectively. The third layer is a dense layer with 8 neurons with ReLU activation. The input-output layer has (4, 3) neurons in Iris and (5, 2) in the Predictive maintenance dataset, which is decided by the number of features and classes. The total number of parameters in both models is 267 and 922, respectively. We kept both models simple and multi-layered to understand the convergence for deep neural networks. We trained both models with 100

rounds and analyzed how each feature converges as the overall accuracy converges. We implemented circular data extraction to simulate continuous data generation. Hence, once the data is exhausted on the fog node, it starts fetching it from the initial index. For this work, we are focused on knowing the convergence of the model, so we are not performing hyperparameter tuning.

5.4.4 Result and analysis

The overall training on the Iris dataset is shown in Fig 5.1. As training rounds increased, the accuracy of the model improved. The model started stabilizing after 40 rounds. The training trajectory clearly suggests that even though local models deviate from training, the global model (blue dotted line) cancels out noise and optimizes the training.

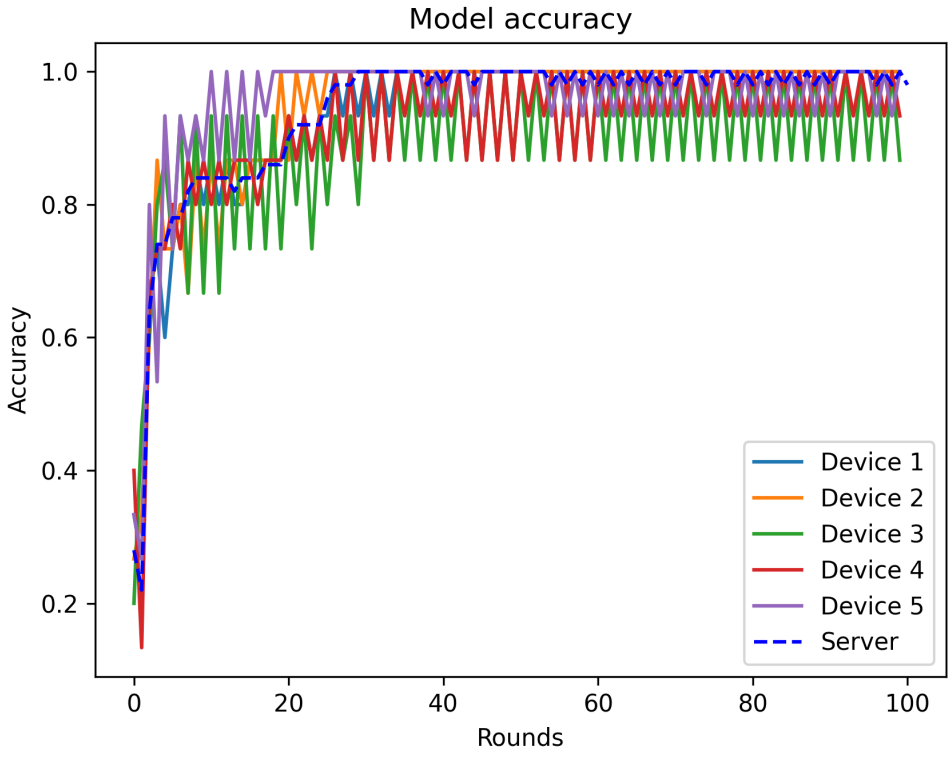


Figure 5.1: Model performance for Iris data

Although the global model started converging after 40^{th} rounds, we are interested in knowing how each feature converges over rounds and whether there is any trend in

the feature's impact. Fig 5.2 shows impact trajectory of every feature over training rounds.

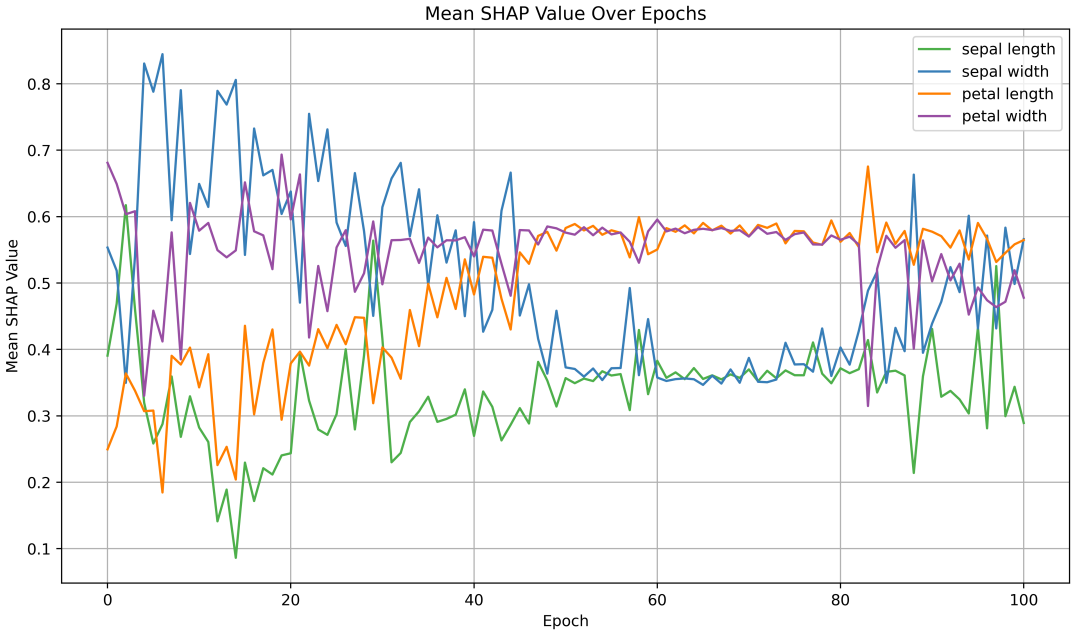


Figure 5.2: Feature trends on aggregated model over rounds on Iris data

Fig 5.2 clearly suggests that there is a correlation between feature impact and global model convergence. In the initial rounds of the training, sepal width and petal width contributed more while petal length and sepal length were marginally contributing. However, once the model started converging around 40^{th} round, SHAP values of the features started converging. Although there are drifts in the later rounds because of a local update (discussed later). This behavior is also attributed to the use of online training, with only 6 samples per round. Still, it stabilized along with accuracy in Fig 5.1. We further plotted the SHAP values of every feature of the global model during training in Fig 5.3. The SHAP bar graph of each feature shows that the impact of a feature grows or shrinks over time as the model matures. For example, sepal length started with a higher value and it shrank sooner (red highlight), later it grew consistently, but petal length started with a lower value and increased gradually. So it confirms that the global model converges based on the feature convergence.

Although SHAP values converge with the global model, we wanted to understand how the local model performs. As discussed earlier, one of the key reasons FedAvg

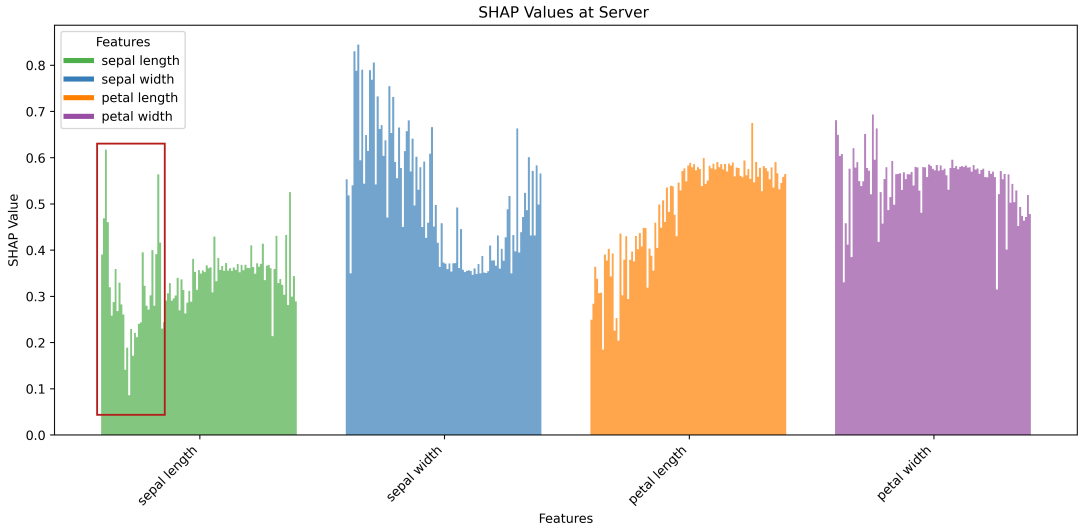


Figure 5.3: Impact of every feature on aggregated model during training on Iris data

works is that it starts with the same global model and every client fine-tunes it. This works as a regularizer that minimizes variance and drift in convergence. Accuracy result also confirms it in Fig 5.1. However, we have plotted SHAP values of the local model in Fig 5.4 to Fig 5.8.

