Moreau Envelopes-based Personalized Asynchronous Federated Learning: Improving Practicality in Distributed Machine Learning

by

Anwar Munther As'ad

A thesis
presented to Lakehead University
in fulfillment of the
thesis requirement for the degree of

Master of Science

Thunder Bay, Ontario, Canada, 2023

Examination Committee

Dr. Zubair Fadlullah
Supervisor
(Associate Professor, Department of Computer Science, Western University, London, ON, Canada.)
(Adujnct Professor, Department of Computer Science, Lakehead University, Thunder Bay, ON, Canada)

Dr. Muhammad Asaduzzaman
Internal Examiner
(Assistant Professor, Department of Computer Science, Lakehead University, Thunder Bay, Ontario, Canada.)

Dr. Al-Sakib Khan Pathan
External Examiner
(Professor, Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh)

Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Acknowledgment

In the name of Allah, the Most Gracious, the Most Merciful.

Firstly, I would like to express my deepest gratitude to Allah, the Almighty, for His continuous guidance, blessings, and strength throughout the journey of my Master's program in Computer Science, specializing in Artificial Intelligence, at Lakehead University.

I am immensely grateful to my thesis supervisor, Dr. Zubair Fadlullah, for their invaluable guidance, encouragement, and mentorship. Their expertise, patience, and support have been instrumental in the completion of this research. I would also like to extend my appreciation to my committee members, Dr. Muhammad Asaduzzaman and Dr. Al-Sakib Khan Pathan, for their insightful feedback and constructive criticism.

I am grateful to the faculty and staff at the Department of Computer Science, Lakehead University, for providing a stimulating and supportive academic environment.

I would like to express my heartfelt appreciation to my parents, for their unwavering support, prayers, and encouragement throughout my academic journey despite their sickness. Their love, sacrifices, and dedication have been my source of inspiration and motivation. My parents have constantly inspired me and pushed me to pursue my dreams, instilling in me the importance of hard work, perseverance, and faith. Their belief in my abilities has given me the courage and resilience to continue, even in the face of challenges.

Special thanks go to my dear friend, Anas Al-Ghabra, for their constant support, understanding, and companionship. They have made the experience of pursuing this degree enjoyable and memorable.

Finally, I dedicate this thesis to all those who have inspired me and believed in my abilities.

Abstract

Federated learning is a promising approach for training models on distributed data, driven by increasing demand in various industries. However, it faces several challenges, including communication bottlenecks and client data heterogeneity. Personalized asynchronous federated learning addresses these challenges by customizing the model for individual users based on their local data while trading model updates asynchronously. In this paper, we propose Personalized Moreau Envelopes-based Asynchronous Federated Learning (APFedMe) that combines personalized learning with asynchronous communication and Moreau Envelopes as clients' regularized loss functions. Our approach uses the Moreau Envelopes to handle non-convex optimization problems and employs asynchronous updates to improve communication efficiency while mitigating heterogeneity data challenges through a personalized learning environment. We evaluate our approach on several datasets and compare it with PFedMe, FedAvg, and PFedAvg federated learning methods. Our experiments show that APFedMe outperforms other methods in terms of convergence speed and communication efficiency. Then, we mention some well-performing implementations to handle missing data in distributed learning. Overall, our work contributes to the development of more effective and efficient federated learning methods that can be applied in various real-world scenarios.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Research objectives and significance . . . 2
- 1.3 Contributions . . . 2
- 1.4 Thesis Structure . . . 3

2 Federated Learning Use Cases **4**
- 2.1 Privacy . . . 4
- 2.2 Healthcare . . . 4
- 2.3 Finance . . . 5
- 2.4 Smart Vehicles . . . 6
- 2.5 Mobile Applications . . . 6
- 2.6 Remarks About the Use Cases . . . 7

3 Related Work **8**
- 3.1 Federated Learning . . . 8
- 3.2 Personalized Federated Learning . . . 8
- 3.3 Asynchronous Federated Learning . . . 11

4 Methodology **12**
- 4.1 PFedMe Problem Formulation . . . 12
- 4.2 PFedMe Algorithm Implementation . . . 14
- 4.3 Asynchronous Learning into PFedMe . . . 15
- 4.4 APFedMe Algorithm Implementation . . . 16

5 Experiments & Results **21**
- 5.1 Experiment Setup . . . 21
- 5.2 Data Distribution . . . 22
- 5.3 Running the Experiment . . . 25
  - 5.3.1 Final Remarks . . . 30

6 Practicality Showcase of Asynchrony in Federated Learning **31**
- 6.1 Dataset . . . 31
- 6.2 Experiment Setup . . . 32
- 6.3 Experiment Results . . . 33

7 Combating Missing Data in Local Models in the Federated Learning Framework 35
7.1 Causes of Missing Data 35
7.1.1 Causes of Missing Data in Federated Learning 36
7.2 Dealing with Missing Data 36
7.2.1 Dealing with Missing Data in Federated Learning 39
7.3 Evaluating Imputation Methods for Distributed Learning 42
7.3.1 Experiment Design 42
7.3.2 Test Results 42
7.3.3 Final Remarks 43

8 Conclusion 45

List of Figures

1.1 Schematic Diagram of Federated Learning 2

3.1 Comparison between synchronous and asynchronous updates 10

4.1 Visualization of APFedMe Algorithm Flow 20

5.1 MNIST Distribution: 10 Users with 2 Labels Each. 23
5.2 MNIST Distribution: 20 Users with 4 Labels Each. 24
5.3 Synthetic Distribution: 10 Users, $\alpha = 0.5$, $\beta = 0.5$. 24
5.4 Synthetic Distribution: 20 Users, $\alpha = 0.5$, $\beta = 0.5$. 25
5.5 APFedMe - MNIST - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red) 28
5.6 APFedMe - Synthetic - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red) 28
5.7 PFedMe - MNIST - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red) 29
5.8 PFedMe - Synthetic - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red) 29

6.1 DoS Attack Mitigation Through Federated Learning 32
6.2 Decentralized Setup: Showcase of Users' Performance in IDS Dataset using APFedMe in 400 Global Rounds 34
6.3 Centralized Setup: Centralized Model Performance in 40 Epochs 34

7.1 MICE Visual Outline for a Dataset with 3 Variables Containing Missing Data . . . 40
7.2 KNN - Nearest Neighbors (Mode Emputation) Example (k=3) 42
7.3 Comparison of KNN and MICE Imputation Methods 44

List of Tables

5.1 APFedMe - MNIST Dataset 26
5.2 PFedMe - MNIST Dataset 27
5.3 APFedMe - Synthetic Dataset 27
5.4 PFedMe - Synthetic Dataset 27

6.1 Centralized & Decentralized Setups: Model Performance 33

7.1 Artificial Dataset - Recovery Through Data Imputation 38
7.2 Dataset - Recovery Through LOCF and WOCF 38
7.3 KNN Imputation Results 43
7.4 MICE Imputation Results 44

Mathematical Notations

Chapter 3 — FedAvg	
Notation	**Meaning**
K	Number of clients
k	Local client
B	Local minibatch size
b	Local minibatch
E	Number of local epochs
η	Learning rate
t	Current round
S	Random set of clients
n_k	Number of samples in client k's dataset
m_t	Total number of samples in round t
P_k	Client k's dataset
f	Loss function
Chapter 3 — PFedAvg	
Notation	**Meaning**
K	Number of clients
k	Local client
α	Step size
η	Learning rate
T	Number of global rounds
t	Current round index
S	Random set of clients
w	Model weights
τ	Number inner rounds
D	Mini-batch of data
n_k	Number of samples in client k's dataset
m_t	Total number of samples in round t
f	Loss function

Chapter 3 — ASOFed	
K	Number of clients
k	Local client
η	Learning rate
T	Number of global rounds
t	Current round index
f	Loss function
Chapter 4 — (PFedMe & APFedMe)	
Notation	**Meaning**
T	Number of global rounds
R	Number of local update rounds
S	Size of the sampled client subset
λ	Regularization parameter
η	Learning rate
β	Global model controller coefficient
N	Total number of clients
D	Mini-batch of data
$\tilde{h}$	Local objective function
θ	Personalized Model
S	subset of clients
ϵ	Threshhold for stale updates
F	Definition of Moreau Envelopes
f	Loss function
ν	Accuracy level

Chapter 1

Introduction

1.1 Motivation

In recent years, machine learning has revolutionized the way we solve complex problems [1], leading to groundbreaking solutions that were once unimaginable. However, traditional machine learning models rely on centralized data storage, which can be problematic when handling sensitive or decentralized data. To address this issue, federated learning was introduced. Federated learning is a paradigm that enables multiple devices or nodes to collaboratively train a shared model without exchanging raw data or compromising privacy.

The main concept of federated learning is illustrated in Figure 1.1. Each device participating in the training process downloads a global model from a central server, and then uses its local data to train the model further. Once local training is complete, the device uploads gradient information to the central server, which aggregates updates from all or a subset of devices, computes an updated model, and sends it back to each device as a renewed global model. This process is repeated until the model meets the desired criteria [2].

Although federated learning has demonstrated significant potential, it faces several challenges that limit its practicality. One critical challenge is data heterogeneity across clients [3]. Each device may have different data distributions, causing variations in data quality and quantity. In other words, data across clients are likely to be Non-Independent and Identically Distributed (Non-IID). Additionally, some clients may have missing or undocumented data, further complicating the problem. These limitations can result in a biased model [4] and negatively impact its performance and convergence. Simply put, if some clients have more diverse or representative data than others, their influence on the model will be more significant, potentially leading to overfitting or underfitting.

Therefore, this research is motivated by the powerful applications of machine and deep learning in various sectors. However, many sectors deal with massive sensitive datasets, creating significant challenges when training a model [5, 6]. Federated learning can alleviate these constraints, potentially eliminating the aforementioned challenges and facilitating the development of robust solutions.

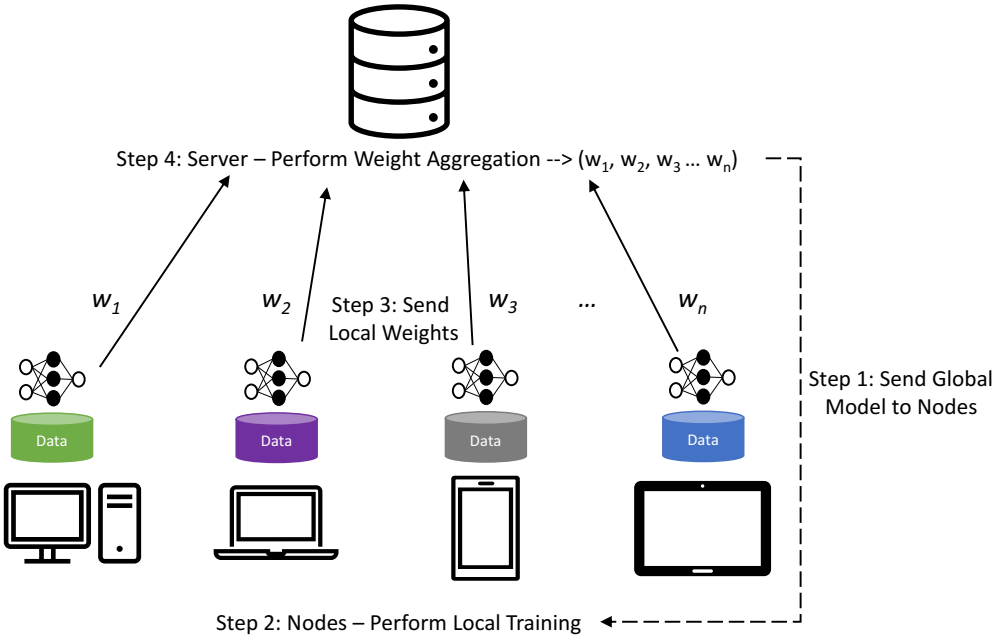

Figure 1.1: Schematic Diagram of Federated Learning

1.2 Research objectives and significance

The primary objective of this research is to enhance the scalability, data confidentiality for clients, and robustness of federated learning by integrating asynchronous communication into an existing state-of-the-art federated learning framework [3]. Additionally, this study addresses the challenges previously mentioned in distributed machine and deep learning. Finally, it also tackles the problem of missing labels in distributed learning, a well-known challenge in machine learning environments [7].

Applying impactful research findings on federated learning will not only help develop more efficient and effective distributed learning systems but also contribute to overcoming the difficulties faced when handling large sensitive data. In fact, federated learning has attracted significant interest from businesses, healthcare sectors, and the Internet of Things (IoT) [8], demonstrating its promising potential. Our objectives are to provide better-performing and more reliable federated learning systems, making them more practical for real-world scenarios.

1.3 Contributions

Through this research, we aim to provide a comprehensive overview of federated learning, its advantages, as well as the challenges faced when dealing with distributed learning environments. Furthermore, we highlight some of the advancements made in federated learning and offer an enhanced implementation to address some of the setbacks of distributed learning. More specifically:

1. We address the benefits and the use cases of federated learning.

2. We introduce a recent study in which a state-of-the-art federated learning environment was proposed, called Personalized Federated Learning with Moreau Envelopes (PFedMe) [3]. Additionally, we provide an explanation of FedAvg and personalized FedAvg, and illustrate the key differences between each of them.

3. We present Moreau Envelopes-based Personalized Asynchronous Federated Learning by utilizing PFedMe [3] as a basis for our solutions, and attempt to further improve the study by integrating asynchronous communication between the server and clients.

4. We explain in detail what personalized learning and asynchronous communication environments are. Using this information, we then demonstrate how both can provide benefits to client-server communications and Non-IID data in a distributed learning environment.

5. We present and discuss the findings and results of our enhanced implementation of PFedMe, and compare the results with PFedMe to provide a better understanding and comparison of our proposed method. Tests are performed on synthetic and real datasets to understand the overall performance better.

6. We showcase a simulated use-case of our enhanced solution on a dataset illustrates the practicality of asynchronous federated learning implementations.

7. We investigate how to better address missing labels in a distributed learning environment and present our findings to help mitigate client-side errors.

1.4 Thesis Structure

In Chapter 2, we will discuss the powerful use cases of federated learning to provide a better insight of where it can be applied. Chapter 3 explains the related work in the field of federated learning, specifically: the original form of federated learning, penalization, and asynchrony as well. Chapter 4 however discusses our methodology, where we thoroughly explain PFedMe, then propose asynchrony into the algorithm. Chapter 5 discusses our experiment results on datasets using our proposed methodology. Chapter 6 showcases a potential powerful application of our implementation, as well as test how the algorithm behaves as more users are added. Chapter 7 explains the problem of missing data in federated learning, provides potential solutions, and tests some of the proposed algorithms to impute missing data. Lastly, Chapter 8 provides a conclusion to our thesis, and provides insights of future work.

Chapter 2

Federated Learning Use Cases

Before we delve into our research topics, we will detail why federated learning is a groundbreaking methodology. Therefore, this chapter will focus on the use cases of federated learning, its benefits, and its enhanced advantages to machine and deep learning implementations.

2.1 Privacy

First of all, federated learning addresses one of the global concerns in the era of big data, which is user data privacy. As mentioned earlier, the nature of federated learning does not require the data to be centralized in order to train a model. This means that no user data is required to be gathered. Therefore, this concept mitigates the risk of data breaches. These days, we have been relying on artificial intelligence models to help enhance our solutions for real-world problems. However, if data privacy is not preserved, these solutions might not be feasible due to the dangers behind holding sensitive data. Therefore, distributed learning is a powerful model learning solution for data privacy concerns.