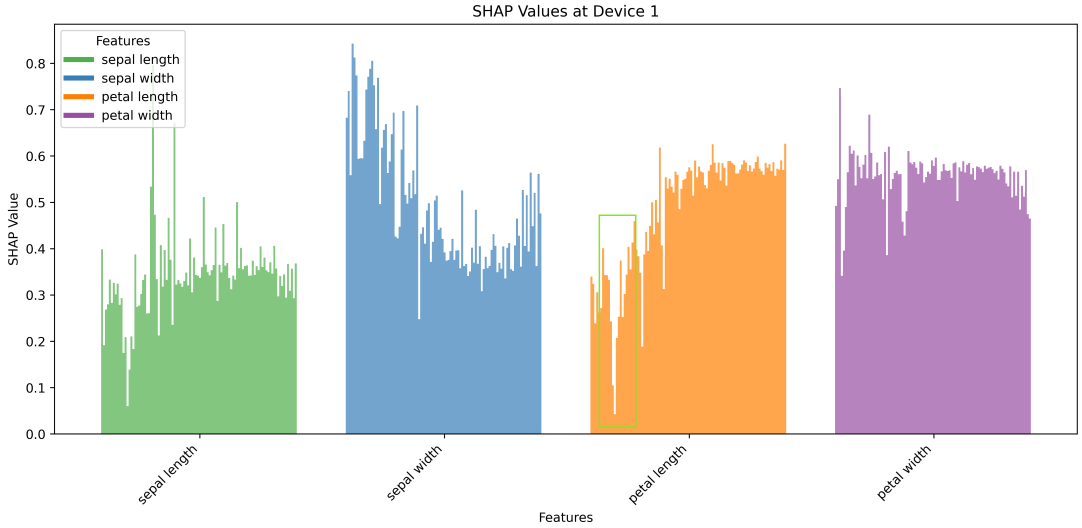


Figure 5.4: Impact of every feature on device 1 during training on Iris data

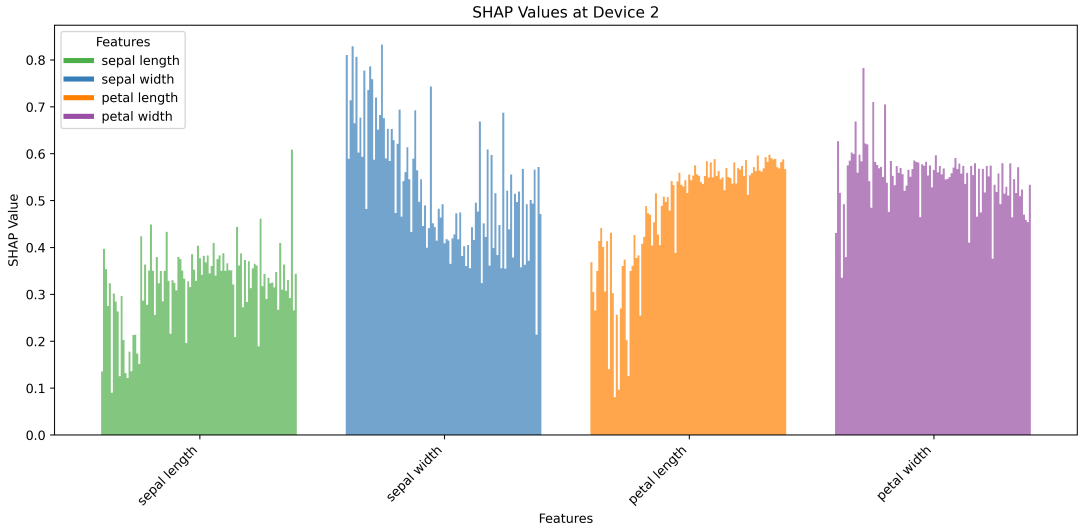


Figure 5.5: Impact of every feature on device 2 during training on Iris data

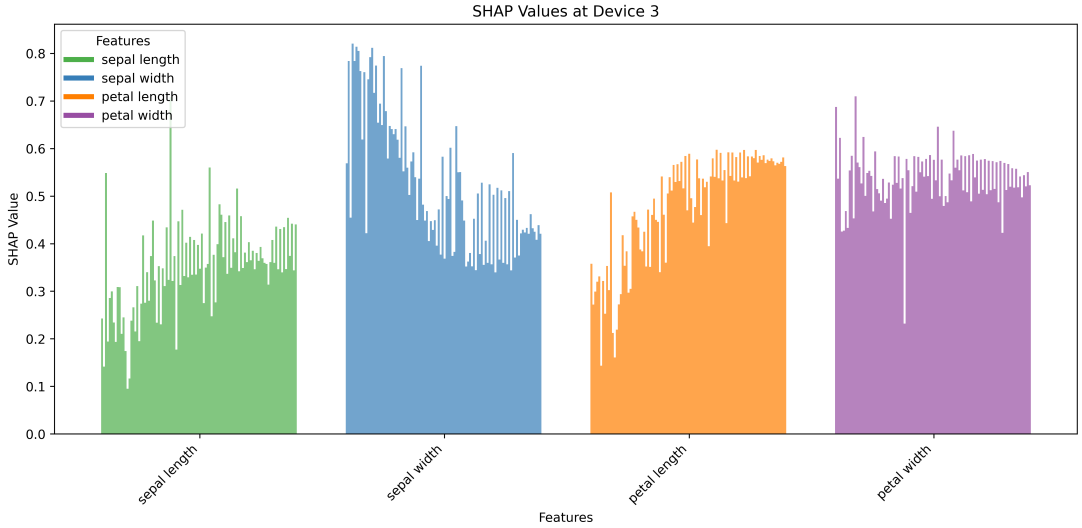


Figure 5.6: Impact of every feature on device 3 during training on Iris data

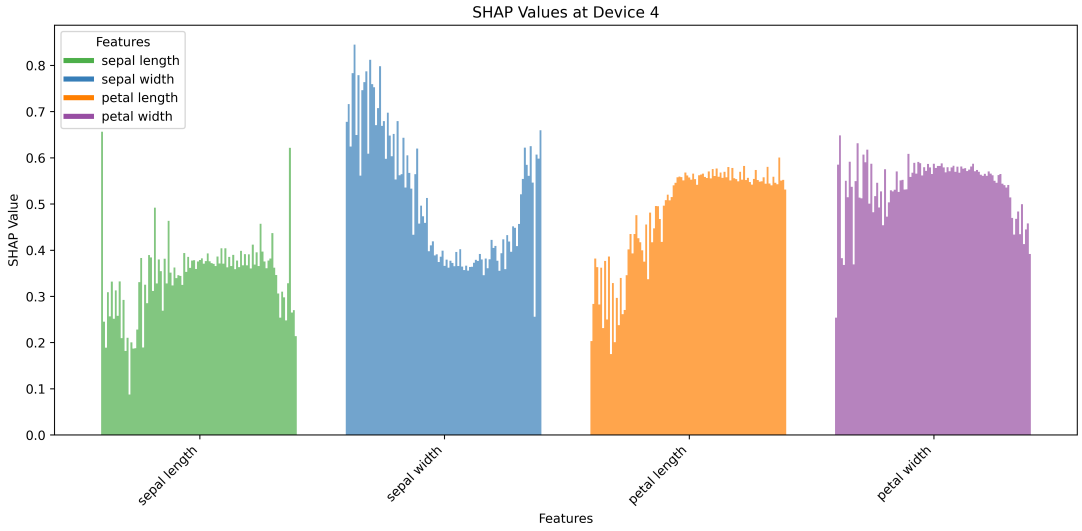


Figure 5.7: Impact of every feature on device 4 during training on Iris data

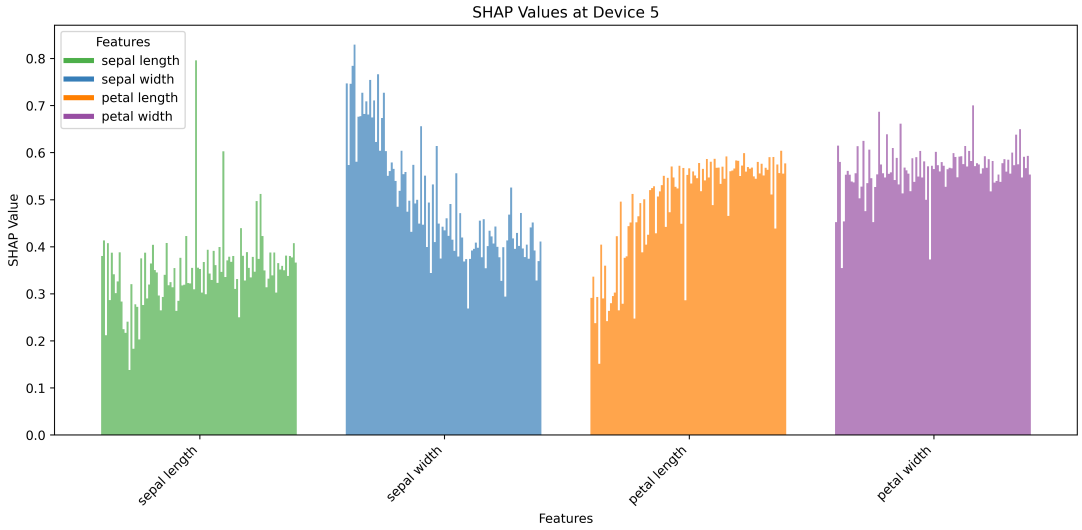


Figure 5.8: Impact of every feature on device 5 during training on Iris data

Fig 5.4 to Fig 5.8 shows feature impact on local models. It mostly follows global trends because the dataset is IID. However, some local updates tried to influence the global model, but other updates normalized it to the averaged value. For example, there is a sharp decline in petal length on device 1 (green highlight) in comparison to the rest of the devices. However, other devices elevate petal length. Hence, the global

model continued to grow.

As discussed earlier, Fig 5.2 suggests that the model converges, but it starts fluctuating after 80^{th} rounds. This behavior is attributed to a small number of samples (6 at a time) used during training. This can be verified by the bar plot of global SHAP, which suggests that sepal width has started contributing more in later rounds. In this case, Device 4 has contributed more towards the sepal width, which creates drift in the global convergence shown in Fig 5.7. Hence, more data can be used to train the model that will stabilize the training. To test it, we rerun the experiments with 30 samples training at a time. In this case, it learnt quickly and converged properly as shown in Fig 5.9. This justifies that larger samples at the fog node have a stabilizing effect and after convergence.

Figure 5.9: Feature trends on aggregated model over rounds with 30 samples on Iris data

Finally, we experimented with the Predictive maintenance dataset shown in Fig 5.10. Though other features have marginal contributions, Torque has a significantly higher impact, even though it started with a lower impact.

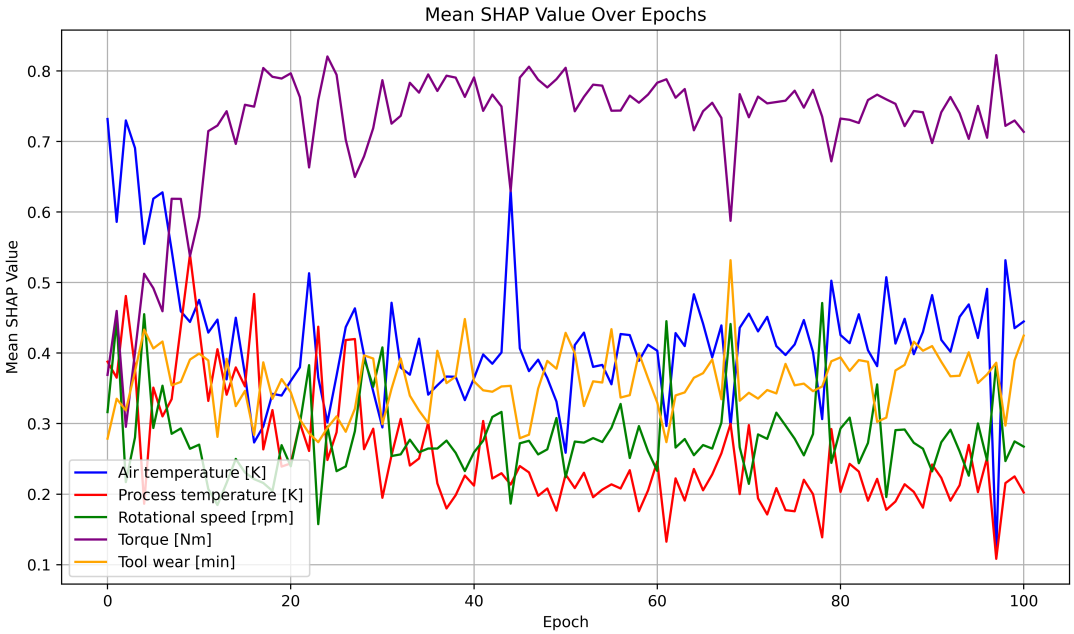


Figure 5.10: Feature trends on aggregated model over rounds on PM data

Similarly, Air temperature started with a higher impact, but it converges to a lower value during the training, also shown in bar graph 5.11. The overall Torque is the most influential feature for the trained model. We have also traced feature impact on the local model, which has a similar correlation as we found in the Iris dataset shown in Fig 5.12 to Fig 5.16. Some features (Air temperature, Process temperature) start with a larger impact, but they end up converging to their actual impact value. The result also suggests that the global model works as a stabilizer to optimize the overall training. For example, Process temperature in almost all devices has varying values, but the global model values are more stable and converging. Finally, the result demonstrates a clear understanding not only of how a feature's importance evolves during training, but also reveals the value of a particular feature, which can be used to evaluate the black box model. This enhances confidence and trust in the model and can help mitigate model bias.

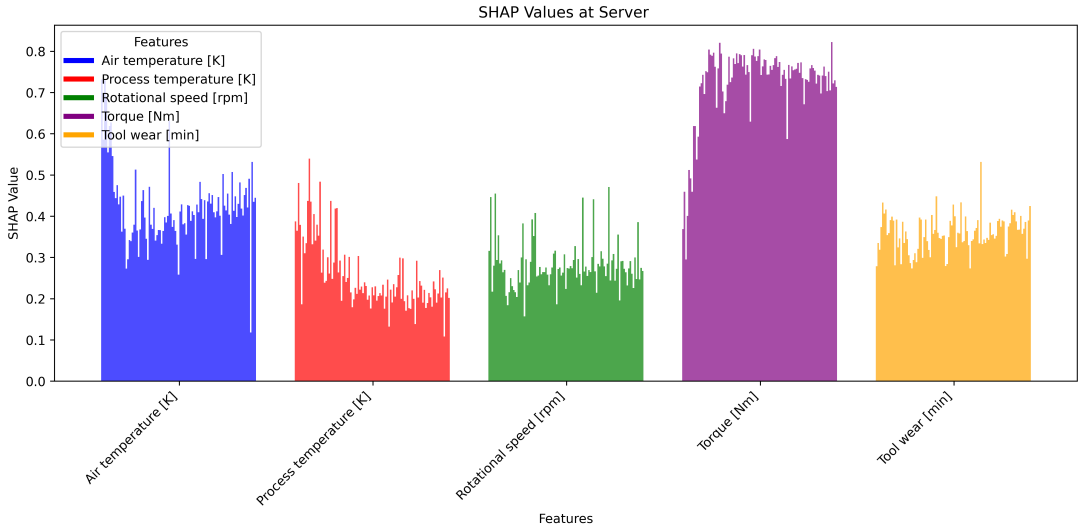


Figure 5.11: Impact of every feature on aggregated model during training on PM data

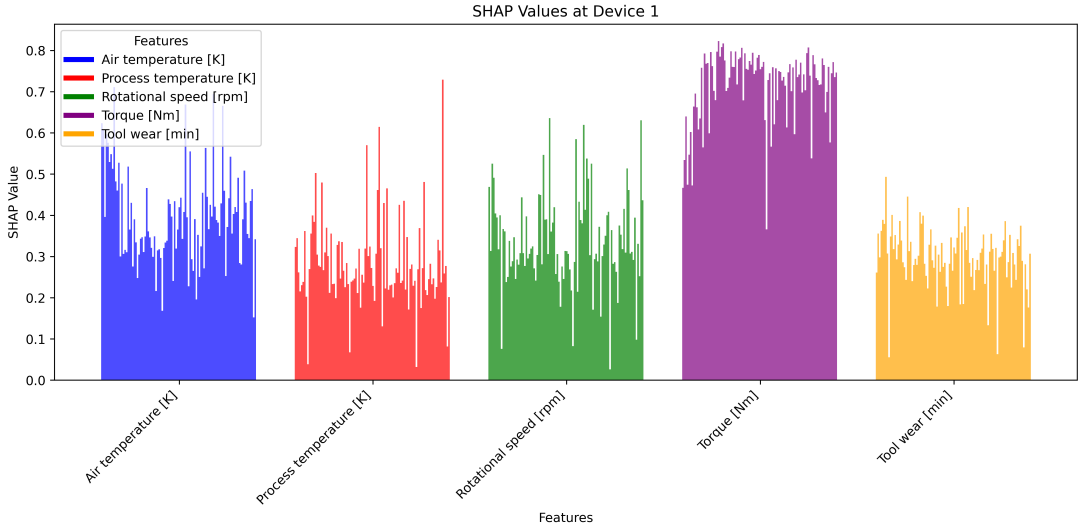


Figure 5.12: Impact of every feature on device 1 during training on PM data

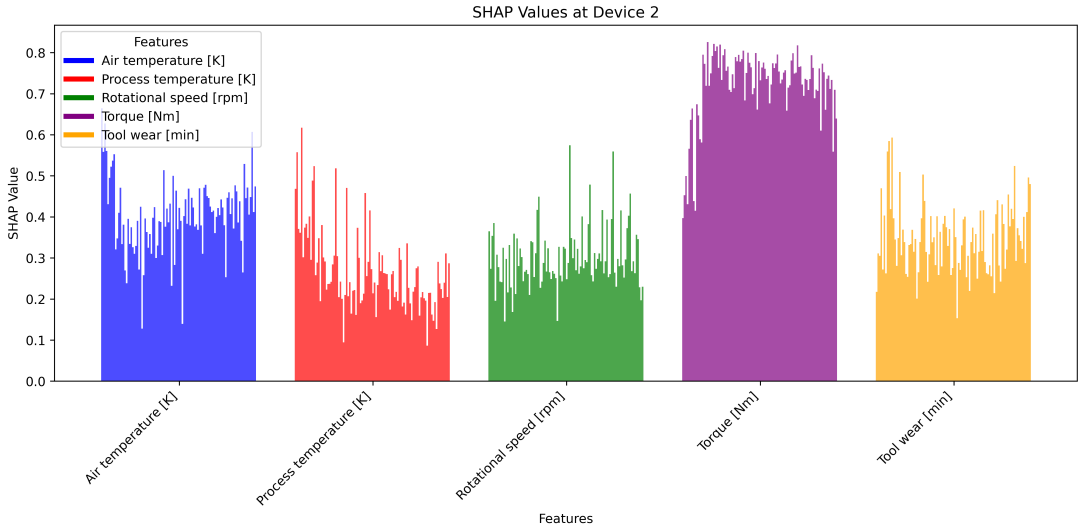


Figure 5.13: Impact of every feature on device 2 during training on PM data

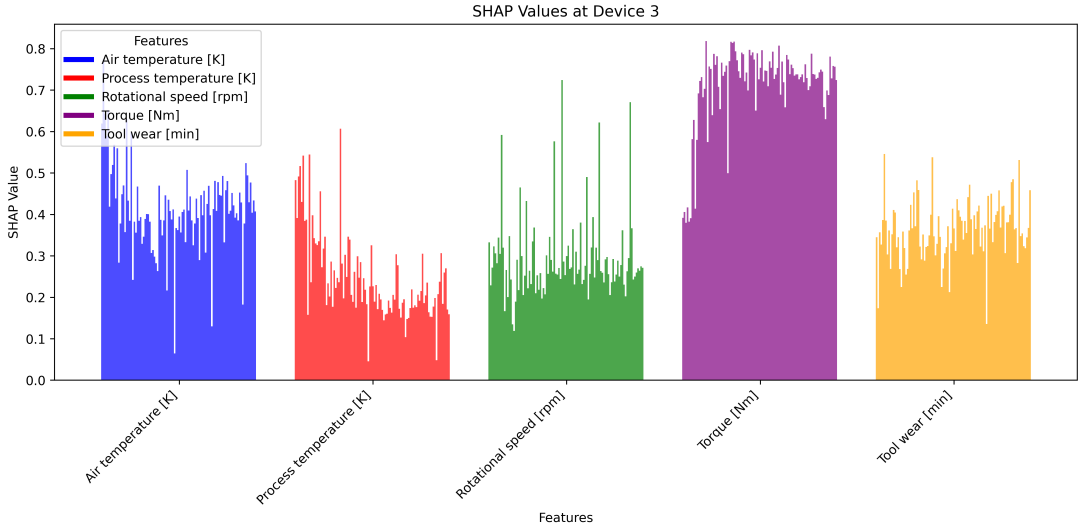


Figure 5.14: Impact of every feature on device 3 during training on PM data

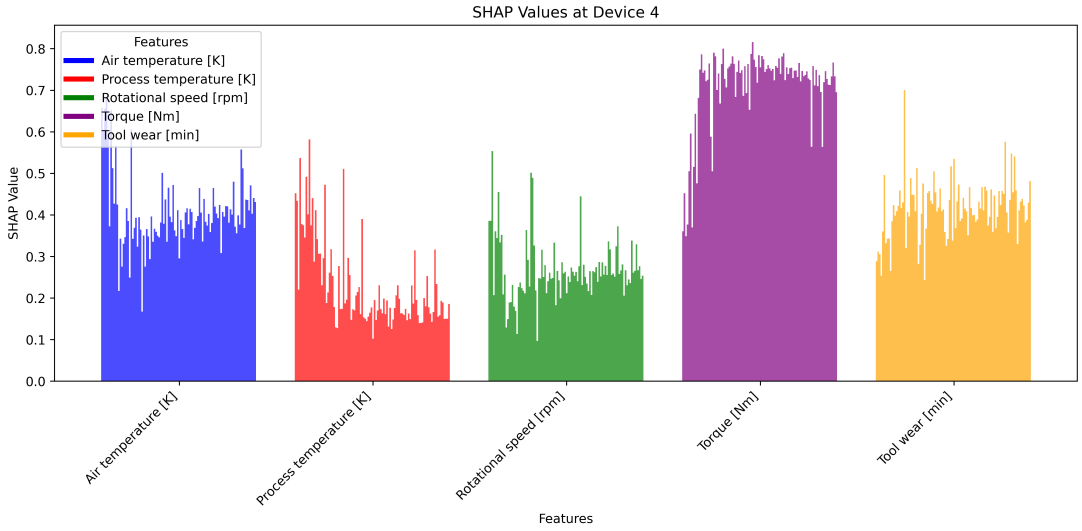


Figure 5.15: Impact of every feature on device 4 during training on PM data

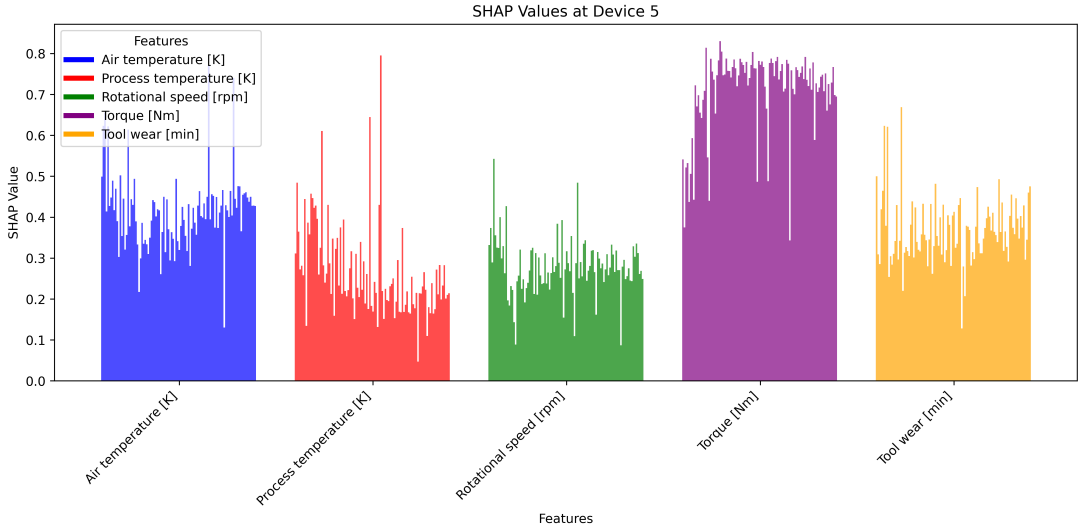


Figure 5.16: Impact of every feature on device 5 during training on PM data

5.5 Conclusion

In this work, we discuss federated learning convergence behaviors. The chapter explained various aspects of why FedAvg converges for a better understanding of the convergence of federated learning. We integrated an explanation module into federated learning. With SHAP-based explanation of each feature, it provides insights about how a particular feature converges. It reveals a correlation between model convergence and feature importance. The results show that the global model stabilizes once feature importance converges. We found a pattern in the impact of features on model creation. It influences each feature at the device/server level. The work is focused on the convergence of the FedAvg with feature-level explanation. However, the result of the SHAP value can be utilized to create more stable and fair aggregation.