2.2 Healthcare

In a recent study, researchers performed a distributed learning task based on a deep convolutional neural network to train a global model for glioma brain tumor segmentation, aiming to identify tumor boundaries in medical imaging data [9]. Gliomas are tumors found in the central nervous system with various properties. Such a solution can help provide information to study tumor properties and treatment responses [9]. However, acquiring data to train such a model can be difficult due to data availability, legal, and privacy challenges [9]. Therefore, to showcase a solution, they designed federated learning and centralized deep learning models to compare their performance on the same validation datasets [9]. As a result, they were able to develop a global model generated from their federated learning setup that had comparable performance to the centralized setup. In fact, the centralized model only performed slightly better, with only around a 1% difference. Furthermore, to help provide a better idea of how federated learning can be used in healthcare, they split their dataset among different institutions to perform the distributed learning task. In other words, federated learning is able to allow institutions to take part in training a model without sharing their patients' data. In healthcare, some patients allow their data to be collected by the hospital

in which they are receiving treatment but do not permit their data to be shared with other parties. Therefore, federated learning has demonstrated its ability to eliminate privacy concerns regarding patients' data by illustrating that it can generate powerful models without a centralized setup.

In another study related to the recent pandemic, researchers implemented a federated learning model called "EXAM (Electronic Medical Record (EMR) Chest X-ray AI Model)" to help predict the clinical outcomes of patients with Covid-19 using vital signs and chest X-rays [10]. In their study, they developed a federated learning model trained on data from 16,148 patients across 20 different institutions around the globe. Specifically, they trained their model to collect data from patients suffering from Covid-19 to help predict their future oxygen requirements. This recent study is exceptional because it showcases the strengths of federated learning in critical real-life problems. In fact, utilizing only their distributed learning, they achieved nearly a 92% prediction score for the patients. Additionally, they developed their implementation without gathering user data from institutions. Furthermore, since the institutions were spread around the globe, their federated learning model had up to 38% better generalizability when compared to models trained with data from a single institute. The reason behind this is that it was trained on heterogeneous data, where patients had different kinds of symptoms and Covid-19 variants from different parts of the world. This illustrates another advantage of implementing a federated learning solution in healthcare.

These days, IoT devices are becoming more prevalent [11]. In fact, some smart devices, can read medical data, like heart rates, Oxygen levels, and generate an electrocardiogram through their built-in sensors [11, 12], which can be used to implement a federated learning model to predict future health risks and heart attacks. Therefore, federated learning undoubtedly provides robust solutions in the field of medicine without invading patients' privacy.

2.3 Finance

Another sector where federated learning excels at is finance [13]. Financial institutions, such as banks and insurance companies, hold sensitive customer data like social insurance numbers, credit card numbers, etc. Consequently, they may be reluctant to share this information with other institutions for artificial intelligence implementation purposes. Thus, if a bank decides to build a credit card fraud detection system, it will most likely have to rely solely on the data they have collected. A study was conducted in which researchers explored using a federated learning system to develop a credit card fraud detection system [14]. According to them, one of the main issues when implementing such a system is that it is challenging to gather enough data. For instance, credit card frauds are difficult to detect and do not occur frequently. As a result, the number of samples representing fraudulent transactions is minuscule, not providing enough data for a model to learn. Additionally, they mention that banks are typically not allowed to share transaction data, which increases the overall difficulty of the task. Therefore, they implemented a distributed learning simulation as a suggested solution, wherein they had their dataset containing a mix of fraudulent and standard transactions split among artificial clients. Moreover, they had to preprocess the data using the Synthetic Minority Over-sampling Technique [15] to balance out the data for the problem mentioned above. Specifically, their dataset had 284,807 instances, with only 492 instances representing fraudulent transactions. Nevertheless, they were able to develop a well-performing Convolutional Neural Network-based model through federated learning with an average validation

accuracy of 95.5%. Although it was a simulated setup, it still demonstrates the capabilities that federated learning can provide in the finance sector.

2.4 Smart Vehicles

One of the major trends around the world these days is autonomous driving [16]. Deep learning models have become extremely powerful, and they are nearly ready to drive vehicles autonomously. In fact, famous car brands, like Tesla, have shown that we are closer than ever to achieving complete autonomy in vehicle driving. However, in order to further enhance our chances of achieving that, we first need to address several challenges. Specifically, model generalization for autonomous driving might be hard to achieve since (1) not many drivers consent to collect their data due to privacy concerns [17], (2) deep learning models for autonomous driving require a substantial amount of data to learn [18], and (3) there might be high variance in data since there are numerous types of cars and driving conditions [19]. These challenges make federated learning highly applicable in this area of research. Specifically, we are able to eliminate the aforementioned challenges with distributed learning since no user data is collected, preserving drivers' privacy. Furthermore, cars worldwide can participate in the training process with different road and driving conditions. In addition, achieving autonomy does not only benefit road drivers, but it will also help in storage facilities, factories, etc. Hence, federated learning has a vast amount of use cases in the field of intelligent vehicles.

2.5 Mobile Applications

Another powerful use case of federated learning can be found in mobile applications. Smartphones these days have become extremely popular, with 68% of adults in the United States owning at least one in 2015 [20]. In fact, smartphones are ranked as the third most owned devices in the United States, and trends show that users are increasingly shifting to them [20]. Therefore, their widespread use and strong processing power can be exploited to develop new solutions. One study by Google researchers proposed an idea that utilizes federated learning to develop mobile keyboard prediction [21]. Since smartphones are popular and one of their most common uses is exchanging messages, developing solutions to enhance the reliability of the texting experience is important. As the research name indicates, their solution attempts to develop a model through federated learning that predicts the next most probable words that the user will type. For their implementation, they used the original form of federated learning along with a recurrent neural network language model to train devices that use the famous Google's keyboard application, Gboard. Furthermore, they developed a centralized model along with the distributed one to better compare performance. However, since federated learning does not collect users' data, the centralized and distributed models are not trained on the same data. Specifically, the centralized model trains on 7.5 billion instances, with an average sentence length of 4.1 words. On the other hand, the federated learning model trains on approximately 600 million instances spread across 1.5 million clients. The centralized model converges after 150 million steps of stochastic gradient descent, whereas the distributed learning model converges at around 3000 global rounds. They evaluated their models based on top-n recall metrics. Specifically, Top-1 recall evaluates the performance of the model based on

whether the most likely predicted word (highest probability of what the model suggests) matches the actual word that the user typed. Top-3 recall, on the other hand, considers whether the actual word that the user typed appears among the top 3 predicted words. The centralized model has a Top-1 recall score of 11.13% and a Top-3 recall score of 20.36% on average. On the other hand, the federated learning model scored 11.11% for Top-1 recall and 20.37% for Top-3 recall on average. Both models' performance is extremely comparable, with nearly no performance loss for utilizing federated learning. Therefore, this study showcases that federated learning is able to produce models nearly as powerful as the centralized ones on real-life applications while preserving privacy. Furthermore, this research highlights a powerful use case of federated learning in mobile applications — A solution that numerous users benefit from every day while exchanging messages using Gboard keyboard application.

Another application of federated learning in mobile applications is human activity recognition using smartphones [22]. Such human activities can include walking, running, climbing, sleeping, etc. In fact, detecting human activities provides a variety of uses in healthcare, sports, and even in virtual reality games [22]. Although most smartphones have the necessary sensors to train such a model (like accelerometers and gyroscopes [22]), not many users may consent to allowing third parties to collect such sensitive data, making federated learning a desirable implementation for this solution. In [22], they proposed a federated learning-based solution where they implement two distributed (simulated) and centralized learning setups. The first setup utilizes a deep neural network and the other one uses multinomial logistic regression. For their training and testing, they use a Human Activity Recognition dataset that contains a variety of activities, including biking, sitting, standing, walking, stair up, stair down, and null, recorded from 8 different Android smartphones and 2 smartwatches. Null activity indicates that there is no label for the given features, so they deleted such data instances. In their test results, they showcase that their distributed and centralized setups have similar validation accuracy, where the centralized model scored 93% with the deep neural network model, and 83% with the multinomial logistic regression. On the other hand, the decentralized setup scored 89% on the deep neural network model and 80% on multinomial logistic regression. Although the centralized setup slightly outperformed the distributed one, their performance is still fairly comparable. Furthermore, sensors across devices may be calibrated differently, which may result in lower validation accuracy in real-life scenarios [22]. However, distributed learning may remedy this problem by involving a variety of devices from different users to take part in the learning process, allowing for the development of a more generalized model.

2.6 Remarks About the Use Cases

Considering these use cases, federated learning has proven to be a powerful approach in various sectors, including healthcare and finance, along with mobile applications. Distributed learning helps preserve data privacy, provides good model performance, and facilitates cross-institutional collaboration. Federated learning's potential to tackle data privacy and model generalization challenges makes it a promising solution for a wide range of applications, such as medical predictions, fraud detection, and even autonomous driving. The aforementioned use cases of federated learning are only a hint of what distributed learning can achieve. Hence, further studies in this field will enhance the development of more powerful models to provide better solutions for real-world problems.

Chapter 3

Related Work

In this chapter, we discuss four related studies that have been previously published in the field of federated learning. These particular research studies were chosen for discussion due to their relevance to our proposed research topic and their significant contributions to the eventual development of APFedMe. In addition, simplified algorithms for the discussed studies are provided to help better understand the general infrastructure of distributed learning without straying from the scope of this chapter.

3.1 Federated Learning

In 2016, Google researchers introduced the term "Federated Learning," where they developed an algorithm called FedAvg [23], a form of federated learning, to mitigate data privacy concerns. Their version of federated learning works by averaging the model weights obtained from multiple local devices to update the global model. As shown in Algorithm 1, a subset of devices is randomly selected to perform the training process on each iteration. These devices receive the global model weights and use Stochastic Gradient Descent (SGD) to update their local models. Once a device completes its local training, it waits for a response from the global server to send its new parameters. This response is sent once all devices finish their local training. The global server then aggregates the local models' weights, averages them, and produces an updated global model to send to the clients. The algorithm keeps running for T number of global iterations. However, While they had both IID and Non-IID data distributions in mind during research and testing, the heterogeneity of data was still a limiting factor, which pushed further research in enhancing federated learning.

3.2 Personalized Federated Learning

In an attempt to alleviate the data diversity challenges in federated learning, personalization was being proposed in various research to be implemented into distributed learning environments. To put it simply, personalization in federated learning is to use clients' data to tailor model outputs for specific contexts or users' needs. Therefore, in one research [24], the authors propose a personalized variant by exploiting the idea behind Model-Agnostic Meta-Learning (MAML) framework. In MAML, the main objective is to find an initialization that performs well after being updated

Algorithm 1 FedAvg [23]

Require: The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Initialize w_0
for each round $t = 1, 2, \dots$ **do**
    $S_t =$ (random set of K clients)
  for each client $k \in S_t$ **do**
      $w_k^{t+1} =$ ClientUpdate(k, w^t)
  end for
    $w^{t+1} = \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
end for
ClientUpdate(k, w):
  $B =$ (split P_k into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in B$ **do**
      $w = w - \eta \nabla f(w; b)$
  end for
end for
  return w to server

with respect to a new task. Then, the problem can be reformulated by providing users with the initial point, which is then updated with respect to their loss function. Based on this concept, they proposed PFedAvg, a personalization influenced by MAML that works similarly to FedAvg. However, the main difference is that PFedAvg aims to find a global model to send to devices as an initialization. Algorithm 2 provides a better illustration where the global server initializes a global model to send to the clients, similar to how FedAvg works. Once a client receives the model, it calculates the Hessian matrix and the gradient with respect to its loss function on independent data batches to obtain a personalized model. Once the devices calculate their new weights, they send them back to the server once all the devices have finished their training to average them and continue the distributed learning operation. However, since they utilize the Hessian Matrix in their implementation, which might require high computational resources, it could make the implementation impractical to utilize in specific scenarios. With that being said, part of this concept will be used to help develop PFedMe [3].

In a related study sharing similar objectives with PFedAvg, PFedMe is proposed. As mentioned earlier, PFedMe is a federated learning algorithm that integrates personalized learning and Moreau Envelopes frameworks. This approach addresses the personalization challenge by employing a regularized loss function using the Moreau Envelopes for each client. This allows clients to pursue personalized models while remaining close to the global one. Furthermore, they are able to control how far or how close clients' models are by using a regularizer variable. In addition, unliked PFedAvg, PFedMe relies on gradient calculation rather than Hessian matrix computation. Moreover, incorporating Moreau Envelopes properties demonstrates the potential for improved convergence rates while adhering to assumptions related to bounded diversity in federated learning training data. PFedMe algorithm with additional details and analysis will later be provided in chapter 4.

Algorithm 2 PFedAvg [24]

Require: Initial weights w_0. The K clients are indexed by k; a is the step size, η is the learning rate, and T global rounds are indexed by t.

for each round $t = 1, 2, \ldots, T$ **do**
  $S_t =$ (random set of K clients)
  Server sends w_t to all users in S_t;
  for all $k \in S_t$ **do**
    $w^k_{t+1,0} = w_t$;
    for $r : 1$ to τ **do**
      $w^k_{t+1,r} = w^k_{t+1,r-1} - \alpha \nabla f_k(w^k_{t+1,r-1}, D^k_r)$;
      $w^k_{t+1,r} = w^k_{t+1,r-1} - \eta(I - \alpha \nabla^2 f_k(w^k_{t+1,r-1}, D''^k_r))\nabla f_k(w, D'^k_r)$;
    end for
    Client k sends w back to server;
  end for
  Server updates its model by averaging over received models: $w^{t+1} = \sum_{k \in S_t} \frac{n_k}{m_t} w^{t+1}_k$;
end for

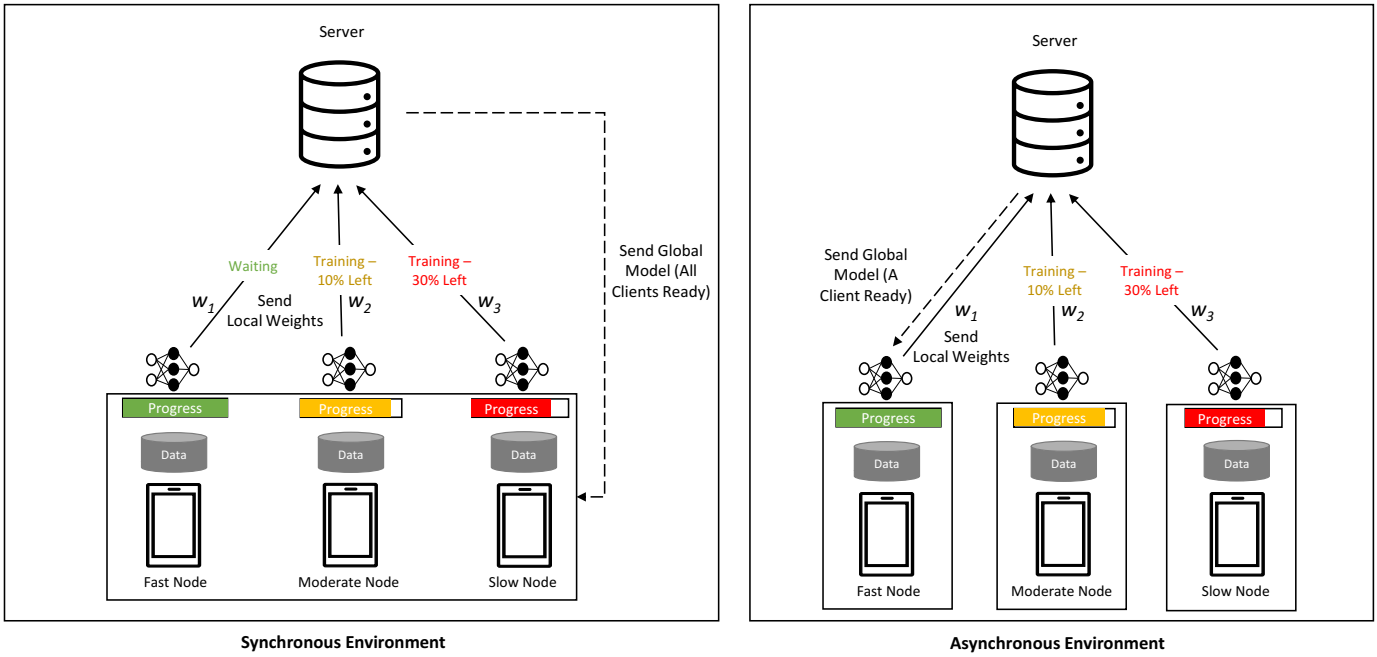

Figure 3.1: Comparison between synchronous and asynchronous updates

3.3 Asynchronous Federated Learning

In another research, where its focus is to lessen the aforementioned challenge of communication bottleneck, they propose Asynchronous Online Federated Learning (ASOFed) [25], an asynchronous method of distributed learning. In their implementation, they have the central model update its weight parameters right after receiving an update from a single client without waiting for the others. As illustrated in Algorithm 3 The algorithm starts similarly to other discussed federated learning implementations by initializing a global model and storing it in its memory. However, the server sends the global model to all clients rather than choosing a subset. Once a client receives the global model, it performs local training. A decay coefficient B prevents local models from deviating too much from previous learning since clients might have different update frequencies and data arrival rates. Once a single client is done with the local training process, weight parameters are sent to the server. However, instead of averaging the aggregated weights, the global model performs weight normalization and cross-client feature extraction to mitigate the inconsistencies that might arise in the model due to the asynchronous updates. Figure 3.1 Provides a better illustration of synchronous and asynchronous environments. In the case of synchronous implementation, the central server must wait since Device 1 lacks network connectivity; therefore, Device 3 is forced to wait until the device reconnects. Conversely, asynchronous updates demonstrate a more efficient process where waiting is unnecessary, resulting in a more streamlined workflow.