5.6 Summary

This chapter investigates the convergence of federated learning by introducing an explainable AI approach using SHAP values. The core idea is to understand how the FedAvg algorithm converges by tracking the influence of individual features throughout the training process. The chapter outlines an architecture where an explanation module is implemented on the server that calculates SHAP values for both local client models and the aggregated global model. This allows for real-time monitoring of feature importance evolution, revealing a direct correlation between feature impact and model convergence. Experimental results on two different datasets demonstrate that the global model's stability is achieved once the feature impact converges.

Chapter 6

Conclusions and Future Work

This chapter concludes the thesis and summarizes the research work carried out during the PhD. It outlines the key contributions, highlights the limitations, and discusses potential directions for future research.

6.1 Conclusions

The thesis addresses key challenges of training a machine learning model on the edge-to-cloud continuum. Unlike the conventional training paradigm, it adopted a distributed data processing over multiple clients using federated learning. The training not only optimizes resources through distributed computing but also ensures data privacy. However, edge and fog nodes lack sufficient resources for model training. Additionally, in an edge-to-cloud continuum, the continuously growing data makes it difficult to train models at the edge of the network. An IoT network consists of heterogeneous devices connected through a wireless medium. In such a network, stragglers are inevitable. Therefore, understanding how to train a model under these conditions and analyzing their impact on model convergence is essential. At the same time, how federated learning converges needs an explanation. To address these, the goal of the thesis was threefold: to address the four research questions. The first is to investigate whether machine learning training is possible on resource-constrained edge/fog nodes. The second is focused on building a framework for failure-prone and heterogeneous IoT networks. Third, to find impact of stragglers on model convergence. And fourth, analyzes how federated learning converges over time. To achieve the three research goals, we have

addressed the research questions as outlined below.

RQ - 1: How to train a privacy-preserving decentralized model on distributed nodes on resource-constrained heterogeneous edge/fog nodes? Is it possible to train the neural network on a continuously growing dataset with resource limitations?

To answer RQ 1, we created a fog-enabled federated learning architecture for model training. It is a 3-layered architecture that implements federated learning on continuous data. The edge node produces data, fog nodes perform model training, and the cloud works as an aggregator. The implementation of federated learning ensures privacy of the data because it guarantees data localization at the nearest fog node. The architecture provides a solution for model training on distributed, resource-constrained devices. The key challenges in the training are the resource constraints and continuous inflow of data from the edge node. Our literature review in Chapter 2 indicates that there has been no attention on training a model with continuous data. So we proposed an online training scheme to address resource constraints on a continuously growing dataset. As per architecture, the edge node generates continuous data and shares with fog node. The fog and cloud nodes perform federated learning collaboratively. The main contribution of the work is that it provides a realistic solution for model training on a continuously growing dataset. The online scheme trains the model on recently generated data points. At the same time, the number of samples it uses depends on the hardware capabilities of the fog node. We implemented the architecture and simulated an IoT network to showcase the capabilities of the proposed work. The simulation result showed that it can learn a global model on continuous data with online training.

RQ - 2: What is the feasibility of developing a framework for distributed model training in a failure-prone and heterogeneous IoT network for real-world applications? What challenges are involved in its implementation, and can such a framework achieve convergence comparable to that of centralized training?

To address RQ 2, we built a fog-integrated federated learning framework, FIDEL, for neural network training on a heterogeneous IoT network. The major challenges of

model training in an IoT network are resource constraints, stragglers, hardware heterogeneity, and unreliable networks. The framework should address all these issues while preserving the privacy of the data and performing equally as centralized training. Though there have been multiple efforts to build frameworks, none of the frameworks achieves all of these as discussed in Chapter 2. To build FIDEL, we extended our previous architecture that can perform federated learning on resource-constrained devices. The cloud node is used for aggregation, the fog node is used for online training, and the edge node generates data. It supports distributed training on heterogeneous devices and is inherently built to protect data sharing to the cloud. The framework is implemented using Docker containers to handle hardware heterogeneity, thus FIDEL can be executed on any compute node that supports Docker. We created a prototype implementation of the framework to test the capabilities on a real-world IIoT dataset. The result suggests that it achieves equal results to its counterpart's centralized training.

RQ - 3: To what extent do stragglers impact model training in distributed learning environments? Furthermore, can the contributions from these delayed or underperforming nodes be effectively utilized to improve the model's generalization performance?

To answer RQ 3, first, we analyzed the impact of the stragglers on model convergence. The result suggested that it has a negative influence on model training. We found that the FedAvg method aggregates every local model equally, irrespective of staleness of the update. Existing work discussed in Chapter 2 mainly focused on partial aggregation by sampling updates. This approach may work with large clients but is infeasible for smaller networks. To address this, we proposed a stragglers-aware aggregation method for federated learning. The work finds out the stale update and penalizes it during aggregation. Hence, it prioritizes the latest updates over stale ones. FedStrag optimizes both stale updates and latest updates rather than eliminating based on staleness. We tested the FedStrag method on two possible stragglers scenarios, fixed and random stragglers, with both IID and non-IID datasets. The results show that FedStrag outperforms the baseline on all possible scenarios.

RQ - 4: How significantly does each individual feature contribute to the construction of the global model during federated learning? Furthermore, what patterns emerge in feature importance over the course of training rounds?

To answer RQ 4, we have conducted extensive experiments to understand the aggregation method FedAvg's convergence behavior. We have seen that federated learning works well for training a neural network. It trained multiple local models and aggregated them into a global model. But, how the FedAvg method works needs to be known. The state-of-the-art works mainly focus on providing theatrical guarantees and statistical convergence analysis. However, this analysis is based on multiple assumptions that may not hold in practical implementation. Hence, to know how federated learning converges, we added explainable AI into federated learning. The proposed method uses SHAP value explanation to compute feature importance during training. It computes feature influence on both global and local models to trace how it converges during training. The results show that the model convergence and feature impact convergence are correlated. We tried understanding how feature impact influences the model drift that can lead to divergence or prolonged convergence. The analysis provides feature-wise granular explanation of model convergence.

This thesis presented a comprehensive feasibility study of federated learning on resource-constrained IoT networks, addressing the key challenges in implementing distributed machine learning across the compute continuum. The proposed FIDEL framework offers a practical solution for building AI models directly on IoT devices, accommodating the inherent heterogeneity of such environments. Its asynchronous implementation effectively mitigates straggler issues, ensuring seamless system operation. Furthermore, the proposed FedStrag introduces a novel aggregation scheme that optimizes training performance even in failure-prone networks. Experimental evaluations demonstrate the effectiveness of both FIDEL and FedStrag in enhancing the robustness and efficiency of federated learning systems. The convergence behaviour of FIDEL was further examined to understand the feature-level impact on model training. Integrating the xAI (SHAP) approach into FIDEL makes the system more transparent and trustworthy, ensuring the machine learning is fair and free from bias.

Overall, the thesis contributes towards enabling next-generation AI models for diverse applications in distributed environments. The developed prototype paves the way

for broader adoption of federated learning in real-world IoT deployments.

6.2 Future research directions

The experiments in this thesis were conducted in a controlled real-world setting. However, it can be extended to a large-scale IoT setup, which may enhance the system's performance. The prototype implementation was executed on a Raspberry Pi as a fog node, but the system can produce better results with higher-end devices such as a Nano Jetson. The methodology in the proposed framework has a well-defined task for every node/layer. To this end, there is a scope to provide a hybrid solution of model parallelization or data offloading. Additionally, resource-aware computing and scheduling can also be proposed as future work. The further enhancement in the system may require on-device processing, event-driven processing with tiny ML implementation of novel frameworks.

The thesis worked in the direction of proposing a practical solution for model training in an IoT network. The proposed framework assumes that the participants are honest and transparent. Hence, it did not consider any security aspect of the system. This is a major limitation of the work, which will be the direction of our future research. It should cover data security, model security, adversarial attack, eavesdropping, and malicious client attack. We also aim to incentivize the federated learning framework for better adoption in the real world. This will include proposing an incentives model for data owners and device providers.

In our experiments, we have noticed that the proposed framework finds it relatively hard to train on non-IID data. Although this is a known limitation in federated learning. But for real-world deployment, the limitations need to be overcome to solve practical problems. We will look into this to find better training and aggregation strategies to mitigate non-IID data impacts. Chapter 5 had a detailed discussion about convergence analysis of federated learning using SHAP. The work sets the basis for explainable federated learning for trustworthy training. We will work in the direction to achieve interpretable and reliable federated learning for better adoption of the system. We have demonstrated the capabilities of the framework for the limited use cases. In the future, we will leverage the framework to provide a system for various applications such as smart healthcare and smart agriculture.

Large language models (LLMs) have the potential to build next-generation real-world applications. Future research will explore the integration of LLMs into federated learning across the edge–cloud continuum, with the aim of enabling collaborative model adaptation without compromising data privacy. We will explore low-rank parameter fine-tuning techniques (LoRA/QLoRA) suited for resource-constrained devices, as well as hierarchical orchestration strategies that coordinate training across device, edge, and cloud.

Energy consumption remains a critical concern that needs significant attention. Although the current system performs effectively, model training continues to demand larger energy resources. The escalating demand for computational power poses a challenge to sustainability. We planned to work in the direction of efficient and sustainable computing by adopting emerging paradigms such as neuromorphic computing. This approach has the potential to enable the development of human-like intelligence at the edge of the network, while significantly minimizing energy consumption.