Algorithm 3 ASOFed [25]

Require: The K clients are indexed by k, η is the learning rate, and T global rounds are indexed by t.

Procedure at Central Server
for global iterations $t = 1, 2, \ldots, T$ **do**
    /* get the update on w_t */
    compute w_t
    update w_t with feature learning
end for
Procedure of Local Client k **at round** t
receive w_t from the server
Calculate Gradient ∇f^k
$w_{t+1}^k = w_t^k - \eta \nabla f^k$
upload w_{t+1}^k to the server

A different approach for an asynchronous implementation in federated learning was introduced in [26] proposing Distributed Online Multi-tasks (DOM). In DOM, they have the clients utilize the Soft Confidence-Weighted (SCW) classifier in their implementation that assumes a Gaussian distribution of weights. Clients process data in blocks, accumulating samples in a set-size buffer. When a client fully learns the data segment, it will develop an updated prediction model, forward its representative data, and model to the server. The server stores all clients' latest models and connections using a correlation matrix. It carries out asynchronous multi-task learning and updates the client's model without impacting others. Lastly, once the multi-task learning method produces the model, the server sends it to the client. However, despite its benefits, sharing representative data may reveal client information, raising potential privacy concerns.

Chapter 4

Methodology

This chapter covers the details of the PFedMe algorithm, its problem formulation, and implementation. However, the convergence analysis will not be included, as it has already been covered in great detail in [3]. In addition, this chapter discusses the differences in problem formulation between general, PFedAvg, and PFedMe federated learning algorithms. Furthermore, the chapter introduces our enhanced implementation, APFedMe, explaining how it is implemented and differs from PFedMe.

4.1 PFedMe Problem Formulation

First of all, the general problem of federated learning environments can be denoted as follows:

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{N} \sum_{i=1}^{N} f_i(w) \tag{4.1}$$

Where we attempt to minimize the average loss across participating clients. N denotes the number of clients communicating with the server and function f_i represents the expected loss of the client i. However, instead of attempting to solve the above problem, PFedMe tackles it using another approach. This approach involves the use of a regularized loss function with l_2-norm, which is represented as follows:

$$f_i(\theta_i) + \frac{\lambda}{2} ||\theta_i - w||^2 \tag{4.2}$$

Where λ is a regularizer variable that can be defined as one of the inputs, and θ represents the personalized model. As mentioned, λ is an input and can control how far or close the local models should be to the global model. A large value of λ forces a client to be closer to the global model, while a smaller λ value allows the client to be more personalized towards its data. In other words, if we are dealing with a client with unreliable data, it would be better to use a large λ value. Conversely, if we are confident in the quality of the client's data, a smaller λ would probably work better. However, defining a proper value for λ may vary from one case to another, and further analysis is preferable. Nonetheless, based on the above information and formulations, the PFedMe global and local problemsz can be defined as follows:

$$\text{PFedMe: } \min_{w\in\mathbb{R}^d}\{F(w) := \frac{1}{N}\sum_{i=1}^{N} F_i(w)\}, \quad \text{where } F_i(w) = \min_{\theta_i\in\mathbb{R}^d}\left[f_i(\theta_i) + \frac{\lambda}{2}||\theta_i - w||^2\right] \tag{4.3}$$

In the PFedMe implementation, the global model w is calculated by aggregating client parameters, whereas a local model θ is optimized based on its own data while maintaining a reasonable distance from the global model. Furthermore, it can be observed from problem formulation 4.3 that $F_i(w)$ is the definition of Moreau Envelopes, which will be used to help implement personalization characteristics into the clients. Therefore, we can define the optimal personalized model as follows:

$$\hat{\theta}_i(w) := \text{prox}_{f_i/\lambda}(w) = \underset{\theta_i\in\mathbb{R}^d}{\text{argmin}}\left[f_i(\theta_i) + \frac{\lambda}{2}||\theta_i - w||^2\right] \tag{4.4}$$

Furthermore, to formulate the gradient, PFedMe has proposed, "If f_i is convex or nonconvex with L-Lipschitz ∇f_i, then ∇f_i is L_F-smooth with $L_F = \lambda$ (with the condition that $\lambda > 2L$ for nonconvex L-smooth f_i)." Therefore, we can derive the gradient using the previous formulations:

$$\begin{aligned}
F_i(w) &= \min_{\theta_i\in\mathbb{R}^d} f_i(\theta_i) + \frac{\lambda}{2}\|\theta_i - w\|^2 \\
\frac{\partial F_i(w)}{\partial w} &= \frac{\partial}{\partial w}\left(f_i(\hat{\theta}_i(w)) + \frac{\lambda}{2}\|\hat{\theta}_i(w) - w\|^2\right) \\
&= 0 + \lambda(w - \hat{\theta}_i(w)) \\
&= \lambda(w - \hat{\theta}_i(w))
\end{aligned}$$

$$\nabla F_i(w) = \lambda(w - \hat{\theta}_i(w)) \tag{4.5}$$

Hence, we are able to compute the gradient using equation 4.5.

Moreover, according to [3], the closest implementation to PFedMe is PFedAvg. Specifically, PFedAvg's local, personalized client updates has similar problem formulation, which can be donated as follows:

$$\theta_i(w) = w - \alpha\nabla f_i(w) = \underset{\theta_i\in\mathbb{R}^d}{\arg\min}\,\langle\nabla f_i(w), \theta_i - w\rangle + \frac{1}{2\alpha}\|\theta_i - w\|^2 \tag{4.6}$$

However, there are a couple of crucial differences between PFedMe and PFedAvg. First of all, PFedAvg is primarily designed to perform a single-step gradient update for the local personalized model, whereas PFedMe can pursue multiple-step updates. Additionally, PFedAvg only optimizes the first-order approximation of the loss function, whereas PFedMe directly minimizes the loss function. Lastly, PFedAvg requires the estimation of the Hessian matrix, which according to [3] can be computationally expensive, while PFedMe only requires gradient calculations.

4.2 PFedMe Algorithm Implementation

Utilizing problem formulations explained in Chapter 4.1, we can properly formulate the pFedMe algorithm implementation. First of all, as mentioned earlier, the pFedMe algorithm provides a bi-level solution. At the outer level of a client model, pFedMe implements the local model weight update as follows:

$$w_{i,r+1}^t = w_{i,r}^t - \eta \nabla F_i(w_{i,r}^t) \tag{4.7}$$

where $\nabla F_i(w)$ is the gradient as shown in Equation 4.5, and $w_{i,r}^t$ is the local client model. However, at the inner client level, this is where they solve their client personalization problem, which they denote as $\hat{\theta}_i(w_{i,r}^t)$. Specifically, in the PFedMe algorithm, each client has to solve the personalization problem to find their personalized model. However, finding the exact personalized model can be computationally expensive and may require additional information that may not be readily available. In addition, finding an exact solution for the personalized model is difficult and may not be possible using a simple, direct solution. Therefore, to overcome this challenge, they use a δ-approximation of the personalized model, denoted as $\tilde{\theta}_i(w_{i,r}^t)$. The δ-approximation is an estimate of the personalized model that is close enough to the true personalized model, such that the expected difference between the two is maintained in a reasonable boundary and no greater than δ:

$$E[\|\tilde{\theta}_i(w_{i,r}^t) - \hat{\theta}_i(w_{i,r}^t)\|] \leq \delta \tag{4.8}$$

Furthermore, PFedMe achieves this estimation by approximating $\nabla f_i(\theta_i)$ through local clients' mini-batch of data, denoted as:

$$\nabla \tilde{f}_i(\theta_i, D_i) := \frac{1}{|D_i|} \sum_{\xi_i \in D_i} \nabla \tilde{f}_i(\theta_i; \xi_i) \tag{4.9}$$

where D_i represents the sample data taken from client i. In addition, they use an iterative approach to estimate the local personalized model. This is achieved by first defining the following formulation:

$$\tilde{h}_i(\theta_i; w_{i,r}^t, D_i) := \tilde{f}_i(\theta_i; D_i) + \frac{\lambda}{2}||\theta_i - w_{i,r}^t||^2 \tag{4.10}$$

Therefore, by applying modified accelerated gradient descent to help satisfy the following condition:

$$||\nabla \tilde{h}_i(\tilde{\theta}_i; w_{i,r}^t, D_i)||^2 \leq \nu \tag{4.11}$$

Where ν represents an accuracy level, they are able to approximate local personalized model with high accuracy. Moreover, according to [3], in PFedMe's implementation, they are able to adjust the approximation by adjusting $|D|$ and ν values.

Lastly, the global model update is fairly simple and close to how FedAvg works. However, they have presented β parameter to help control how much the averaged weights contribute to the global model. In fact, when β is set to equal 1, it can be seen that this is simply averaging the client weights. This hyperparameter can prevent unreliable clients from causing a major change to the global model, providing better flexibility to the PFedMe algorithm.

Using the above derivations, we are able now to finally introduce a fully constructed form of the PFedMe algorithm, which is shown in algorithm 4.

Algorithm 4 pFedMe: Personalized Federated Learning using Moreau Envelope Algorithm [3]

input: $T, R, S, \lambda, \eta, \beta, w^0$
for $t = 0$ to T **do**
  Server sends w_t to all clients
  for all $i = 1$ to N **do**
    $w_{i,0}^t = w_t$
    for $r = 0$ to R **do**
      Sample a fresh mini-batch D_i with size $|D|$ and minimize $\tilde{h}_i(\theta_i; w_{i,r}^t, D_i)$, to find a δ-approximate $\tilde{\theta}_i(w_{i,r}^t)$
      $w_{i,r+1}^t = w_{i,r}^t - \eta\lambda(w_{i,r}^t - \tilde{\theta}_i(w_{i,r}^t))$
    end for
  end for
  Server uniformly samples a subset of clients S^t with size S, and each of the sampled client sends the local model $w_{i,R}^t, \forall i \in S^t$, to the server.
  Server updates the global model: $w_{t+1} = (1-\beta)w_t + \beta \sum_{i \in S^t} \frac{w_{i,R}^t}{S}$
end for

Using the formulations above along with the algorithm outline, pFedMe algorithm can be outlined in the following steps:

1. Initialize the input parameters.
2. Perform global communication rounds:
   (a) The server sends the global model to all clients.
   (b) Each client performs local updates:
      i. Set their local model equal to the global model.
      ii. Perform local update rounds:
         A. Sample a mini-batch of data.
         B. Minimize a function $\tilde{h}_i$ to find a δ-approximate personalized model.
         C. Update the local model using the learning rate, λ, and the δ-approximate personalized model.
   (c) The server receives updated local models from a subset of clients.
   (d) The server updates the global model using the average of the received local models and parameter β.

4.3 Asynchronous Learning into PFedMe

Analyzing PFedMe in theory, it certainly seems to offer numerous benefits to distributed learning. It provides control over how much clients can contribute to the global model in each round, allowing us to mitigate the problem of local devices with unreliable data. Moreover, implementing

personalization into local clients not only improves performance but also enhances convergence speed. Therefore, we chose to further refine the PFedMe algorithm, as it brings significant advantages to federated learning environments.

One problem PFedMe does not address, which is common in almost every federated learning setup, is communication bottlenecks. As mentioned earlier, the way most distributed learning environments are implemented requires waiting for all the chosen clients in a global round to finish their training before moving to the next global round. With this implementation, clients that have already completed their tasks are forced to wait for other clients to finish their training before uploading their local models to the global server. Our enhanced method addresses this issue by implementing asynchronous distributed learning, which, as mentioned earlier, we call APFedMe.

Implementing an asynchronous environment into the PFedMe algorithm can be tricky. First of all, PFedMe updates the global model by averaging the chosen clients in a single round. There are a couple of obvious advantages to utilizing weight averaging in a synchronous environment, such as:

1. **Simplicity**: Averaging client weights is a straightforward method that averages local model updates into a single global update. This simplicity is attractive as it allows for efficient and rapid implementation.

2. **Robustness**: Averaging client weights incorporates an element of noise reduction, as the change to the global model of any single client is reduced by involving other clients. This makes the global model more robust against unreliable data from individual clients. In other words, weight averaging prevents clients that train their local models on unreliable data from contributing significantly to the global model, thereby mitigating inconsistencies caused by them.

These advantages are attractive for real-life scenarios when implementing a distributed learning environment, which is why we aim to maintain these benefits in our asynchronous solution. Specifically, our refined implementation, to be discussed in the latter part of this section, will not include weight averaging but still manages to provide a simple approach, as straightforward as weight averaging. Moreover, we will be able to maintain robustness in the implementation by utilizing the benefits that the nature of PFedMe offers to a distributed learning environment. Therefore, in our refined solution, we aim to incorporate an asynchronous learning environment while maintaining nearly all the advantages that typical federated learning and PFedMe provide.

4.4 APFedMe Algorithm Implementation

To better explain how an asynchronous implementation is integrated into PFedMe, we first have to understand the changes that need to be made. First of all, to incorporate asynchronous federated learning, we must change how the global model updates its weights. Specifically, the nature of PFedMe's implementation requires a synchronous environment, as it updates the global model using weight averaging. However, to average the chosen clients' weights, we would be forced to wait for multiple clients to send their weights before pushing an update, which would not work in an asynchronous environment. Therefore, the first step in implementing an asynchronous environment into PFedMe is eliminating weight averaging.