References

[1] **TensorFlow Federated: Machine Learning on Decentralized Data**. https://www.tensorflow.org/federated, 2024. Accessed: 2025-06-13. (11, 48)

[2] **OpenFL: an open framework for federated learning**. https://openfl.io/, 2025. Accessed: 2025-06-17. (11, 49)

[3] Abebe Abeshu and Naveen Chilamkurti. **Deep Learning: The Frontier for Distributed Attack Detection in Fog-to-Things Computing**. *IEEE Communications Magazine*, **56**(2):169–175, 2018. (38)

[4] Haftay Gebreslasie Abreha, Mohammad Hayajneh, and Mohamed Adel Serhani. **Federated Learning in Edge Computing: A Systematic Survey**. *Sensors*, **22**(2), 2022. (38)

[5] Mohammed Al-Khafajiy, Thar Baker, Atif Waraich, Omar Alfandi, and Aseel Hussien. **Enabling high performance fog computing through fog-2-fog coordination model**. In *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, 2019. (37)

[6] Marica Amadeo, Claudia Campolo, Antonella Molinaro, Giuseppe Ruggeri, and Gurtaj Singh. **Mitigating the Communication Straggler Effect in Federated Learning via Named Data Networking**. *IEEE Communications Magazine*, **62**(11):92–98, 2024. (46)

[7] IoT Analytics. **Number of connected IoT devices growing rapidly**. https://iot-analytics.com/number-connected-iot-devices/, 2024. Accessed: 2025-07-29. (1)

[8] Kevin Ashton et al. **That 'internet of things' thing**. *RFID journal*, **22**(7):97–114, 2009. (25)

[9] Tomisin Awosika, Raj Mani Shukla, and Bernardi Pranggono. **Transparency and Privacy: The Role of Explainable AI and Federated Learning in Financial Fraud Detection**. *IEEE Access*, **12**:64551–64560, 2024. (52)

[10] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. **Flower: A Friendly Federated Learning Research Framework**. *CoRR*, **abs/2007.14390**, 2020. (11, 48)

[11] Dost Muhammad Saqib Bhatti, Mazhar Ali, Junyong Yoon, and Bong Jun Choi. **Efficient Collaborative Learning in the Industrial IoT Using Federated Learning and Adaptive Weighting Based on Shapley Values**. *Sensors*, **25**(3), 2025. (52)

[12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. **Fog computing and its role in the internet of things**. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery. (2, 36)

[13] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. **TiFL: A Tier-Based Federated Learning System**. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, page 125–136, New York, NY, USA, 2020. Association for Computing Machinery. (46)

[14] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. **FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers**. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. (12, 46)

[15] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. *Internet of Things (IoT) and New Computing Paradigms*, chapter 1, pages 1–23. John Wiley & Sons, Ltd, 2019. (2, 25, 36, 117)

[16] Daoyuan Chen, Dawei Gao, Yuexiang Xie, Xuchen Pan, Zitao Li, Yaliang Li, Bolin Ding, and Jingren Zhou. **FS-REAL: Towards Real-World Cross-Device Federated Learning**. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 3829–3841. Association for Computing Machinery, 2023. (45)

[17] Junbin Chen, Jipu Li, Ruyi Huang, Ke Yue, Zhuyun Chen, and Weihua Li. **Federated Transfer Learning for Bearing Fault Diagnosis With Discrepancy-Based Weighted Federated Averaging**. *IEEE Transactions on Instrumentation and Measurement*, **71**:1–11, 2022. (40)

[18] Ming Chen, Bingcheng Mao, and Tianyi Ma. **FedSA: A staleness-aware asynchronous Federated Learning algorithm with non-IID data**. *Future Generation Computer Systems*, **120**:1–12, 2021. (46)

[19] Zheyi Chen, Weixian Liao, Kun Hua, Chao Lu, and Wei Yu. **Towards asynchronous federated learning for heterogeneous edge-powered internet of things**. *Digital Communications and Networks*, **7**(3):317–326, 2021. (43)

[20] OpenFog Consortium. **OpenFog Reference Architecture for Fog Computing, Technical Report**, February, 2017. (36)

[21] Qiang Dai, Tongjiang Yan, and Pengcheng Ren. **FedCSR: A new cluster sampling based on rotation mechanism in horizontal federated learning**. *Computer Communications*, **210**:312–320, 2023. (12, 46)

[22] Yongheng Deng, Feng Lyu, Tengxi Xia, Yuezhi Zhou, Yaoxue Zhang, Ju Ren, and Yuanyuan Yang. **A Communication-Efficient Hierarchical Federated Learning Framework via Shaping Data Distribution at Edge**. *IEEE/ACM Transactions on Networking*, **32**(3):2600–2615, 2024. (42)

[23] Shaohan Feng, Dusit Niyato, Ping Wang, Dong In Kim, and Ying-Chang Liang. **Joint Service Pricing and Cooperative Relay Communication for Federated Learning**. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 815–820, 2019. (41)

[24] R. A. Fisher. **Iris**. UCI Machine Learning Repository, 1936. DOI: https://doi.org/10.24432/C56C76. (126)

[25] Hongchang Gao, Junyi Li, and Heng Huang. **On the Convergence of Local Stochastic Compositional Gradient Descent with Momentum**. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, **162** of *Proceedings of Machine Learning Research*, pages 7017–7035. PMLR, 17–23 Jul 2022. (51)

[26] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. **Internet of Things (IoT): A vision, architectural elements, and future directions**. *Future Generation Computer Systems*, **29**(7):1645–1660, 2013. (1)

[27] Farzin Haddadpour and Mehrdad Mahdavi. **On the Convergence of Local Descent Methods in Federated Learning**. *CoRR*, **abs/1910.14425**, 2019. (14)

[28] 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**, 2019. (39)

[29] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. **A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures**. *IEEE Access*, **7**:82721–82743, 2019. (27)

[30] Abhishek Hazra, Mainak Adhikari, Tarachand Amgoth, and Satish Narayana Srirama. **Joint computation offloading and scheduling optimization of IoT applications in fog networks**. *IEEE Transactions on Network Science and Engineering*, **7**(4):3266–3278, 2020. (37)

[31] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. **FedML: A Research Library and Benchmark for Federated Machine Learning**. *CoRR*, **abs/2007.13518**, 2020. (49)

[32] István Hegedűs, Gábor Danner, and Márk Jelasity. **Decentralized learning works: An empirical comparison of gossip learning and federated learning**. *Journal of Parallel and Distributed Computing*, **148**:109–124, 2021. (41)

[33] Mohammadsadeq Garshasbi Herabad. **Communication-efficient semi-synchronous hierarchical federated learning with balanced training in heterogeneous IoT edge environments**. *Internet of Things*, **21**:100642, 2023. (45)

[34] Chung-Hsuan Hu, Zheng Chen, and Erik G. Larsson. **Scheduling and Aggregation Design for Asynchronous Federated Learning over Wireless Networks**. *IEEE Journal on Selected Areas in Communications*, pages 1–1, 2023. (42)

[35] Chao Huang, Jianwei Huang, and Xin Liu. **Cross-Silo Federated Learning: Challenges and Opportunities**, 2022. (45)

[36] Wei Huang, Ye Shi, Zhongyi Cai, and Taiji Suzuki. **Understanding Convergence and Generalization in Federated Learning through Feature Learning Theory**. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Representation Learning*, **2024**, pages 25655–25686, 2024. (14)

[37] IBM Corporation. **IBM Federated Learning**. https://ibmfl.res.ibm.com/, 2025. Accessed: 2025-06-17. (11, 48)

[38] 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. (40)

[39] Prateek Jain and Purushottam Kar. **Non-convex Optimization for Machine Learning**. *Foundations and Trends® in Machine Learning*, **10**(3-4):142–363, 2017. (6)

[40] Zhongming Ji, Li Chen, Nan Zhao, Yunfei Chen, Guo Wei, and F. Richard Yu. **Computation Offloading for Edge-Assisted Federated Learning**. *IEEE Transactions on Vehicular Technology*, **70**(9):9330–9344, 2021. (46)

[41] Yalan Jiang, Dan Wang, Bin Song, and Shengyang Luo. **HDHRFL: A hierarchical robust federated learning framework for dual-heterogeneous and noisy clients**. *Future Generation Computer Systems*, **160**:185–196, 2024. (42)

[42] Yuang Jiang, Shiqiang Wang, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. **Model Pruning Enables Efficient Federated Learning on Edge Devices**. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2022. (39)

[43] Rajesh Kalakoti, Sven Nõmm, and Hayretdin Bahsi. **Federated Learning of Explainable AI(FedXAI) for deep learning-based intrusion detection in IoT networks**. *Computer Networks*, **270**:111479, 2025. (52)

[44] Goutham Kamath, Pavan Agnihotri, Maria Valero, Krishanu Sarker, and Wen-Zhan Song. **Pushing Analytics to the Edge**. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2016. (37)

[45] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. **SCAFFOLD: stochastic controlled averaging for federated learning**. In *Pro-*

ceedings of the 37th International Conference on Machine Learning, ICML'20. JMLR.org, 2020. (6, 51, 119)

[46] Latif U. Khan, Walid Saad, Zhu Han, and Choong Seon Hong. **Dispersed Federated Learning: Vision, Taxonomy, and Future Directions**. *IEEE Wireless Communications*, **28**(5):192–198, 2021. (41)

[47] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. **Federated Optimization: Distributed Machine Learning for On-Device Intelligence**. *CoRR*, **abs/1610.02527**, 2016. (38)

[48] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. **Federated Learning: Strategies for Improving Communication Efficiency**. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. (2, 38)

[49] Ajay D Kshemkalyani and Mukesh Singhal. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011. (28)

[50] Aditya Kumar and Satish Narayana Srirama. **Fog Enabled Distributed Training Architecture for Federated Learning**. In Satish Narayana Srirama, Jerry Chun-Wei Lin, Raj Bhatnagar, Sonali Agarwal, and P. Krishna Reddy, editors, *Big Data Analytics*, pages 78–92, Cham, 2021. Springer International Publishing. (4)

[51] Aditya Kumar and Satish Narayana Srirama. **FedStrag: Straggler-aware federated learning for low resource devices**. *Digital Communications and Networks*, 2024. (5, 47, 119)

[52] Aditya Kumar and Satish Narayana Srirama. **FIDEL: Fog integrated federated learning framework to train neural networks**. *Software: Practice and Experience*, **54**(2):186–207, 2024. (4, 97, 101, 118)

[53] Andrew Kusiak. **Federated explainable artificial intelligence (fXAI): a digital manufacturing perspective**. *International Journal of Production Research*, **62**(1-2):171–182, 2024. (52)

[54] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. **FedScale: Benchmarking Model and System Performance of Federated Learning at Scale**. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, **162** of *Proceedings of Machine Learning Research*, pages 11814–11827. PMLR, 17–23 Jul 2022. (49)

[55] Leslie LAMPORT and Nancy LYNCH. **CHAPTER 18 - Distributed Computing: Models and Methods**. In JAN VAN LEEUWEN, editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 1157–1199. Elsevier, Amsterdam, 1990. (2)

[56] Yann LeCun, Corinna Cortes, and CJ Burges. **MNIST handwritten digit database, 1998**. *http://yann.lecun.com/exdb/mnist*, **(Accessed 2 Dec 2024)**, 2010. (107)

[57] Li Li, Jun Wang, and ChengZhong Xu. **FLSim: An Extensible and Reusable Simulation Framework for Federated Learning**. In Houbing Song and Dingde Jiang, editors, *Simulation Tools and Techniques*, pages 350–369, Cham, 2021. Springer International Publishing. (11)

[58] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. **Federated Optimization in Heterogeneous Networks**. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, **2**, pages 429–450, 2020. (50, 119)

[59] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. **On the Convergence of FedAvg on Non-IID Data**. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. (6, 50, 119)

[60] Xingyu Li, Zhe Qu, Bo Tang, and Zhuo Lu. **Stragglers Are Not Disaster: A Hybrid Federated Learning Algorithm with Delayed Gradients**. *CoRR*, **abs/2102.06329**, 2021. (12, 46)

[61] Yipeng Li and Xinchen Lyu. **Convergence Analysis of Sequential Federated Learning on Heterogeneous Data**. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, **36**, pages 56700–56755. Curran Associates, Inc., 2023. (14, 51)

[62] Yiran Li, Hongwei Li, Guowen Xu, Tao Xiang, Xiaoming Huang, and Rongxing Lu. **Toward Secure and Privacy-Preserving Distributed Deep Learning in Fog-Cloud Computing**. *IEEE Internet of Things Journal*, **7**(12):11460–11472, 2020. (39)

[63] Feiyuan Liang, Qinglin Yang, Ruiqi Liu, Junbo Wang, Kento Sato, and Jian Guo. **Semi-Synchronous Federated Learning Protocol With Dynamic Aggregation in Internet of Vehicles**. *IEEE Transactions on Vehicular Technology*, **71**(5):4677–4691, 2022. (43)