Furthermore, the main difference between APFedMe and PFedMe is that we attempt to refine the solution primarily on the global server of a distributed learning environment instead of the local clients. However, since removing weight averaging in our solution might compromise robustness and fairness in a federated learning environment, we must compensate for that. In fact, we do so by utilizing the β variable that PFedMe was presented and included in its implementation. The significant benefit of using β in global client updates is that we can control how each client contributes to the global model, reducing our reliance on weight averaging. Specifically, using this parameter allows us to maintain the fairness and robustness that weight averaging provides in federated learning. Therefore, with the reliance on the β value and removal of weight averaging, we can specify how weight updating will be implemented in the algorithm with the following:

$$w_{t+1} = (1 - \beta)w_t + \beta w_{i,R}^t \tag{4.12}$$

i represents the client number, and R represents the local rounds. Nonetheless, defining a proper β value can drastically improve both the global model performance and the convergence speed. However, for the time being, there is no direct method or a defined way to help in choosing the right value for it. Furthermore, manually adjusting the hyperparameter is possible for certain clients if you consider them trustworthy contributors to the global model.

The other change that APFedMe introduces to the original algorithm is how clients communicate with the server. As mentioned earlier, in a typical synchronous federated learning scenario, clients idle until they receive a message from the central server to send out their updates. However, to introduce asynchrony into APFedMe, clients will no longer be required to wait for others to finish before sending their new updates. Specifically, as soon as a client has finished their local updates, it sends out its local weights for the global model to update the weights immediately. The global server stores its model on the server and constantly updates it as soon as a new response arrives. With such an implementation, clients are constantly running, helping them utilize their resources more effectively.

With that being said, despite the fact that asynchronous federated learning provides excellent benefits, there is a side-effect that we need to address, which is the updates from stale clients. Stale clients are devices that participate in the learning process. However, due to their slow updates and communication with the global server, they might send out an old global model update, which will most likely decrease the model's performance and cause convergence issues. Hence, to prevent this from happening, clients who upload a relatively old model to the server will have their β value changed to prevent them from influencing the global model too much. This is a relatively simple solution to this problem since PFedMe's implementation has already included a method to control how strongly a client model will affect the global model. With this, we can mitigate this side effect if it ever occurs. Furthermore, we can detect this issue and control it by defining a threshold, ϵ, where we keep track of the number of global updates that happen before the server receives the slow client's weights. If the number exceeds the threshold, β will be modified accordingly to reduce the strength of this update's effect on the global model. However, once the global model is updated, the client with the slow update will have their β parameter reset since it may not always be the case that the client will consistently have a slow training and communication process. Using all the changes we mentioned earlier, we are able to construct APFedMe as shown in algorithm 5. Furthermore, Figure 4.1 outlines a diagram of APFedMe's algorithm flow to help better showcase both the implementation and asynchrony environments as well. Therefore, using the formulations above

along with the algorithm outline, APFedMe algorithm steps can be summarized in the following:

1. Initialize the input parameters.
2. Perform global communication rounds:
   (a) The server sends the global model to all clients.
   (b) Each client performs local updates:
      i. Set their local model equal to the global model.
      ii. Perform local update rounds:
         A. Sample a mini-batch of data.
         B. Minimize a function $\tilde{h}_i$ to find a δ-approximate personalized model.
         C. Update the local model using the learning rate, λ, and the δ-approximate personalized model.
      iii. Send the local model to the server as soon as the local training is done.
   (c) Server checks how many global model updates occurred during the client's training and modify β value if necessary.
   (d) The server updates the global model using the received local models and parameter β.
   (e) Reset β to original value if the client had a stale update.

Simply put, PFedMe attempts to address the general problem of federated learning environments through a unique approach that uses an l_2 optimizer. The main difference between PFedMe and its closest implementation, PFedAvg, is that PFedMe can pursue an iterative approach with multiple-step updates, directly minimizes the loss function, and only requires gradient calculations. In PFedMe's algorithm the global model is calculated by averaging the aggregated parameters from clients, while local models are optimized based on their data while maintaining a reasonable distance from the global model. This method is efficient when directly compared to PFedAvg's implementation since it does not require the estimation of Hessian matrix.

However, in order to address communication bottlenecks commonly found in federated learning setups, APFedMe, an asynchronous version of PFedMe, is introduced. This approach eliminates weight averaging by refining PFedMe's solution primarily on the global server of a distributed learning environment. By utilizing the β variable introduced in PFedMe, the asynchronous method maintains robustness in the learning environment while reducing reliance on weight averaging. Asynchronous federated learning environments suffer from stale client updates. APFedMe solves the stale problem by automatically reducing slow clients' distribution to the global server when their updates are sent. The main advantage of APFedMe is that it significantly reduces waiting times for clients, which will help improve efficiency and scalability in real-world applications.

APFedMe differs from the original federated learning [23] in several key aspects. While federated averaging relies on synchronous communication, requiring clients to coordinate and share their model updates simultaneously, asynchronous Learning allows clients to contribute updates independently, reducing communication bottlenecks. The integration of Moreau envelopes offers improved regularization, enhancing robustness of the learning process. Additionally, APFedMe also provides better control over client contributions to the global model, addressing potential issues with unreliable clients. Furthermore, the personalization of local clients tailors the learning

process to individual clients' data, mitigating the negative effects of Non-IID data on the model. This personalized approach results in more effective learning for each client and better overall model performance.

Now that all the algorithms have been thoroughly explained, along with their strengths and differences, we are ready to run the experiments and showcase the results, which will be shown in Chapter 5.

Algorithm 5 APFedMe: Asynchronous Personalized Federated Learning using Moreau Envelopes

input: $T, R, \lambda, \eta, \beta, w^0, \epsilon$
    Server:
for $t = 0$ to T **do**
    Server sends w_t to a client
    while clients are active **do**
        Receive local model $w_{i,R}^t$
        if $stale$ **then**
            Modify β_i for client i
        end if
        Update the global model: $w_{t+1} = (1 - \beta_i)w_{t+1} + \beta_i w_{i,R}^t$
        Reset β_i for client i
    end while
end for
    Client i**:**
for all $i = 1$ to N **do**
    Receive w_t from the server
    $w_{i,0}^t = w_t$
    for $r = 0$ to R **do**
        Sample a fresh mini-batch D_i with size $|D|$ and minimize $\tilde{h}_i(\theta_i; w_{i,r}^t, D_i)$, to find a
        δ-approximate $\tilde{\theta}_i(w_{i,r}^t)$
        $w_{i,r+1}^t = w_{i,r}^t - \eta\lambda(w_{i,r}^t - \tilde{\theta}i(w_{i,r}^t))$
    end for
    Send the local model $w_{i,R}^t$
end for

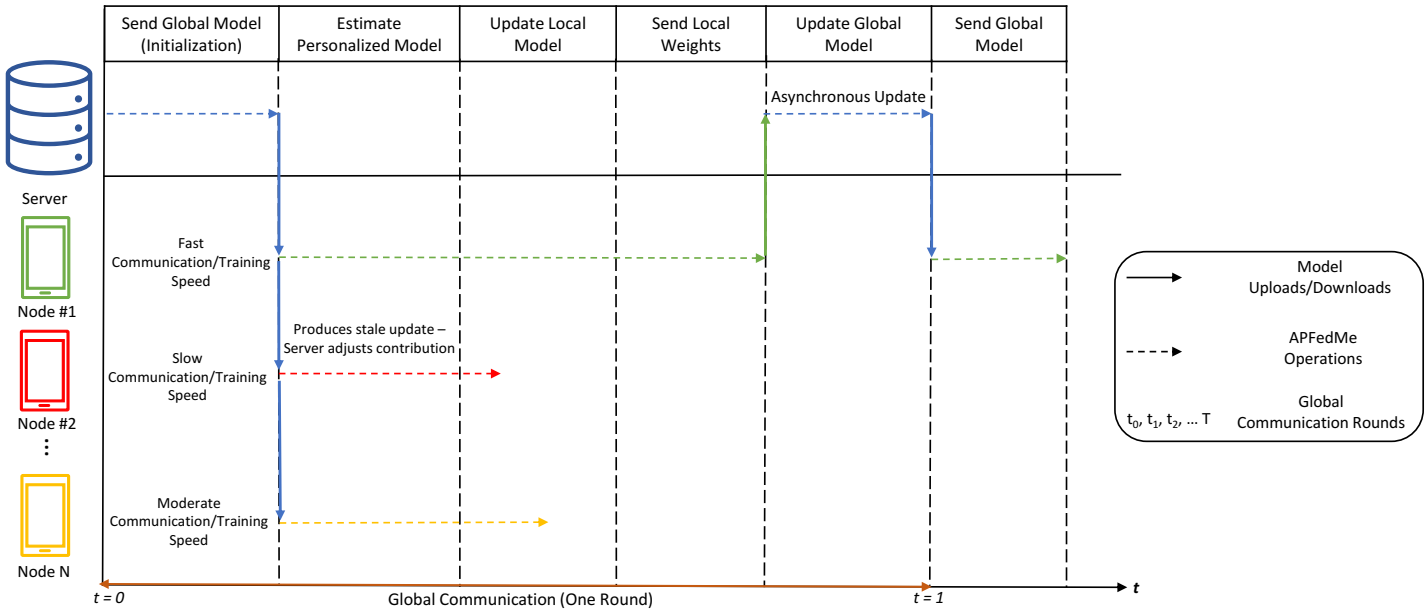

Figure 4.1: Visualization of APFedMe Algorithm Flow

Chapter 5

Experiments & Results

This chapter discusses our experiments on the abovementioned algorithms (PFedMe, APFedMe). To ensure that we are running our tests in scenarios that are relatively close to real-life implementations, we will be conducting our tests on Non-IID data. Precisely, we will split the datasets into different sets of clients, where each one has a different distribution than the others. Details on how these data are split will be provided later in the diagrams. Moreover, we will conduct our tests on synthetic data to generate a variety of scenarios as a way to test the algorithms comprehensively. Afterward, we will present our findings with detailed graphs and tables to help better outline the strengths and differences between the implementations.

5.1 Experiment Setup

As mentioned earlier, we will use multiple setups to test the limits of the algorithms, which will help us better understand their potential and how well they perform in real-life environments, so we can outline the feasibility of implementing such distributed learning methods in real-life applications. With that being said, the datasets that we will be using in the experiments are as follows:

1. **MNIST**: A large dataset collection of handwritten digits ranging from 0 to 9. It typically contains 60,000 training images and 10,000 testing images (7,000 images per class) [27].

2. **Synthetic**: Artificially generated data that is able to mimic the properties of real-world datasets. This kind of dataset is often used to test algorithms and models under various conditions, allowing us to evaluate the performance and robustness of the methods. Furthermore, in our particular setup, the synthetic dataset will provide the flexibility to control the number of users and their data distribution.

The reasons why we chose these particular datasets are first because of their relative simplicity. They are usually easy to deal with and pre-process if needed for machine learning tasks. In addition, these datasets are widely used in both machine learning and distributed learning tasks, allowing us to compare better and understand where our implementation stands. Lastly, their public availability allows us, and other implementers, to use a similar setup, providing ease of comparison in other studies and further refinements.

When testing each dataset, we will consider these factors where different parameters will be used on each run. (it is worth noting that in order to ensure consistency, each test will run 5 times, then average out the results):

1. **Data Distribution**: As mentioned earlier, we are splitting the datasets among clients with a Non-IID split, where we are also able to specify how many labels each user will get. This allows us to mimic real-life scenarios better since most distributed learning setups deal with non-IID data.

2. **Number of Clients**: We will be testing each run with 10 and 20 clients to understand better how well the implementations scale as the network of clients grows. 10 number of clients is particularly chosen since PFedMe had a similar experiment setup, which will help us better compare our results with their findings. In addition, we chose 20 clients for the other setup to further study how well the algorithms scale.

3. **Communication Rounds**: We will evaluate the implementations' performance over different numbers of communication rounds. However, when presenting our results, we will aim to find a reasonable number of communication rounds to use in our test cases, where going above that number does not yield too much difference across the different algorithms and setups as well.

4. **Regularizer and Controller Parameters**: We will be testing a variety of different numbers for λ and β hyperparameters, as they undoubtedly play a significant role in our asynchronous implementation of federated learning with Moreau Envelopes. In addition, they will also help us further comprehend how large or small values will affect the performance of PFedMe and APFedMe. The specifics of the values will be pointed out later in the results.

5. **Performance Metrics**: We will be using global model's prediction accuracy, as well as the loss values to help compare performance between the different implementations. In addition, to show where APFedMe stands out, we will be calculating the run-time speed for each run to use it as a performance metric, providing a better overview of the performance difference between synchronous and asynchronous federated learning environments.

Furthermore, the tests will be conducted in Google Colab [28] using Python. The main reason we chose Python is because it provides numerous flexible tools for machine and deep learning, as well as some tools that facilitate federated learning implementations along with excellent analytical tools. In addition, since we are running on Google Colab [28], the server-client communications and training is simulated. In other words, the server and the clients are artificial and all located within Google Colab's servers. With that being said, specifications of the hardware are hard to specify since Google Colab works by providing whatever resources are available on their side. Lastly, we will mainly rely on PyTorch [29] for most of the model designs and implementation, as well as for the training process.

5.2 Data Distribution

As mentioned earlier, we will split the experiment into 10 and 20 users setups. For the MNIST dataset setup, we have 14,780 samples split for 10 users, whereas we have 28,911 samples for

the 20 users setup. Our setups have an 80-20 data split, meaning we are preserving 80% of the dataset for training and 20% for testing. Furthermore, Figure 5.1 and Figure 5.2 represent the data distribution for the clients, which helps us verify that we are running a non-IID data training environment.

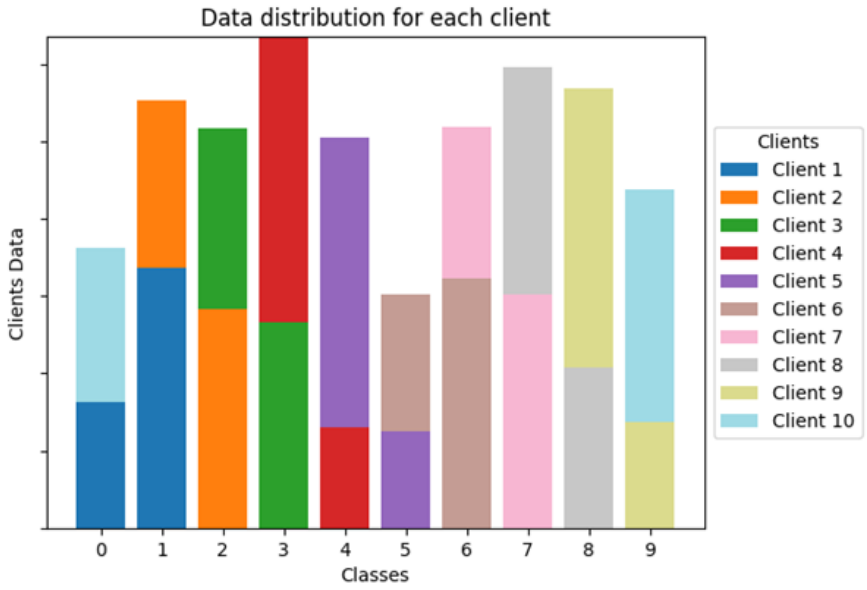

Figure 5.1: MNIST Distribution: 10 Users with 2 Labels Each.

We are not utilizing the entire dataset for MNIST on each run since the samples are chosen and distributed randomly among clients. On top of that, to ensure that we are dealing with a non-IID setup, our setup maintains that some clients will have a relatively low amount of data samples to train on since this is a scenario that may happen in a real-life federated learning environment. Furthermore, for the 10 users setup, we have 2 labels assigned to each client, whereas we have 4 labels assigned to each client in the 20 users setup.

Lastly, the synthetic dataset setup is similar to MNIST's setup. However, the main difference is that this dataset is artificially generated, where we can control how random the distribution of the generated dataset is. Furthermore, with this particular implementation, we can control additional parameters to decide whether we want a non-IID or IID setup. This can be controlled by the following parameters:

1. α: This parameter can control how different each local client model is from one another.
2. β: This parameter controls how different the local data is for each client.
3. iid_check: A simple parameter that allows us to control whether we want a non-IID distribution or not.

In order to develop a similar setup to PFedMe's, we have set $a = 0.5$, and $b = 0.5$. Furthermore, we have iid_check set to false in order to generate non-IID data. Our data distribution can be seen in Figure 5.3 and Figure 5.4, where we have 69,590 data samples for the 20 users setup and 52,795 samples for the 10 users setup.

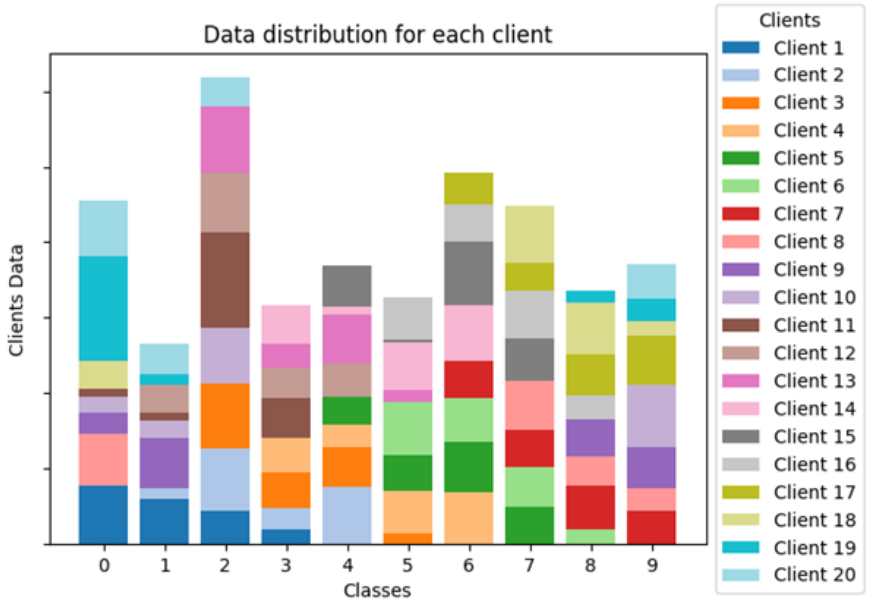

Figure 5.2: MNIST Distribution: 20 Users with 4 Labels Each.

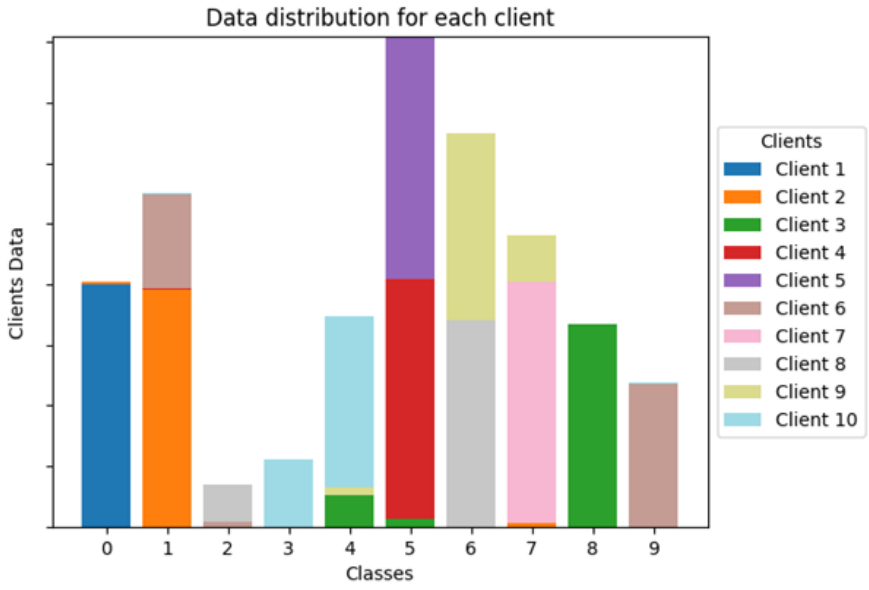

Figure 5.3: Synthetic Distribution: 10 Users, $\alpha = 0.5$, $\beta = 0.5$.

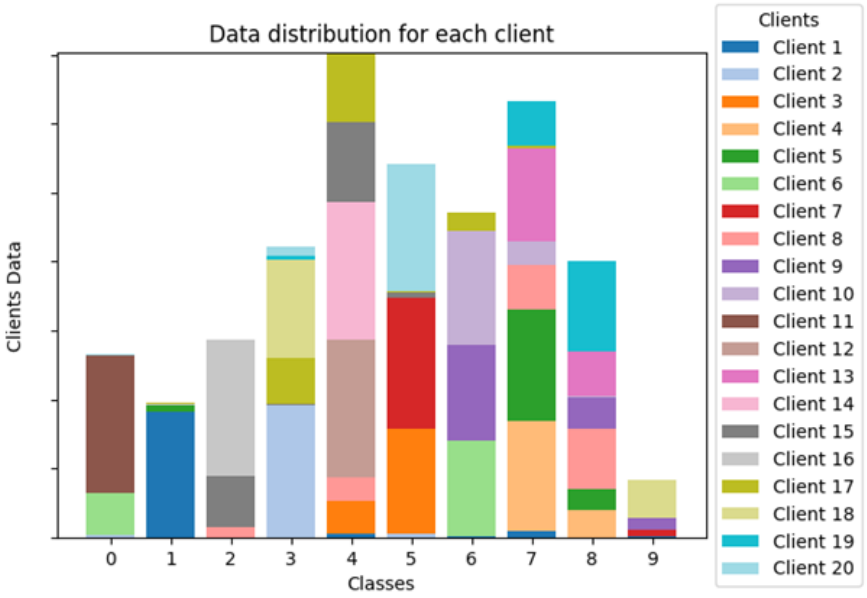

Figure 5.4: Synthetic Distribution: 20 Users, $\alpha = 0.5$, $\beta = 0.5$.

We are using this particular data generation algorithm since it has been used in various federated learning experiments, and it is also easy to use and control. Further details on how the algorithm works can be found in [30]. However, in simple terms, the data generation and distribution are done with the help of the softmax function and power law.

5.3 Running the Experiment

In order to produce the best cases for the three algorithms, we performed a grid search to find suitable learning rates, personal learning rates, personalization steps, and global communication numbers. Most learning rates that performed well were similar from one dataset and algorithm to another. Therefore, we set the personal learning rate to 0.08 and the global learning rate to 0.005 for MNIST experiments. Moreover, we had a personal learning rate of 0.02 and a global learning rate of 0.005 with the Synthetic experiments. Furthermore, we used Multi-class Logistic Regression (MCLR) for the MNIST dataset experiments and a Deep Neural Network model for the Synthetic datasets. We chose these two particular models because they have been used in PFedMe's experimentation, helping us better compare and verify our results. However, for communication rounds, we kept our global communication rounds at around 500 since anything further than that did not yield a significant performance increase. Additionally, we found that the optimal number of personalization steps is around 5 with 20 local epochs, similar to what is used in [3]. We also wanted to highlight how the primary hyperparameters introduced to PFedMe's implementations ($\lambda|\beta$) can drastically affect performance. Therefore, we will primarily highlight these hyperparameters in our results. Furthermore, there was no threshold set for stale clients in the asynchronous implementation since the clients are artificial with the same amount of computing resources, meaning they all have equal training and computation times.

First of all, our initial set of results, shown in Table 5.1 and Table 5.3, outline the performance

of our implementation, APFedMe. In the asynchronous implementation, we can see that the average accuracy levels in both client and global are close to PFedMe's performance (Table 5.2 and Table 5.4), with some scenarios where our implementation performs slightly worse than PFedMe's and vice versa. Furthermore, we have noticed that increasing the personalization parameter across different tests has yielded better overall performance. However, providing a too high value for personalization causes convergence issues (similar to PFedMe), which could pose a slight challenge in choosing an optimal value for it. In addition, choosing a good β value can be tricky as well since it is preferable to involve the global model in global updates for APFedMe, which is why we chose $\beta = 0.9$ in APFedMe. That being said, since the two implementations are similar, increasing or decreasing λ and β yields similar results for most scenarios between the two algorithms. Furthermore, we decided to mainly showcase these values for λ and β because they provided the best performance on average for our setups. Other values either caused convergence issues or simply did not enhance performance.

However, when comparing run times, we can see in the table results that APFedMe excels by a great margin. In fact, there are some cases where it was able to run around 50% and up to 67% faster compared to the other algorithm, providing a much faster convergence speed than PFedMe. Furthermore, as mentioned before, an asynchronous implementation does not require all clients to finish training in a single global round, resulting in faster speeds. Therefore, despite the fact that our implementation does not provide enhanced performance in accuracy scenarios, it still dominates the other implementation in terms of run-time. In fact, one benefit that we can gain from faster run times is the ability to implement slightly heavier models due to the time saved running an asynchronous implementation. In other words, APFedMe provides an outstanding balance between accuracy levels and convergence speed levels.

Furthermore, as previously mentioned, we have included multiple numbers of users for each implementation and dataset to gain better insight into a variety of factors, including scalability. Firstly, the run-time for both algorithms increases when the number of clients increases. This is expected since more data and users are involved in the training. Moreover, both algorithms do lose some accuracy when increasing the number of users, which is anticipated based on our experiment design and data distribution. Importantly, it is worth noting that increasing the number of users does not negatively impact APFedMe as much as it does for PFedMe's solution when comparing algorithm run times. In fact, the more users we add to the experiment, the more significant the gap between PFedMe and APFedMe run-times becomes, with our implementation running much faster.

Table 5.1: APFedMe - MNIST Dataset

Hyperparameters	**Acc - Local**	**Acc - Global**	**Global - Loss**	**Run-time**	**Number of Users**
$\lambda = 15 \vert \beta = 2$	95.22%	94.03%	0.155	64.83 Seconds	10
$\lambda = 15 \vert \beta = 2$	94.11%	92.21%	0.165	101.61 Seconds	20
$\lambda = 15 \vert \beta = 0.9$	94.77%	92.11%	0.177	66.01 Seconds	10
$\lambda = 15 \vert \beta = 0.9$	92.51%	90.90%	0.198	99.30 Seconds	20

Table 5.2: PFedMe - MNIST Dataset

Hyperparameters	**Acc - Local**	**Acc - Global**	**Global Loss**	**Run-time**	**Number of Users**
$\lambda = 15 \vert \beta = 2$	95.64%	94.79%	0.142	164.58 Seconds	10
$\lambda = 15 \vert \beta = 2$	94.50%	92.78%	0.155	210.32 Seconds	20
$\lambda = 15 \vert \beta = 1$	95.11%	92.49%	0.162	162.41 Seconds	10
$\lambda = 15 \vert \beta = 1$	92.45%	90.59%	0.192	216.44 Seconds	20

Table 5.3: APFedMe - Synthetic Dataset

Hyperparameters	**Acc - Local**	**Acc - Global**	**Global Loss**	**Run-time**	**Number of Users**
$\lambda = 20 \vert \beta = 2$	84.70%	84.34%	0.576	113.44 Seconds	10
$\lambda = 20 \vert \beta = 2$	84.33%	83.32%	0.603	223.65 Seconds	20
$\lambda = 15 \vert \beta = 2$	83.47%	82.68%	0.612	112.84 Seconds	10
$\lambda = 15 \vert \beta = 2$	83.01%	82.33%	0.654	220.14 Seconds	20

Table 5.4: PFedMe - Synthetic Dataset

Hyperparameters	**Acc - Local**	**Acc - Global**	**Global Loss**	**Run-time**	**Number of Users**
$\lambda = 20 \vert \beta = 2$	84.40%	83.88%	0.591	225.18 Seconds	10
$\lambda = 20 \vert \beta = 2$	84.11%	83.25%	0.620	330.08 Seconds	20
$\lambda = 15 \vert \beta = 2$	84.29%	83.61%	0.609	220.94 Seconds	10
$\lambda = 15 \vert \beta = 2$	83.16%	82.50%	0.635	317.77 Seconds	20

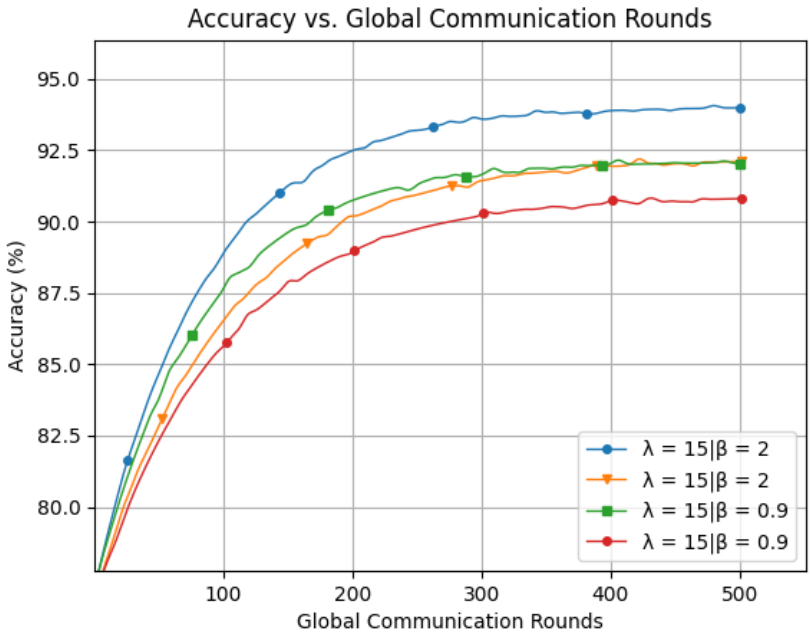

Figure 5.5: APFedMe - MNIST - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red)

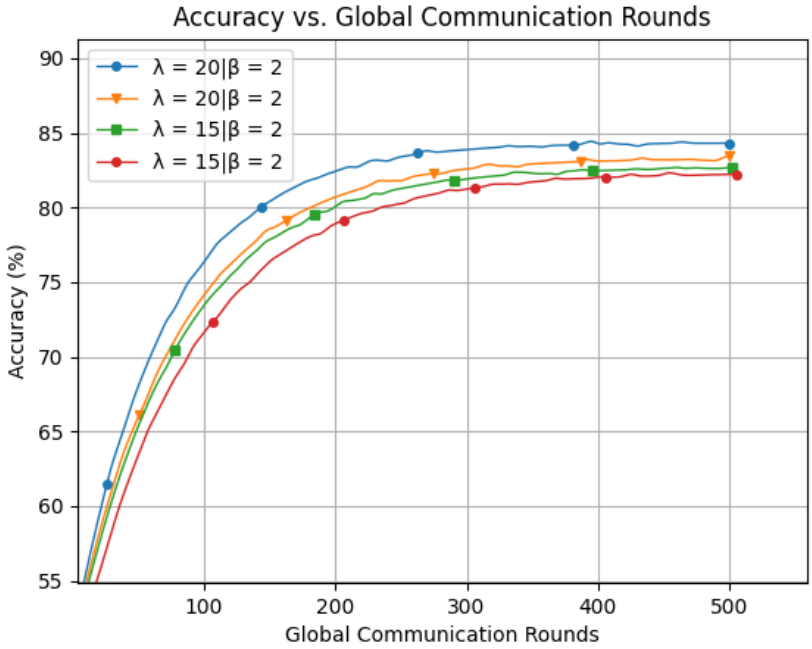

Figure 5.6: APFedMe - Synthetic - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red)