[64] Ji Liu, Juncheng Jia, Tianshi Che, Chao Huo, Jiaxiang Ren, Yang Zhou, Huaiyu Dai, and Dejing Dou. **FedASMU: efficient asynchronous federated learning with dynamic staleness-aware model update**. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. (47)

[65] Juncai Liu, Jessie Hui Wang, Chenghao Rong, Yuedong Xu, Tao Yu, and Jilong Wang. **FedPA: An adaptively partial model aggregation strategy in Federated Learning**. *Computer Networks*, **199**:108468, 2021. (12, 46)

[66] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. **FATE: an industrial grade platform for collaborative learning with data protection**. *J. Mach. Learn. Res.*, **22**(1), January 2021. (11, 49)

[67] Yiran Liu, Ye Dong, Hao Wang, Han Jiang, and Qiuliang Xu. **Distributed Fog Computing and Federated-Learning-Enabled Secure Ag-**

gregation for IoT Devices. *IEEE Internet of Things Journal*, **9**(21):21025–21037, 2022. (40)

[68] Luis M. Lopez-Ramos, Florian Leiser, Aditya Rastogi, Steven Hicks, Inga Strümke, Vince I. Madai, Tobias Budig, Ali Sunyaev, and Adam Hilbert. **Interplay between Federated Learning and Explainable Artificial Intelligence: a Scoping Review**, 2025. (52)

[69] Renhao Lu, Weizhe Zhang, Qiong Li, Hui He, Xiaoxiong Zhong, Hongwei Yang, Desheng Wang, Zenglin Xu, and Mamoun Alazab. **Adaptive asynchronous federated learning**. *Future Generation Computer Systems*, **152**:193–206, 2024. (43)

[70] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. **Differentially Private Asynchronous Federated Learning for Mobile Edge Computing in Urban Informatics**. *IEEE Transactions on Industrial Informatics*, **16**(3):2134–2143, 2020. (39)

[71] Sam Lucero et al. **IoT platforms: enabling the Internet of Things**. *White paper*, 2016. (25)

[72] Scott M. Lundberg and Su-In Lee. **A unified approach to interpreting model predictions**. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. (6)

[73] Siqi Luo, Xu Chen, Qiong Wu, Zhi Zhou, and Shuai Yu. **HFEL: Joint Edge Association and Resource Allocation for Cost-Efficient Hierarchical Federated Edge Learning**. *IEEE Transactions on Wireless Communications*, **19**(10):6535–6548, 2020. (38, 39)

[74] Qianpiao Ma, Yang Xu, Hongli Xu, Zhida Jiang, Liusheng Huang, and He Huang. **FedSA: A Semi-Asynchronous Federated Learning Mechanism in Heterogeneous Edge Computing**. *IEEE Journal on Selected Areas in Communications*, **39**(12):3654–3672, 2021. (42, 85)

[75] Stephan Matzka. **Explainable Artificial Intelligence for Predictive Maintenance Applications**. *2020 Third International Conference on Artificial Intelligence for Industries (AI4I)*, pages 69–74, 2020. (128)

[76] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. **Communication-Efficient Learning of Deep Networks from Decentralized Data**. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, **54** of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017. (2, 38)

[77] Jed Mills, Jia Hu, and Geyong Min. **Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT**. *IEEE Internet of Things Journal*, **7**(7):5986–5994, 2020. (39)

[78] Sambit Kumar Mishra, Subham Kumar Sahoo, and Chinmaya Kumar Swain. **A Systematic Review on Federated Learning in Edge-Cloud Continuum**. *SN Comput. Sci.*, **5**(7), September 2024. (3)

[79] Anwesha Mukherjee and Rajkumar Buyya. **Federated Learning Architectures: A Performance Evaluation With Crop Yield Prediction Application**. *Software: Practice and Experience*, **55**(7):1165–1184, 2025. (41)

[80] Ambigavathi Munusamy, Mainak Adhikari, Mohammad Ayoub Khan, Varun G. Menon, Satish Narayana Srirama, Linss T. Alex, and Mohammad R. Khosravi. **Edge-Centric Secure Service Provisioning in IoT-Enabled Maritime Transportation Systems**. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10, 2021. (37)

[81] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. **Federated Learning for Internet of Things: A Comprehensive Survey**. *IEEE Communications Surveys & Tutorials*, **23**(3):1622–1658, 2021. (38)

[82] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. **Federated Learning with Buffered Asynchronous Aggregation**. In Gustau Camps-Valls,

Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, **151** of *Proceedings of Machine Learning Research*, pages 3581–3607. PMLR, 28–30 Mar 2022. (43, 45)

[83] NVIDIA Corporation. **NVIDIA FLARE: Federated Learning Application Runtime Environment**. https://developer.nvidia.com/flare, 2025. Accessed: 2025-06-17. (11, 48)

[84] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. **Sageflow: Robust Federated Learning against Both Stragglers and Adversaries**. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, **34**, pages 840–851. Curran Associates, Inc., 2021. (47)

[85] Sarthak Pati, Sourav Kumar, Amokh Varma, Brandon Edwards, Charles Lu, Liangqiong Qu, Justin J. Wang, Anantharaman Lakshminarayanan, Shih han Wang, Micah J. Sheller, Ken Chang, Praveer Singh, Daniel L. Rubin, Jayashree Kalpathy-Cramer, and Spyridon Bakas. **Privacy preservation for federated learning in health care**. *Patterns*, **5**(7):100974, 2024. (52)

[86] Pinyarash Pinyoanuntapong, Prabhu Janakaraj, Pu Wang, Minwoo Lee, and Chen Chen. **FedAir: Towards Multi-hop Federated Learning Over-the-Air**. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5, 2020. (40)

[87] Partha Pratim Ray. **A survey of IoT cloud platforms**. *Future Computing and Informatics Journal*, **1**(1-2):35–46, 2016. (27)

[88] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. **Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity**. *IEEE Journal on Selected Areas in Information Theory*, **3**(2):197–205, 2022. (46)

[89] José Ribeiro, Ricardo Santos, Cesar Analide, and Fábio Silva. **Implementing Federated Learning and Explainability Techniques in Regression Models to Increase Transparency and Reliability**. *Studies in Informatics and Control*, **33**(4):15–24, 2024. (51)

[90] Gaith Rjoub, Jamal Bentahar, and Omar Abdel Wahab. **Explainable Trust-aware Selection of Autonomous Vehicles Using LIME for One-Shot Federated Learning**. In *2023 International Wireless Communications and Mobile Computing (IWCMC)*, pages 524–529, 2023. (52)

[91] Nuria Rodríguez-Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámara, Gerardo González-Seco, M. Victoria Luzón, Miguel Angel Veganzones, and Francisco Herrera. **Federated Learning and Differential Privacy: Software tools analysis, the Sherpa.ai FL framework and methodological guidelines for preserving data privacy**. *Information Fusion*, **64**:270–292, 2020. (11)

[92] Théo Ryffel and the OpenMined community. **Federated Learning and Additive Secret Sharing using the PySyft framework**. https://openmined.org/blog/federated-learning-additive-secret-sharing-pysyft/, September 2020. Accessed: 2025-06-17. (11, 49)

[93] Rituparna Saha, Sudip Misra, and Pallav Kumar Deb. **FogFL: Fog-Assisted Federated Learning for Resource-Constrained IoT Devices**. *IEEE Internet of Things Journal*, **8**(10):8456–8463, 2021. (41)

[94] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. **Distributed Federated Learning for Ultra-Reliable Low-Latency Vehicular Communications**. *IEEE Transactions on Communications*, **68**(2):1146–1159, 2020. (41)

[95] Guto Leoni Santos, Matheus Ferreira, Leylane Ferreira, Judith Kelner, Djamel Sadok, Edison Albuquerque, Theo Lynn, and Patricia Takako Endo. *Integrating IoT + Fog + Cloud Infrastructures: System*

Modeling and Research Challenges, chapter 3, pages 51–78. John Wiley & Sons, Ltd, 2019. (2)

[96] Sanjay Sarma, David L Brock, and Kevin Ashton. **The networked physical world**. *Auto-ID Center White Paper MIT-AUTOID-WH-001*, pages 1–16, 2000. (25)

[97] Stefano Savazzi. **Federated Learning: example dataset (FMCW 122GHz radars)**, 2019. (64)

[98] Stefano Savazzi, Monica Nicoli, and Vittorio Rampa. **Federated Learning With Cooperating Devices: A Consensus Approach for Massive IoT Networks**. *IEEE Internet of Things Journal*, **7**(5):4641–4654, 2020. (64)

[99] Scaleout Systems. **FEDn: A Scalable Federated Learning Framework**. https://www.scaleoutsystems.com/framework, 2025. Accessed: 2025-06-17. (49)

[100] Detlef Schoder. *Introduction to the Internet of Things*, chapter 1, pages 1–50. John Wiley & Sons, Ltd, 2018. (25)

[101] Jungwon Seo, Ferhat Ozgur Catak, and Chunming Rong. **Understanding Federated Learning from IID to Non-IID dataset: An Experimental Study**. In *36th Norwegian ICT Conference for Research and Education, NIKT*, 2024. (51)

[102] Sajjad Hussain Shah and Ilyas Yaqoob. **A survey: Internet of Things (IOT) technologies, applications and challenges**. In *2016 IEEE Smart Energy Grid Engineering (SEGE)*, pages 381–385, 2016. (117)

[103] Wilson Valdez Solis, Juan Marcelo Parra-Ullauri, and Attila Kertesz. **Exploring the Synergy of Fog Computing, Blockchain, and Federated Learning for IoT Applications: A Systematic Literature Review**. *IEEE Access*, **12**:68015–68060, 2024. (3)

[104] Satish Narayana Srirama. **A Decade of Research in Fog computing: Relevance, Challenges, and Future Directions**. *Software: Practice and Experience*, 2023. (36)

[105] Satish Narayana Srirama. **Distributed edge analytics in edge-fog-cloud continuum**. *Internet Technology Letters*, **8**(3):e562, 2025. (118)

[106] Satish Narayana Srirama, Freddy Marcelo Surriabre Dick, and Mainak Adhikari. **Akka framework based on the Actor model for executing distributed Fog Computing applications**. *Future Generation Computer Systems*, **117**:439–452, 2021. (37)

[107] Satish Narayana Srirama and Deepika Vemuri. **CANTO: An actor model-based distributed fog framework supporting neural networks training in IoT applications**. *Computer Communications*, **199**:1–9, 2023. (37)

[108] Konstantinos D. Stergiou and Konstantinos E. Psannis. **Federated Learning Approach Decouples Clients From Training a Local Model and With the Communication With the Server**. *IEEE Transactions on Network and Service Management*, **19**(4):4213–4218, 2022. (41, 84)

[109] Khadija Sultana, Khandakar Ahmed, Bruce Gu, and Hua Wang. **Elastic Optimization for Stragglers in Edge Federated Learning**. *Big Data Mining and Analytics*, **6**(4):404–420, 2023. (46)

[110] Mohit Taneja, John Byabazaire, Nikita Jalodia, Alan Davy, Cristian Olariu, and Paul Malone. **Machine Learning Based Fog Computing Assisted Data-Driven Approach for Early Lameness Detection in Dairy Cattle**. *Comput. Electron. Agric.*, **171**(C), apr 2020. (38)