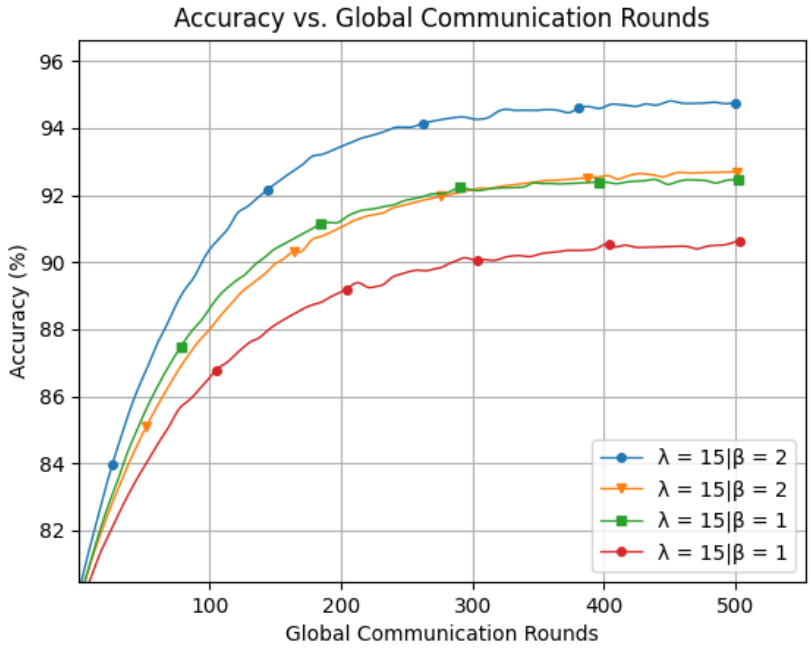

Figure 5.7: PFedMe - MNIST - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red)

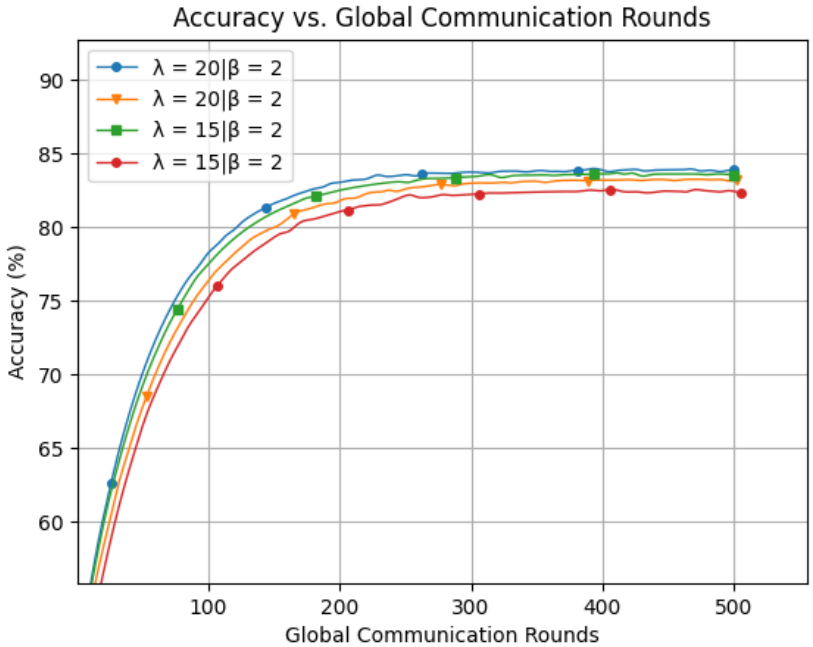

Figure 5.8: PFedMe - Synthetic - Global Accuracy - 10 Users (Blue/Green), 20 Users (Orange/Red)

5.3.1 Final Remarks

A significant advantage of APFedMe over PFedMe can be witnessed in its superior run-time performance, with some instances showing a substantial reduction in run-time. Both implementations have similar convergence rate, but in theory APFedMe has a faster convergence speed when considering the run-times of both implementations. Although APFedMe's accuracy may be slightly lower in some instances, its advantage in run-time performance can make it a more desirable option for real-world applications.

When examining scalability by increasing the number of participating users, both algorithms exhibit the expected increase in run-time and a decrease in accuracy. However, APFedMe is less affected by an increased number of users compared to PFedMe. Therefore, APFedMe offers a well-balanced solution in terms of accuracy and convergence speed, making it an appealing choice for practical implementations. We can further analyze the results by looking at the global models' accuracy shown in graphs 5.4, 5.5, 5.6, and 5.7, which verify that both implementations have relatively close results, with PFedMe performing only slightly better.

As a result of our experiments, asynchronous federated learning indeed offers a significant advantage in convergence speed and helps mitigate the substantial impact of involving a high number of participants in the learning process. Furthermore, asynchronous distributed learning has great potential for improvements and flexibility, as other researchers have proposed multiple methods to enhance asynchronous federated learning even further[31, 32].

Chapter 6

Practicality Showcase of Asynchrony in Federated Learning

In this chapter, we test the practicality of the asynchronous implementation of PFedMe in a scenario where reduced communication overhead is desirable. Specifically, we will utilize a publicly available dataset with various user setups and test their performance based on global model accuracy and loss. This will help better illustrate the advantages that asynchronous federated learning implementations can provide. Moreover, we will implement a similar model to a centralized setup and compare its results to APFedMe to gain a better insight of the performance difference.

6.1 Dataset

First of all, we will be using a publicly available dataset called IDS 2018 [33]. The University of New Brunswick originally created this dataset, and it contains a wide range of network packets. To be specific, these packets can be either benign or malicious. The malicious packets in their dataset are Denial of Service (DoS) packets. DoS is a type of network attack that comes in various forms and aims to disrupt the availability of a network service to users [34, 35]. In fact, DoS attacks is becoming more prevalent in the last few years, making it a dangerous online threat [36]. One of the primary purposes of using this type of dataset is to develop an AI-based intrusion detection system to help defend against DoS attacks.

We chose to showcase this particular dataset to illustrate an area where reduced communication overhead is needed. For instance, we will consider the following scenario: A company is hosting its service online on servers. However, one or multiple servers are being attacked using DoS packets where the current IDS cannot distinguish between malicious and legitimate packets. In order to mitigate the damages, a quick solution will be required. Therefore, as soon as an attack is detected, an asynchronous Federated Learning system can be launched across the servers to help learn and distinguish between the legitimate and malicious packets. The reduced communication overhead in asynchronous federated learning can help servers communicate together to learn faster using the packets they are receiving that are causing a DoS. Since asynchrony provides reduced communication overhead, developing a model will be much faster than in a synchronous environment, allowing the company to implement a solution much quicker, mitigating the DoS damage. Furthermore, Figure 6.1 provides a visual illustration of the scenario. In this visualization, legiti-

mate users are not able to reach the network services since servers are down due to a DoS attack, and IDS was not able to distinguish them. Therefore, once a DoS was detected, an asynchronous federated learning was initiated to learn which packets are malicious to help deal with them.

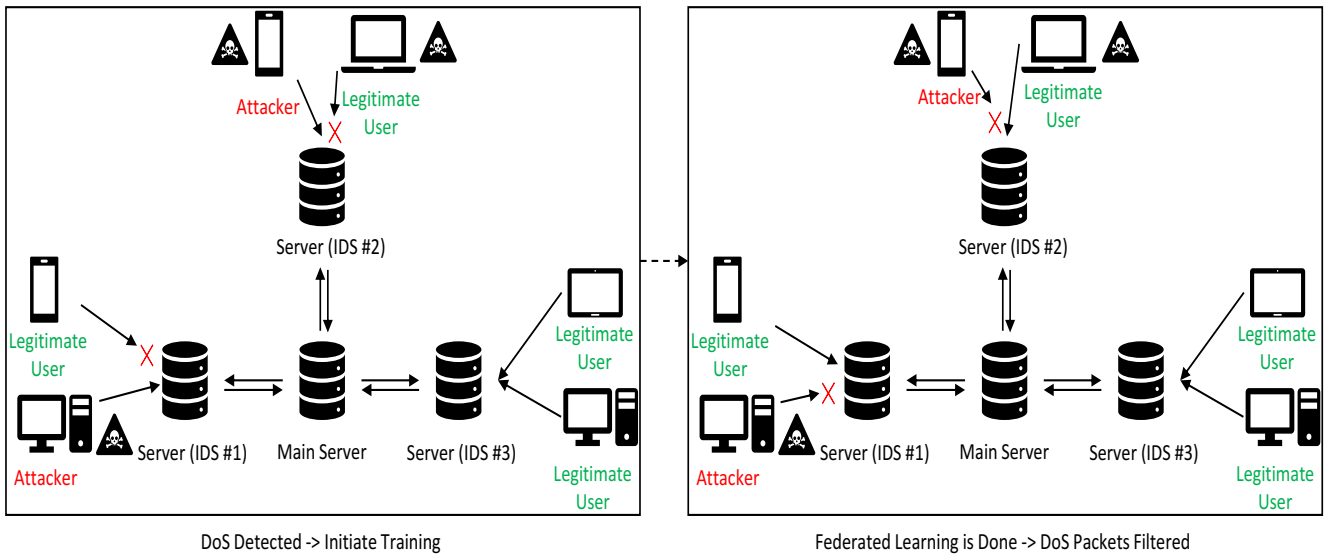

Figure 6.1: DoS Attack Mitigation Through Federated Learning

As mentioned, this dataset will be used with APFedMe to train a model. The dataset contains 1,297,249 instances, where each instance is a packet. Moreover, each packet contains 79 features (packet headers). However, we will be dropping one of the features (timestamp) since we will not need the feature to train an IDS model. In addition, each packet has a label that indicates the packet type. These labels are Benign, FTP-BruteForce, SSH-Bruteforce, DoS attacks-GoldenEye, DoS attacks-Slowloris, and DDOS attack-HOIC. However, since the main aim of this implementation is to develop an IDS, we will be replacing the labels of non-benign packets with malicious. Therefore, we have 663,808 packets with benign intentions (label=0) and 633,441 packets with malicious intentions (label = 1).

6.2 Experiment Setup

This test's setup is similar to what was done in Chapter 5. However, the main difference is that we use a different dataset with different sets of users. Specifically, to help further showcase how well the algorithm scales with different numbers of users, we will test the dataset on 5, 10, 15, 20, and 25. Moreover, following a similar setup to our previous experiment, we are using 80% of the dataset for training and 20% for testing. Furthermore, the dataset will be shuffled, then randomly distributed among the participating users. As a result, users will have different numbers of data instances, where some users will be provided with a relatively high number of samples, and others with a minimal number of instances. As mentioned in previous chapters, this type of setup helps test the algorithm in extreme scenarios where clients have extremely low data points.

The centralized setup, however, is more straightforward. We will use a similar data split, where 80% will be reserved for training, and 20% is reserved for testing. Furthermore, similarly

to the decentralized setup, we develop our model using PyTorch [29]. Furthermore, the same custom Convolutional Neural Network model will be used for both setups. Lastly, since we are dealing with a massive dataset with more than a million data points and 78 features, we had to perform additional garbage collection for both implementations and upgrade our workstation to utilize Google Colab Pro+ [28] to gain more memory.

6.3 Experiment Results

For our federated (APFedMe) experiment on the IDS dataset, we found that $\beta = 2$ and $\lambda = 15$ are suitable parameters for the training process. Furthermore, we used 400 global iterations across all setups. Lastly, we set the global learning rate to 0.005 and the local learning rate to 0.1. The centralized setup will run on a 0.001 learning rate with the Adam optimizer. After running each setup four times and averaging the results, we can now discuss the experiment results.

Firstly, it can be seen that all the decentralized setups have a global model accuracy of at least 97.11% and above. In fact, the highest model accuracy was achieved by the 5 users setup at 99.54%. Furthermore, despite the fact that global model accuracy seems to be dropping as more users are added into the training setup, the performance difference is still low nonetheless. In addition, a higher number of global rounds will likely boost the global model performance since an increased number of users with Non-IID data distribution can cause the training process to achieve desired performance at a slower pace. However, the same number of global rounds was used across all setups. The main motive behind this experiment is to better understand the convergence behavior as more users participate in the learning process.

Furthermore, the centralized setup performed slightly better than the 5 users setup at 400 global rounds, where it reached a validation accuracy of 99.97% at 40 epochs. However, this is expected since centralized setups are relatively less complex than federated learning, giving it an advantage in model training. Nonetheless, with that being said, the performance difference is not major since the difference in performance in our experiments is as low as 2%. Furthermore, this difference might be possible to reduce when using a higher number of global rounds. Therefore, considering the slight performance difference between the centralized and decentralized setups, federated learning still nonetheless has more attractive properties due to the reasons mentioned at the beginning of this thesis.

Table 6.1: Centralized & Decentralized Setups: Model Performance

Users	**Average Accuracy**	**Average Loss**
5	99.54	0.03213
10	99.29	0.05758
15	98.20	0.16118
20	97.76	0.18045
25	97.11	0.18874
Centralized	99.97	0.01801

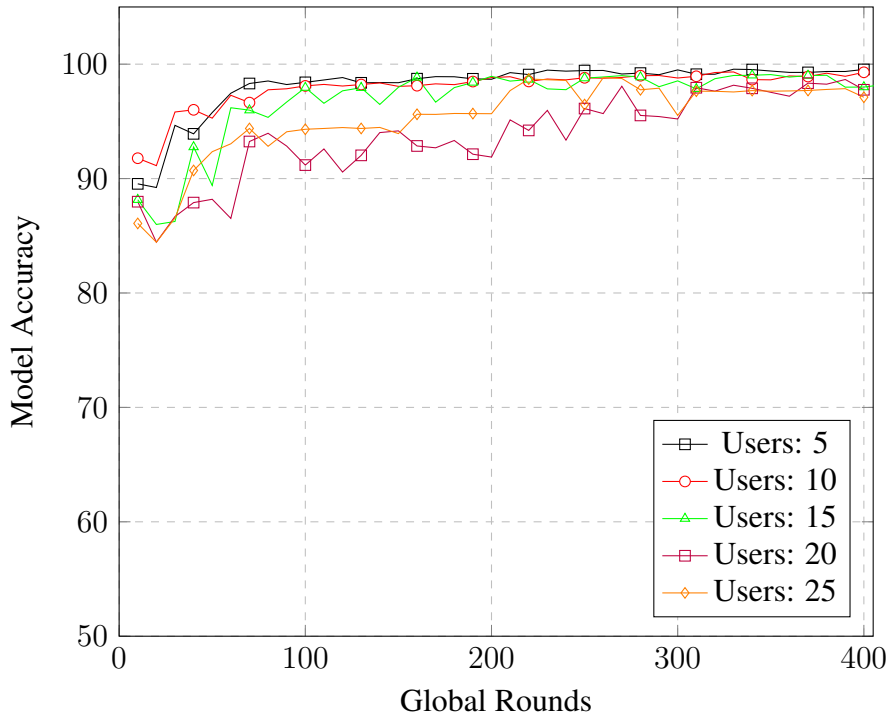

Figure 6.2: Decentralized Setup: Showcase of Users' Performance in IDS Dataset using APFedMe in 400 Global Rounds

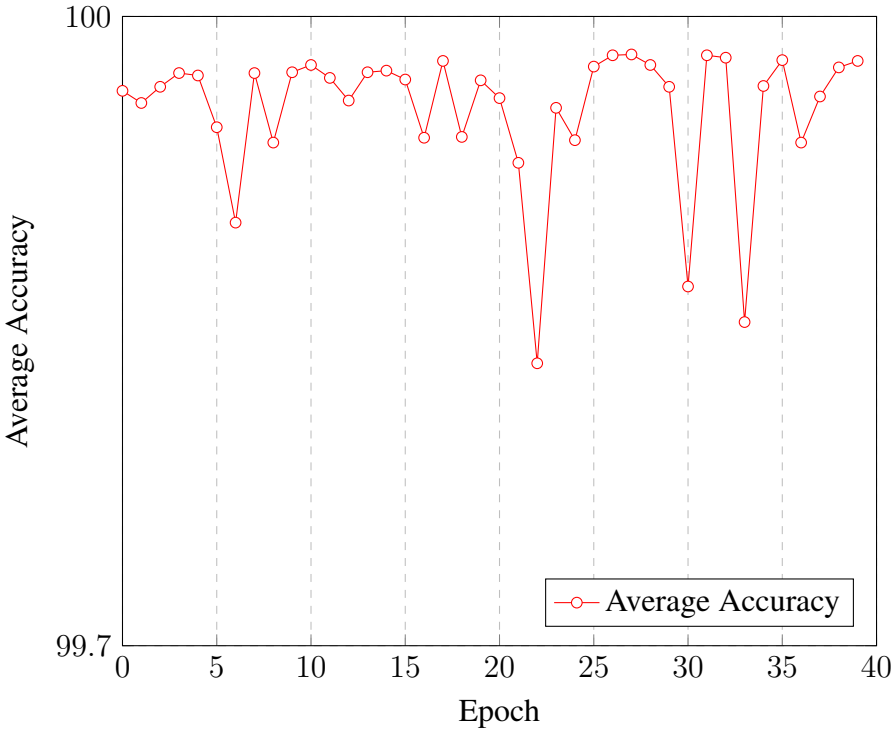

Figure 6.3: Centralized Setup: Centralized Model Performance in 40 Epochs

Chapter 7

Combating Missing Data in Local Models in the Federated Learning Framework

In this chapter, we focus on the challenges related to data in the context of federated learning. Data is a critical aspect of supervised machine learning models, and any discrepancies can significantly impact model performance and the overall learning quality. Specifically, by data issues, we refer to missing values. Therefore, this chapter will address this problem by first defining it and then providing the potential causes. Next, we will present various techniques and strategies to mitigate this challenge. Finally, we will discuss how to incorporate these solutions into a distributed learning environment to enhance the robustness of the learning process.

Addressing the missing data issue in federated learning is undoubtedly crucial. The reason behind that is that in a typical setup, we most likely do not have access to the data, nor can we manually correct them in case they get corrupted unless someone has authorized access to the data. In addition, this issue may not only harm the performance of the local clients, but may also lead to significant generalization issues for the global model, potentially causing the whole training process to fail. Therefore, addressing this problem is a necessity when developing distributed learning environments.

7.1 Causes of Missing Data

Firstly, as the name suggests, missing data in a machine learning environment refers to situations where a variable of a data point or an instance in a dataset lacks a corresponding value, making it difficult or impossible to use the data point for supervised learning tasks. There are three generally defined forms of missing data, which are: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR) [37].

The definitions for the three forms of missing data are relatively close to each other, yet they all represent different scenarios. First, the MCAR scenario refers to the probability that a missing data point is unrelated to both the observed and unobserved data. For instance, accidentally removing an answer on a questionnaire from a survey while inputting data [38]. In addition, MAR refers to the probability of a missing data point depending on the observed data but not the unobserved (missing) data. For example, in a political opinion survey, some individuals might decline to respond to a question due to their demographic characteristics. As a result, the missingness relies

on an observed variable (demographics) rather than the actual answer to the question. This scenario is typically prominent in health science studies datasets. Lastly, MNAR refers to the probability of a missing data point depending on the unobserved (missing) data itself and potentially on the observed data. For instance, in a questionnaire, participants with weaker opinions intentionally skip a specific question, leading to missing data on that question [38].

7.1.1 Causes of Missing Data in Federated Learning

Since this thesis is focused on the research area of federated learning, we will elaborate on the instances that are likely to occur during a federated learning process. In fact, there are a variety of causes that could lead to missing data in a real-life distributed learning setup, especially when dealing with IoT. Such cases can be, but are not limited to:

1. Device Failure (MCAR): Participating devices may have collected data points but did not record all corresponding values due to errors or technical issues. For example, a device collecting data for some machine learning task may have missing values because the automated labeling process failed. This failure can be related to either the device itself or the sensor it uses to gather the data. In fact, in the context of IoT, failures have become a significant risk for them because many of these devices rely on each other to function properly. This means that when one IoT device stops working, it can cause a chain reaction of unexpected and undesirable changes in other devices, leading to missing and corrupted data [39].

2. Resource Constraints (MAR): Some clients participating in a federated learning environment might have limited storage space. These days, devices can collect a large amount of data due to their fast processing speeds. In fact, processing power has become extremely fast and efficient, and it is only expected to become even faster in the future, allowing devices to collect and process data at high speeds [40]. However, even though storage capacities have improved, they are still limited nonetheless. Therefore, if not carefully monitored, storage can fill up extremely quickly, potentially causing the loss of many data points.

3. Privacy Concerns (MNAR): We mainly deal with sensitive user data in a typical federated learning environment. Therefore, some devices participating in the training process may choose not to share all data due to privacy concerns. For instance, users may be hesitant and refuse to provide personal information, such as age, income, or even gender, which could be necessary for labeling specific data points.

Although the aforementioned points do not cover all the causes that can lead to missing data, they are still some of the most common reasons when dealing with real-life setups. With that being said, understanding the causes and the reasons behind this issue is one of the first necessary steps to take in order to mitigate this problem.

7.2 Dealing with Missing Data

Many studies have explored potential ways to deal with missing data. In fact, a variety of these researches are commonly used these days and further improved upon. However, despite the fact

that these methods do not guarantee a proper estimate for the missing value, they nonetheless help recover the values with a reasonable estimation. Therefore, we will introduce various methods to deal with this challenge, then help improve and integrate them into a distributed learning environment.

Data imputation is the standard method for mitigating the problem of missing data. This method generally recovers values by estimating missing values using already-available data. Although data imputation does not guarantee accurate estimations for the missing instances, it may still be helpful since we are estimating the missing data by gathering information from the dataset itself, providing a higher probability of a proper estimation for the actual value. In fact, there are a variety of methods to impute data. However, the four most common data imputation methods are:

1. Mean imputation: This method refers to estimating the missing data by replacing missing values with the mean of the available data. Therefore, following the scenario for Table 7.1, we are able to estimate Eve's Math score as 84, as shown below. However, this method may be more suitable for continuous data.

$$\text{Mean} = \frac{1}{4}(85 + 84 + 92 + 75) = \frac{1}{4}(336) = 84$$

2. Conditional mean imputation: This type of imputation is relatively similar to normal mean imputation. However, the main difference is that the mean calculation is slightly different. Specifically, we will rely on a condition that will be enforced when calculating the mean value. For instance, if we want to estimate Eve's Math score in Table 7.1, we can do so by specifying a condition that will allow us to calculate a more reasonable mean value to use. We can calculate the mean Math score for students who scored similarly to Eve in Science (94 in this case). Therefore, we can estimate Eve's Math score as shown in the following:

$$\text{Conditional Mean (Math score — Science score = 94)} = \frac{85 + 84}{2} = 84.5$$

3. Median imputation: This data imputation method involves replacing missing values with the median. For instance, let us again consider Table 7.1, but with Eve's missing Math score being estimated using median imputation. As a result, we get the missing Math score to be 84.5, as shown below. Furthermore, we may use this method to lower extreme values sensitivity.

$$\text{Sorted: } 75, 84, 85, 92 \Rightarrow \text{Median} = \frac{84 + 85}{2} = 84.5.$$

4. Mode imputation: This type of data imputation refers to replacing missing values with the most frequent value. In fact, this method is more suitable for categorical data. To better illustrate mode data imputation, we will refer to the gender column from Table 7.1, where Eve's gender field is missing. By analyzing the Gender column, we can calculate the most common gender in the dataset, which is Female in this case. Therefore, we can estimate Eve's Gender field as Female.

Table 7.1: Artificial Dataset - Recovery Through Data Imputation

Name	Math Score	Science Score	Gender
Alice	85	94	Female
Andrew	84	94	Female
Siesta	92	82	Female
Annie	75	90	Female
Eve	NaN	94	Female

An additional typical method to deal with dataset missing data is instances removal. As the name implies, this process removes data instances where the data values are unavailable. For instance, consider Table 7.1 as an example, where we will remove the row with missing data entirely from the dataset. Specifically, we will eliminate the name Eve, along with their scores and gender, from the dataset since they are missing at least one tag (in this case, both tags are missing). However, with that being said, deleting instances may cause an imbalance in the dataset and introduce bias [41], especially if we are applying this method in MNAR and MAR scenarios. Therefore, using this method as the last step of data estimation and recovery is wiser.

Another two common methods that can be used to help deal with missing data and mitigate their adverse effects are Last Observation Carried Forward (LOCF) and Worst Observation Carried Forward (WOCF). In fact, these two methods are sometimes referred to as single imputation methods.

Firstly, LOCF refers to taking the last available data point and copying it into the missing data value to estimate the missing data. LOCF is often used in longitudinal or time series data when a measurement is missing at a particular point in the dataset. To further comprehend the idea behind LOCF, we can refer to Table 7.2, which represents an artificial dataset with patients' pain levels across different months. In order to follow the principle behind LOCF, Patient A's pain value in March will be taken from the previous month, which is 6 since it is the last available observation. In addition, Patient B's missing value in May will be 4 since their pain level was 4 in the previous month.

Table 7.2: Dataset - Recovery Through LOCF and WOCF

Week	Patient A - Pain	Patient B - Pain
January	6	2
February	5	2
March	NaN	1
April	3	4
May	7	NaN

On the other hand, WOCF, which has a similar concept to LOCF, refers to taking the previous worst observation found at the time of recovering the data rather than taking the last available observation. However, the worst observation may differ from one context or dataset to another. For better illustration, let us consider Table 7.2 once again. To estimate the patients' value, we will

search for the highest pain level (worst) that occurred before the missing data point. For Patient A's pain level, their pain level in March will be replaced with 6, which is from January. In addition, Patient B's pain level in May will be replaced with 4, a similar value to LOCF. In other words, Patient A and B's pain levels were estimated by copying their worst pain level previously reported (from March and May, respectively). However, with that being said, it is worth noting that some studies show that utilizing LOCF in MCAR scenarios may not yield relevant results [42].

The above methods provide a relatively simple implementation with reasonable performance when estimating missing values, but with some limitations. Moreover, the mentioned data estimation methods work best when analyzing the type of missing data you are dealing with. Therefore, it is crucial to be aware of the type of environment you are dealing with during implementation.

7.2.1 Dealing with Missing Data in Federated Learning

Handling missing data in a federated learning environment may differ from handling them in a typical machine learning setup. As previously mentioned, it is not expected that we will have access to any of the data on local clients, making it difficult for us to implement the previously mentioned methods directly. In addition, not all data estimation methods that work well in a centralized learning setup will perform as well in a distributed learning setup. Therefore, this section will address applicable methods to utilize and mitigate the problem of missing data in a distributed learning setup to help further understand how one can implement them in real-life scenarios. There are many proposed methods where that involve sharing some sort of data among clients. However, it is only sometimes the case that we have authorized access to clients' data. In fact, one of the essential purposes of federated learning is to maintain user confidentiality. Therefore, we will not be discussing those implementations in this thesis.

One imputation method that is used in centralized machine learning that can also be used in federated learning is Multiple Imputation by Chained Equations (MICE) [43]. Figure 7.1 illustrates a visualization of how the algorithm works for a dataset with missing data in 3 different variables. MICE is a popular adaptation of Multiple Imputation (MI) [44]. The implementation starts by defining the number of iterations, which will be discussed later as to why it is needed. After that, we take each variable that contains missing data and calculate a placeholder for them. Then, we can replace them with the average value of the observed numbers for that particular variable for the time being. Afterward, we can implement an analysis method, like a regression. This analysis method will be used to recover the missing values of the current variable with the other observed variables. Once the missing values are estimated, we replace them with their placeholder (the average value in this case) and rerun the same steps. On each iteration, we are replacing the previously found values with the new regressed ones. In addition, it is worth noting that we will use the latest regressed values for each new iteration. Typically, this method is repeated 5 to 10 times, depending on the iterator variable we define at the beginning [45]. Once the loop ends, we repeat the same steps for the next variable. Usually, you can start estimating missing data for variables with the least missing instances. The final output will be a dataset with estimated values of the last iteration of each variable. Therefore, we can outline MICE implementation as follows:

1. Replace every missing data with the mean of the observed values for the variable, which acts as a placeholder.

2. Regress the observed values of a variable with missing data on the other variables in the dataset. Start with variables with the least missing data.

3. Replace the missing values with the predictions derived from the regression model.

4. Imputed values will then be part of the dataset and will be included in the next regression.

5. Repeat steps 2-4 for each variable with missing data for approximately 5 to 10 rounds.

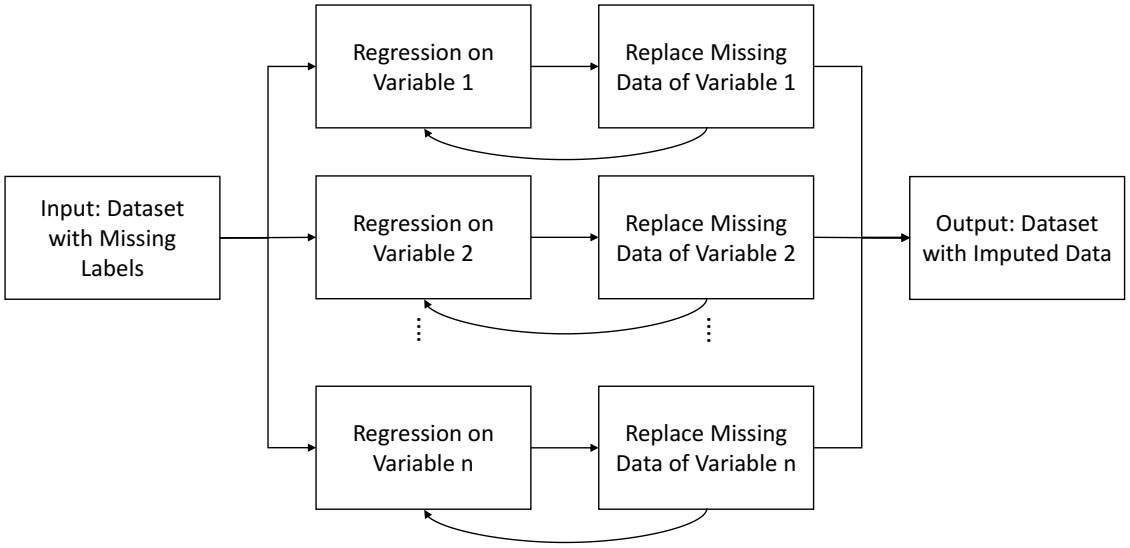

Figure 7.1: MICE Visual Outline for a Dataset with 3 Variables Containing Missing Data

MICE is known to be an effective estimation method of missing data [46, 43]. It can be implemented in federated learning by applying it locally to clients with missing data. However, one limitation of this algorithm is that it does not work well for high-dimensional data [47]. Therefore, we can either refer to other algorithms that support multidimensional data or utilize algorithms for data transformation to reduce dimensionality and adapt the data to MICE [48]. In addition, Principal Component Analysis (PCA) [49] can also be used to perform dimensionality reduction. PCA performs dimensionality reduction with low error/variance when reconstructing the compressed data. With that being said, dimensionality reduction algorithms may cause convergence issues in federated learning settings since undoing the dimensionality reduction may not yield the original data. Furthermore, the type of regression used and the number of iterations can drastically slow down the run-time algorithm. Therefore, these implementations require analysis before considering them.