[111] Nurbek Tastan, Samar Fares, Toluwani Aremu, Samuel Horvath, and Karthik Nandakumar. **Redefining contributions: shapley-driven federated learning**. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. (52)

[112] Petroc Taylor. **Data growth worldwide 2010-2028 — Statista — statista.com**. https://www.statista.com/statistics/871513/worldwide-data-created/. [Accessed 13-03-2025]. (1)

[113] Shreshth Tuli, Redowan Mahmud, Shikhar Tuli, and Rajkumar Buyya. **FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing**. *Journal of Systems and Software*, **154**:22–36, 2019. (37)

[114] Bin Wang, Zhao Tian, Jie Ma, Wenju Zhang, Wei She, and Wei Liu. **A decentralized asynchronous federated learning framework for edge devices**. *Future Generation Computer Systems*, **166**:107683, 2025. (44)

[115] Jianxin Wang, Ming K. Lim, Chao Wang, and Ming-Lang Tseng. **The evolution of the Internet of Things (IoT) over the past 20 years**. *Computers & Industrial Engineering*, **155**:107174, 2021. (25)

[116] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. **Tackling the objective inconsistency problem in heterogeneous federated optimization**. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. (51, 119)

[117] Zhongyu Wang, Zhaoyang Zhang, Yuqing Tian, Qianqian Yang, Hangguan Shan, Wei Wang, and Tony Q. S. Quek. **Asynchronous Federated Learning Over Wireless Communication Networks**. *IEEE Transactions on Wireless Communications*, **21**(9):6961–6978, 2022. (45)

[118] Jiajun Wu, Fan Dong, Henry Leung, Zhuangdi Zhu, Jiayu Zhou, and Steve Drew. **Topology-aware Federated Learning in Edge Computing: A Comprehensive Survey**. *ACM Comput. Surv.*, **56**(10), June 2024. (3)

[119] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. **FederatedScope: A Flexible Federated Learning Platform for Heterogeneity**. *Proc. VLDB Endow.*, **16**(5):1059–1072, January 2023. (11)

[120] Chengjun Xu and Guobin Zhu. **Intelligent manufacturing Lie Group Machine Learning: real-time and efficient inspection system based on fog computing**. *Journal of Intelligent Manufacturing*, **32**, 01 2021. (38)

[121] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. **Asynchronous federated learning on heterogeneous devices: A survey**. *Computer Science Review*, **50**:100595, 2023. (45)

[122] Yunlu Yan, Chun-Mei Feng, Mang Ye, Wangmeng Zuo, Ping Li, Rick Siow Mong Goh, Lei Zhu, and CL Chen. **Rethinking client drift in federated learning: A logit perspective**. *arXiv preprint arXiv:2308.10162*, 2023. (51)

[123] Xiaofan Yu, Lucy Cherkasova, Harsh Vardhan, Quanling Zhao, Emily Ekaireb, Xiyuan Zhang, Arya Mazumdar, and Tajana Rosing. **Async-HFL: Efficient and Robust Asynchronous Federated Learning in Hierarchical IoT Networks**. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, IoTDI '23, page 236–248, New York, NY, USA, 2023. Association for Computing Machinery. (12, 46)

[124] Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and Salman Avestimehr. **Federated Learning for Internet of Things: Applications, Challenges, and Opportunities**, 2022. (45)

[125] Tuo Zhang, Chaoyang He, Tianhao Ma, Lei Gao, Mark Ma, and Salman Avestimehr. **Federated Learning for Internet of Things**. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 413–419, New York, NY, USA, 2021. Association for Computing Machinery. (39)

[126] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. **Staleness-Aware Async-SGD for Distributed Deep Learning**. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 2350–2356. AAAI Press, 2016. (47)

[127] Wenbo Zhang, Yuchen Zhao, Fangjing Li, and Hongbo Zhu. **A Hierarchical Federated Learning Algorithm Based on Time Aggregation in Edge Computing Environment**. *Applied Sciences*, **13**(9), 2023. (42)

[128] Xuyun Zhang, Muhammad Rizwan Anawar, Shangguang Wang, Muhammad Azam Zia, Ahmer Khan Jadoon, Umair Akram, and Salman Raza. **Fog Computing: An Overview of Big IoT Data Analytics**. *Wireless Communications and Mobile Computing*, **2018**:7157192, 2018. (36)

[129] Xiaokang Zhou, Wei Liang, Akira Kawai, Kaoru Fueda, Jinhua She, and Kevin I-Kai Wang. **Adaptive Segmentation Enhanced Asynchronous Federated Learning for Sustainable Intelligent Transportation Systems**. *IEEE Transactions on Intelligent Transportation Systems*, **25**(7):6658–6666, 2024. (44)

[130] Yajie Zhou, Xiaoyi Pang, Zhibo Wang, Jiahui Hu, Peng Sun, and Kui Ren. **Towards Efficient Asynchronous Federated Learning in Heterogeneous Edge Environments**. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 2448–2457, 2024. (44)

[131] Feng Zhu, Jiangshan Hao, Zhong Chen, Yanchao Zhao, Bing Chen, and Xiaoyang Tan. **STAFL: Staleness-Tolerant Asynchronous Federated Learning on Non-iid Dataset**. *Electronics*, **11**(3):314, 2022. (47)

[132] Hongbin Zhu, Yong Zhou, Hua Qian, Yuanming Shi, Xu Chen, and Yang Yang. **Online Client Selection for Asynchronous Federated Learning with Fairness Consideration**. *IEEE Transactions on Wireless Communications*, pages 1–1, 2022. (42)

List of Publications

- Aditya Kumar, S. N. Srirama: **Fog enabled distributed training architecture for federated learning**, 9th International Virtual Conference on Big Data Analytics (BDA 2021), Prayagraj, India, December 15-18, 2021, pp. 78-92. Springer. DOI: https://doi.org/10.1007/978-3-030-93620-4_7 (Scopus-Indexed)
- A. Kumar, S. N. Srirama: **FIDEL: Fog integrated federated learning framework to train neural networks**, Software: Practice and Experience, ISSN: 1097-024X, Volume: 54, Issue: 2, pp. 186-207, February, 2024. Wiley. DOI:10.1002/spe.3265 (SCIE, Q2, IF-2.6)
- A. Kumar, S. N. Srirama: **FedStrag: Straggler-aware federated learning for low resource devices**, Digital Communications and Networks, ISSN: 2352-8648 Volume: 11, Issue: 4, pp. 1214-1224, August, 2025. KeAi. DOI: 10.1016/j.dcan.2024.12.004. (SCIE, Q1, IF-7.5)

Ninth International Conference on Big Data Analytics (BDA 2021)

Indian Institute of Information Technology Allahabad (IIITA), Prayagraj, India

CERTIFICATE OF PARTICIPATION

This is to certify that **Aditya Kumar** from the **University of Hyderabad (UoH)**, has presented a paper titled **Fog enabled distributed training architecture for federated learning** in The Ninth International Conference on Big Data Analytics (BDA 2021), organized by Indian Institute of Information Technology Allahabad, India during 15-18 December 2021.

प्रज्ञानम् ब्रह्म

Dr. Pavan Chakraborthy
Organizing Chair

Dr. Sonali Agarwal
General Chair

Fog Enabled Distributed Training Architecture for Federated Learning

Aditya Kumar and Satish Narayana Srirama(✉)

School of Computer and Information Sciences, University of Hyderabad, Telangana 500046, India
{20mcpc03, satish.srirama}@uohyd.ac.in

Abstract. The amount of data being produced at every epoch of second is increasing every moment. Various sensors, cameras and smart gadgets produce continuous data throughout its installation. Processing and analyzing raw data at a cloud server faces several challenges such as bandwidth, congestion, latency, privacy and security. Fog computing brings computational resources closer to IoT that addresses some of these issues. These IoT devices have low computational capability, which is insufficient to train machine learning. Mining hidden patterns and inferential rules from continuously growing data is crucial for various applications. Due to growing privacy concerns, privacy preserving machine learning is another aspect that needs to be inculcated. In this paper, we have proposed a fog enabled distributed training architecture for machine learning tasks using resources constrained devices. The proposed architecture trains machine learning model on rapidly changing data using online learning. The network is inlined with privacy preserving federated learning training. Further, the learning capability of architecture is tested on a real world IIoT use case. We trained a neural network model for human position detection in IIoT setup on rapidly changing data.

Keywords: Internet of Things · Decentralized learning · Fog computing

1 Introduction

With advances in digital technology, Internet of Things (IoT) [6] devices are prevailing everywhere. Multiple sensors, cameras, mobiles, and smart gadgets are installed to provide support in decision making. As technology progresses, the reliance on such devices is increasing day by day. Deployment of various IoT devices has increased exponentially nowadays. The devices include simple sensors to very sophisticated industrial tools that exchange data/information through the internet. In the past few years, the number of IoT devices has increased rapidly. Currently, there are more than 10 billion IoT devices available worldwide, which is expected to be around 17 billion in 2025 and 26 billion by 2030 [8]. Every standalone device produces data which is shared with other devices for further processing. The IoT devices placed at the edge layer are generally resource

S. N. Srirama et al. (Eds.): BDA 2021, LNCS 13147, pp. 78–92, 2021.
https://doi.org/10.1007/978-3-030-93620-4_7

Received: 29 March 2023 | Revised: 31 July 2023 | Accepted: 21 August 2023

DOI: 10.1002/spe.3265

RESEARCH ARTICLE

WILEY

FIDEL: Fog integrated federated learning framework to train neural networks

Aditya Kumar | **Satish Narayana Srirama**

Cloud & Smart Lab, School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India

Correspondence
Satish Narayana Srirama, Cloud & Smart Lab, School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India.
Email: satish.srirama@uohyd.ac.in

Funding information
SERB, India, Grant/Award Number: CRG/2021/003888; UoH-IoE by MHRD, India, Grant/Award Number: F11/9/2019-U3(A)

Abstract

Technological advancement in the digital era has continued to produce voluminous amounts of data through various devices. Even though data is produced distributively, it needs to be accumulated centrally for processing, analysis, and knowledge extraction that faces several challenges such as bandwidth, latency, congestion, privacy, and security. Fog computing paradigm addresses some of these issues, and can be used as a distributed data processing unit. Federated learning trains a shared model over distributed nodes. However, a fog node can not process continuously growing data due to computational limitations. In this paper, we propose FIDEL: a fog integrated federated learning framework for neural network training using resource-constrained devices. The federation of resource-constrained Internet of Things (IoT) devices creates a shared global model trained on local data, which is generalized on the unseen dataset for prediction/inferences. We have also designed an online training scheme to process continuous data with limited compute resources. The FIDEL supports both synchronous and asynchronous federate learning that empowers resource-constrained devices to train machine learning models. To test the learning capabilities of the FIDEL, we have trained three neural networks (i) Shallow network; (ii) Deep Network; (iii) Convolutional Neural Network (CNN) models for human position detection in industrial IoT setup on rapidly changing datasets. The experimental results show that the framework can learn input–output relationships with significantly high accuracy. The overall system efficiency of the framework is reasonable in terms of latency and memory usage for resource-constrained devices.

KEYWORDS

decentralized training, distributed computing, federated learning, fog computing, Internet of Things

Abbreviations: AFL, Asynchronous federated learning; CNN, Convolutional Neural Network; DN, Deep Networks; FMCW, Frequency-Modulated Continuous Wave; FL, Federated learning; HR, Human-robot; IoT, Internet of Things; IIoT, industrial internet of things; KNN, K-Nearest Neighbors; MLP, Multi-layer Perceptron; MQTT, Message Queuing Telemetry Transport; RNN, Recurrent Neural Networks; SFL, Synchronous federated learning; SN, Shallow Networks.

Digital Communications and Networks 11 (2025) 1214–1224

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

FedStrag: Straggler-aware federated learning for low resource devices ☆

Aditya Kumar, Satish Narayana Srirama *

Cloud & Smart Lab, School of Computer and Information Sciences, University of Hyderabad, India

ARTICLE INFO

Keywords:
Internet of things
Decentralized training
Fog computing
Federated learning
Distributed computing
Straggler

ABSTRACT

Federated Learning (FL) has become a popular training paradigm in recent years. However, stragglers are critical bottlenecks in an Internet of Things (IoT) network while training. These nodes produce stale updates to the server, which slow down the convergence. In this paper, we studied the impact of the stale updates on the global model, which is observed to be significant. To address this, we propose a weighted averaging scheme, FedStrag, that optimizes the training with stale updates. The work is focused on training a model in an IoT network that has multiple challenges, such as resource constraints, stragglers, network issues, device heterogeneity, etc. To this end, we developed a time-bounded asynchronous FL paradigm that can train a model on the continuous inflow of data in the edge-fog-cloud continuum. To test the FedStrag approach, a model is trained with multiple stragglers scenarios on both Independent and Identically Distributed (IID) and non-IID datasets on Raspberry Pis. The experiment results suggest that the FedStrag outperforms the baseline FedAvg in all possible cases.

1. Introduction

The amount of data being generated has exponentially increased over the last decade. This data is continuously generated by various sources, including mobile phones, satellites, sensors, smart appliances, etc, in a distributed way. Training machine learning models on such distributed data can extract useful/relevant insight from raw data, which is crucial for various applications such as smart homes, smart cities, and intelligent predictions. Traditionally, data is sent to powerful servers for training and predictions. However, sending all the raw data to the server poses various challenges, including network bandwidth, congestion, latency, privacy, and security. Federated learning has become one of the popular paradigms for addressing a few of the challenges. FL is introduced to train a machine learning model on decentralized data. The core idea of the FL is to train a model that can ensure user-level privacy by not sharing raw data with the server. It learns a global model collaboratively from multiple locally trained models. All the training is done at the user/client level, and only trained parameters are shared with the server rather than raw data. The server aggregates all the locally trained models into a single model called a global model, which is sent back to the clients for further training.

Due to the distributed nature of training, FL is well suited for edge-fog-cloud continuum based training [1,2]. An IoT network with various devices can provide end-to-end solutions for a real-world problem. Here, federated learning is a promising paradigm that can leverage massively distributed data and computational resources over the IoT network. The edge layer of the network includes data-generating devices such as sensors, cameras, wearables, etc. The fog is a layer between the cloud and the edge to facilitate latency/communication efficient services to the end devices [3,4]. This improves efficiency and communication that enables numerous applications such as smart city, industrial IoT, autonomous vehicle health monitoring, etc [5]. Nodes in the fog layer also have sufficient computational power to process a few data points. This enables a fog node to process data locally and restrict the raw data to the user's vicinity. The cloud/server node is available to aggregate all the locally trained models. It waits for clients to share their updates and combines all learning to create a global model. Since fog nodes only communicate with the server for parameter sharing, not for raw data, it also saves a significant amount of bandwidth intrinsically.

However, in the FL paradigm, the server has to wait for the latest parameter updates, which may cause a training bottleneck for the entire network. The problem aggravates when there are stragglers in the network, or the network is unreliable [6]. The impact of stragglers on the synchronous federated learning is as high as it may hamper entire training. However, in practice, the server may continue the aggregation process with a minimum number of updates (fraction of clients).

☆ Peer review under the responsibility of the Chongqing University of Posts and Telecommunications.

* Corresponding author.
E-mail address: satish.srirama@uohyd.ac.in (S.N. Srirama).

https://doi.org/10.1016/j.dcan.2024.12.004
Received 6 June 2024; Received in revised form 5 December 2024; Accepted 11 December 2024

Available online 12 December 2024

Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration

by Aditya Kumar

Submission date: 21-Aug-2025 12:01PM (UTC+0530)
Submission ID: 2732756986
File name: Aditya_Kumar.pdf (35.26M)
Word count: 46374
Character count: 247204

Design and Feasibility Analysis of Federated Learning on Edge-to-Cloud Continuum: Framework Development, Addressing Stragglers and Explainability Integration

ORIGINALITY REPORT

Actual plagiarism percentage = 38 - 15 - 14 - 7 = 2%.

S. Satish

38%	11%	38%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Aditya Kumar, Satish Narayana Srirama. "FedStrag: Straggler-aware federated learning for low resource devices", Digital Communications and Networks, 2024 Publication	15%
2	Aditya Kumar, Satish Narayana Srirama. "FIDEL: Fog integrated federated learning framework to train neural networks", Software: Practice and Experience, 2023 Publication	14%
3	arxiv.org Internet Source (Student publication - Aditya kumar)	7%
4	Sonali Dhananjay Patil, Rajesh Ingle, Amar Buchade, Vidy Potdar. "Decentralized Healing - Transforming Healthcare with Federated Learning and Blockchain Technologies", CRC Press, 2025 Publication	<1%
5	Latif U. Khan, Walid Saad, Zhu Han, Choong Seon Hong. "Dispersed Federated Learning: Vision, Taxonomy, and Future Directions", IEEE Wireless Communications, 2021 Publication	<1%
6	"Big Data Analytics", Springer Science and Business Media LLC, 2021 Publication	<1%

Satish Narayana Sri
Professor
School of Computer and Informatic
University of Hydera
Hyderabad-500 046,

Satish Narayana Sriram
Professor
School of Computer and Information Sci
University of Hyderabad
Hyderabad-500 046 India

Satish Narayana Srirama
Professor
School of Computer and Information Scienc
University of Hyderabad
Hyderabad-500 046 India

Satish Narayana Srirama
Professor
School of Computer and Information Scienc
University of Hyderabad
Hyderabad-500 046, India

7	fastercapital.com Internet Source	<1%
8	www.jetir.org Internet Source	<1%
9	Thangaprakash Sengodan, Sanjay Misra, M Murugappan. "Advances in Electrical and Computer Technologies", CRC Press, 2025 Publication	<1%
10	Feiyuan Liang, Qinglin Yang, Ruiqi Liu, Junbo Wang, Kento Sato, Jian Guo. "Semi-Synchronous Federated Learning Protocol with Dynamic Aggregation in Internet of Vehicles", IEEE Transactions on Vehicular Technology, 2022 Publication	<1%
11	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1%
12	Shrikaant Kulkarni, Jaiprakash Narain Dwivedi, Dinda Pramanta, Yuichiro Tanaka. "Edge Computational Intelligence for AI-Enabled IoT Systems", CRC Press, 2024 Publication	<1%
13	Md. Mahmodul Hasan, Tangina Sultana, Md. Delowar Hossain, Ashis Kumar Mandal, Ngo Thien Thu, Ga-Won Lee, Eui-Nam Huh. "The journey to cloud as a continuum: Opportunities, challenges, and research directions", ICT Express, 2025 Publication	<1%
14	www.researchgate.net Internet Source	<1%

15	Amit Kumar Tyagi, Shrikant Tiwari. "AI and Blockchain in Smart Grids - Fundamentals, Methods, and Applications", CRC Press, 2025 Publication	<1 %
16	Choong Seon Hong, Latif U. Khan, Mingzhe Chen, Dawei Chen, Walid Saad, Zhu Han. "Federated Learning for Wireless Networks", Springer Science and Business Media LLC, 2021 Publication	<1 %
17	www.coursehero.com Internet Source	<1 %
18	Juncheng Jia, Ji Liu, Chao Huo, Yihui Shen, Yang Zhou, Huaiyu Dai, Dejing Dou. "Efficient federated learning with timely update dissemination", Knowledge and Information Systems, 2025 Publication	<1 %
19	discovery.researcher.life Internet Source	<1 %
20	Ishaani Priyadarshini. "An Explainable Autoencoder-Based Feature Extraction Combined with CNN-LSTM-PSO Model for Improved Predictive Maintenance", Computers, Materials & Continua, 2025 Publication	<1 %
21	Submitted to University of Hyderabad, Hyderabad Student Paper	<1 %
22	Muhammad Ammar, Nadeem Javaid, Abdul Khader Jilani Saudagar, Imran Ahmed. "An optimized Deep and Active Learning oriented framework for intrusion detection in Internet	<1 %

of Sensor Things", Ain Shams Engineering Journal, 2025
Publication

23	Submitted to Coventry University Student Paper	<1 %
24	rgu-repository.worktribe.com Internet Source	<1 %
25	"NEO 2016", Springer Science and Business Media LLC, 2018 Publication	<1 %
26	ijsrset.com Internet Source	<1 %
27	Mubashir Husain Rehmani, Al-Sakib Khan Pathan. "Emerging Communication Technologies Based on Wireless Sensor Networks - Current Research and Future Applications", CRC Press, 2019 Publication	<1 %
28	paperswithcode.com Internet Source	<1 %
29	core.ac.uk Internet Source	<1 %
30	web.archive.org Internet Source	<1 %
31	"Handbook of Computer Networks and Cyber Security", Springer Science and Business Media LLC, 2020 Publication	<1 %
32	Prabh Deep Singh, Mohit Angurala. "Integration of Cloud Computing and IoT - Trends, Case Studies and Applications", CRC Press, 2024 Publication	<1 %

33	spiral.imperial.ac.uk Internet Source	<1 %
34	ebin.pub Internet Source	<1 %
35	Submitted to North South University Student Paper	<1 %
36	Submitted to Thames Valley University Student Paper	<1 %
37	pnrec.org Internet Source	<1 %
38	"Distributed Computing and Artificial Intelligence, 21st International Conference", Springer Science and Business Media LLC, 2025 Publication	<1 %
39	www.geeksforgeeks.org Internet Source	<1 %
40	Bin Wang, Zhao Tian, Jie Ma, Wenju Zhang, Wei She, Wei Liu. "A decentralized asynchronous federated learning framework for edge devices", Future Generation Computer Systems, 2024 Publication	<1 %
41	Submitted to University of Lancaster Student Paper	<1 %
42	"Advances on P2P, Parallel, Grid, Cloud and Internet Computing", Springer Science and Business Media LLC, 2022 Publication	<1 %
43	Pethuru Raj, T. Poongodi, Balamurugan Balusamy, Manju Khari. "The Internet of Things and Big Data Analytics - Integrated	<1 %

Platforms and Industry Use Cases", CRC Press, 2020
Publication

44	Submitted to University College London Student Paper	<1%
45	Vesna Knights, Olivera Petrovska, Marija Prchkovska. "Enhancing Smart Parking Management through Machine Learning and AI Integration in IoT Environments", IntechOpen, 2024 Publication	<1%
46	Submitted to APJ Abdul Kalam Technological University, Thiruvananthapuram Student Paper	<1%
47	Li Huang, Yifeng Yin, Zeng Fu, Shifa Zhang, Hao Deng, Dianbo Liu. "LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data", PLOS ONE, 2020 Publication	<1%

Exclude quotes
Exclude matches
Exclude bibliography