Another well-performing algorithm that is used to impute data in centralized setups [50, 51] and can also be utilized in federated learning is K-Nearest Neighbors (KNN) [52]. In fact, KNN is a popular machine learning method that can also be used to impute missing data. What makes KNN extremely useful is that it can operate with high-dimensional data, allowing it to be usable in many scenarios. The way KNN works is fairly simple. First of all, we will have to determine a variable, K. The reason why we need K will be discussed later on, but it should be noted that a low value of K provides a ”localized” imputation, while a higher value can provide smoother imputation with

less noise. However, once that is done, we will convert our dataset into a multidimensional feature space. The reason for converting our data into multidimensional space is to calculate a distance metric, like Euclidean, Manhattan, and Minkowski distances [53]. The choice of distance metric depends on the nature of the data and the desired outcome, which is why KNN may also require some additional analysis before implementing it. However, with that being said, once we convert our data and define a variable K, we can now start the imputation process. Imputing missing values in KNN starts by first placing the data point with the missing data into the feature space. The data point will be placed based on its current known features (other variables). Afterward, we will calculate the distance between the missing data point and all the other points in the feature space to find a k number of nearest neighbors (Figure 7.2 illustrates an example of imputed data point from 3 nearest neighbors using mode imputation). Once we find the k nearest neighbors to the missing data point, we can impute the missing data based on the neighbors' mean, median, mode, or whatever method applies to the dataset. Once we iterate over all the missing points, we will get an output with a dataset with all missing data points replaced with the values based on their nearest neighbors. Therefore, we can outline the KNN algorithm as follows:

1. Define K to set the number of nearest neighbors to consider.

2. Transform data into multidimensional feature space and choose a feature distance measurement appropriate for your application.

3. Calculate the distance between the missing data point (based on non-missing features), and all other data points.

4. Identify the K nearest neighbors to the missing data point.

5. Calculate either mean, median, or mode from the value of nearest neighbors depending on what is applicable to the dataset.

6. Replace the missing data with the calculated value from the previous step.

7. Iterate over all missing data points and repeat steps 3-6 until all missing data are imputed.

Implementing KNN directly into local clients has shown good performance when imputing data in federated learning setups [52]. In addition, its support for high-dimensional data makes it desirable to utilize. However, it is worth noting that KNN can be a slow algorithm since distance calculation is intensive, but some enhancements can be implemented proposed in other studies. In addition, similarly to many other imputation methods, careful analysis is required to determine the distance metrics and imputation methods to use.

Even though we have only mentioned two methods that can handle missing data in the context of federated learning, there are still plenty of other methods. However, these two methods have shown to be working with reasonable performance [52], and they can operate in various scenarios. In addition, if the slowness of KNN might be an issue and enhancement methods did not work, we can still deal with high-dimensionality data by incorporating an algorithm, like Expectation-Maximization with PCA [54].

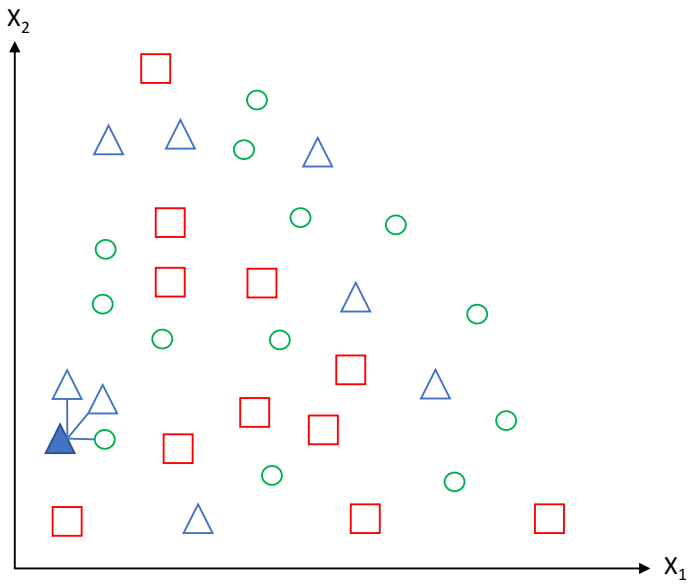

Figure 7.2: KNN - Nearest Neighbors (Mode Emputation) Example (k=3)

7.3 Evaluating Imputation Methods for Distributed Learning

In this section, we will discuss the testing of imputation methods, namely KNN and MICE, to evaluate their performance in a distributed learning setup. Our goal is to assess how these methods affect the global model accuracy and run-time while handling missing labels.

7.3.1 Experiment Design

To gain a better insight of how well the aforementioned algorithms will perform in a distributed learning setup, we will put them to the test. Specifically, we will use the MNIST dataset mentioned in our APFedMe experiment to test the KNN and MICE implementations by randomly deleting labels off data points in the dataset, then impute the missing labels to use them to train the model. We implemented a method to mimic MCAR that deletes a specified percentage from the users' data. In addition, for our imputation experiment, we will run 10 different setups. Each setup will have different percentage of missing labels but the same APFedMe hyperparameters (Global Rounds = 300, $\lambda = 15$, $\beta = 2$). Each setup will have the following percentage of missing labels: 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, and 50%. Furthermore, we ran a grid search for both imputation methods to choose proper imputing hyperparameters. For the KNN implementation, we will consider the three nearest neighbors of a missing data point across all setups, and the take the mode of these neighbours. On the other hand, we will use 7 imputation iterations for MICE for all the setups. Lastly, we ran each setup three times and averaged the results.

7.3.2 Test Results

The first imputation test was done using KNN. As shown in Table 7.3, imputing missing labels using KNN shows fair performance, where APFedMe's global model accuracy barely drops as more instances are deleted. In fact, at 50% label dropout, the global model accuracy has only

gone down by around 4% (from 91.30% to 87.52%), indicating that KNN is imputing labels with relatively fair accuracy. However, KNN's run-time is heavily affected by the number of missing labels in the dataset. As seen in Table 7.3, the algorithm run-time at 5% dropout is only 1.21 seconds, and goes up to 11.36 seconds at 50% dropout. The reason behind this is that the more missing data points there are in the dataset, the more distance calculations will be done, which can be an expensive task. Therefore, a lower number of missing data points in a dataset can make this implementation more appealing to use.

The second imputation test results are shown in Table 7.4. MICE performance is quite robust when considering the minimal performance drop in each setup. In fact, similarly to KNN, MICE has almost dropped only 4% in global model performance (from 90.58% to 86.93%) at 50% label dropout. However, with that being said, MICE's run-time is much higher than KNN. One of the main reasons behind this is because we used 7 imputation iterations, which can be computationally expensive. Reducing the number of iterations has shown much faster run-times, but we chose 7 iterations since lower numbers did not always yield good accuracy. Furthermore, the run-time can be reduced by using less complex regression models. With that being said, an interesting perk of MICE is that its run-time goes down as the number of missing data points increases. This is because the more missing labels there are in a dataset, the fewer data points will be used to develop the regression model, reducing the overall imputation run-time.

Table 7.3: KNN Imputation Results

Percentage	**Run-time**	**Accuracy**
5%	1.21 Seconds	91.30%
10%	2.18 Seconds	90.97%
15%	3.83 Seconds	90.54%
20%	4.29 Seconds	89.94%
25%	5.43 Seconds	90.11%
30%	6.41 Seconds	89.19%
35%	7.73 Seconds	89.11%
40%	9.00 Seconds	88.39%
45%	9.85 Seconds	87.34%
50%	11.36 Seconds	87.52%

7.3.3 Final Remarks

Both implementations show comparable numbers in terms of global model accuracy as shown in Figure 7.3. The slight drop in performance when increasing the dropout percentage is a common trend, as higher amounts of missing data can lead to less accurate imputations. The choice between these methods may depend on the specific percentage of missing data and the desired balance between run-time and accuracy. Furthermore, showcasing the behaviour of these two implementations should enable a better understanding of the strengths and weaknesses of them, which can potentially further enhance the practicality when implementing a federated learning setup.

Table 7.4: MICE Imputation Results

Percentage	Run-time	Accuracy
5%	22.37 Seconds	90.58%
10%	23.79 Seconds	90.66%
15%	22.54 Seconds	90.32%
20%	22.18 Seconds	90.16%
25%	20.86 Seconds	89.38%
30%	20.78 Seconds	89.04%
35%	20.38 Seconds	88.52%
40%	20.18 Seconds	88.48%
45%	18.38 Seconds	87.84%
50%	18.56 Seconds	86.93%

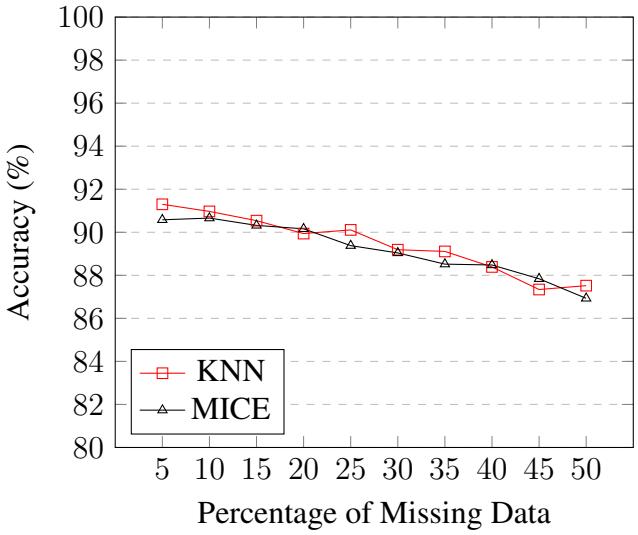

Figure 7.3: Comparison of KNN and MICE Imputation Methods

Chapter 8

Conclusion

In conclusion, this research delved into various aspects of federated learning, tackling key challenges such as data heterogeneity, missing data, and communication bottlenecks. Additionally, a comprehensive overview of related work in federated learning was provided, focusing on algorithms that have attempted to address these challenges. These studies have contributed to developing more effective federated learning approaches and laid the foundation for the asynchronous federated learning algorithm, APFedMe.

Federated learning has demonstrated substantial potential across various sectors. For example, applications in healthcare, finance, intelligent vehicles, and IoT devices also showcase the value of federated learning in addressing data privacy, model generalization challenges, and cross-institution collaboration in machine learning. With ongoing research in this area, federated learning will play an increasingly significant role in the future of AI and machine learning solutions.

The primary contribution of this thesis is the development and evaluation of APFedMe, an enhanced version of the PFedMe algorithm. PFedMe is a personalized federated learning approach using Moreau envelopes that addresses personalization in distributed environments while maintaining relatively good efficiency in the implementation. PFedMe employs an iterative update method and a bi-level solution for optimizing local models while maintaining a reasonable distance from the global model. To improve this, APFedMe, an asynchronous version of PFedMe, was introduced to address communication bottlenecks in synchronous federated learning setups. APFedMe refines the solution primarily on the global server by reducing reliance on weight averaging and improving efficiency as well as scalability by eliminating the need for clients to wait for others before sending updates.

Moreover, a comparison between APFedMe and PFedMe was presented regarding global model accuracy, loss values, and training run times using benchmark datasets (Cifar 10 and Synthetic). The experiments showed that asynchronous federated learning provided significant benefits in convergence speed and helped alleviate the impact of involving participants with relatively slow learning and communication speeds in the process. Asynchronous distributed learning holds immense potential for enhancements and flexibility, with other researchers proposing various methods to improve asynchronous approaches further.

Additionally, the integration of missing data techniques into federated learning was examined. The discussion focused on how each participating client could apply imputation methods at the local level to address missing data issues while adhering to the privacy-preserving principles of federated learning. Some methods, such as MICE and KNN, have shown promising results when

applied to federated learning. However, assessing each method's applicability and potential issues is crucial before applying them to a specific federated learning problem.

Future work in asynchronous federated learning, explicitly targeting the APFedMe implementation, can focus on addressing several key challenges associated with client selection, stale clients, and improving global model updates. Specifically, the research scope can investigate the following areas:

Future work in asynchronous federated learning, explicitly targeting the APFedMe implementation, can focus on addressing several key challenges associated with client selection, stale clients, and improving global model updates. Specifically, the research scope can investigate the following areas:

1. Developing more effective node selection: Optimizing client participation based on several factors, such as data quality, availability, and communication latency, researchers can design algorithms that enhance the overall performance of the asynchronous federated learning system. This would involve creating adaptive selection mechanisms that prioritize clients contributing the most relevant and timely data. In fact, this can also include dropping out clients who are not providing a good contribution to the global model.

2. Investigating new techniques to handle stale clients: Stale clients that provide outdated or slow updates can negatively impact global model performance. This is a common issue found in asynchronous federated learning implementations, and mitigating it should provide significant enhancements to the learning process. Specifically, exploring innovative approaches to either incorporate their updates more effectively or minimize their influence on the global model can help maintain the accuracy and efficiency of the asynchronous federated learning system.

3. Exploring advanced model aggregation methods: In the context of APFedMe, we are reducing inconsistency through the β variable. However, it is wiser to account for varying factors related to clients when aggregating their local models. By developing and implementing more advanced aggregation techniques that weigh clients' contributions better, researchers can enhance the global model updates and improve the overall accuracy of the asynchronous federated learning system.

By addressing these challenges, the efficiency and performance of the APFedMe algorithm can be further refined, unlocking the potential of asynchronous federated learning in various real-world applications.

Furthermore, in this thesis, we have presented a novel approach without delving into the convergence analysis of the proposed implementation. As an essential aspect of any machine learning method, convergence analysis helps in understanding the stability and reliability of the algorithm in reaching optimal solutions. Thus, a direction for future work would be to provide a convergence analysis for the method proposed in this thesis. This analysis could involve evaluating the rate of convergence, the impact of communication delays and increased users, as well as the influence of straggling nodes on the overall learning process. Additionally, the study of convergence guarantees under various conditions, such as non-convex loss functions and non-IID data distributions, will further bolster the applicability and robustness of the proposed algorithm in real-world scenarios.

In summary, this thesis contributed to the understanding and developing of federated learning algorithms, particularly in the context of asynchronous learning and handling missing data. It opens up avenues for future research and the potential for further enhancements to federated learning systems' efficiency and overall performance. The findings and advancements presented in this study demonstrate the potential of federated learning to revolutionize the way machine learning models are developed and deployed while maintaining data privacy and security.

Bibliography

[1] C. E. A. Zaouiat and A. Latif, "Internet of things and machine learning convergence: The e-healthcare revolution," in *Proceedings of the 2nd International Conference on Computing and Wireless Communication Systems*, ser. ICCWCS'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3167486.3167551

[2] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A Review of Applications in Federated Learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220305532

[3] C. T. Dinh, N. H. Tran, and T. D. Nguyen, "Personalized Federated Learning With Moreau Envelopes," 2020. [Online]. Available: https://arxiv.org/abs/2006.08848

[4] X. Ma, J. Zhu, Z. Lin, S. Chen, and Y. Qin, "A state-of-the-art survey on solving non-iid data in federated learning," *Future Generation Computer Systems*, vol. 135, pp. 244–258, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22001686

[5] D. Li and J. Wang, "FedMD: Heterogenous Federated Learning via Model Distillation," *CoRR*, vol. abs/1910.03581, 2019. [Online]. Available: http://arxiv.org/abs/1910.03581

[6] A. L'Heureux, K. Grolinger, H. El Yamany, and M. Capretz, "Machine Learning With Big Data: Challenges and Approaches," *IEEE Access*, vol. 5, pp. 7776–7797, Apr 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7906512

[7] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On Challenges in Machine Learning Model Management," *IEEE Data Engineering Bulletin*, 2015. [Online]. Available: https://www.amazon.science/publications/on-challenges-in-machine-learning-model-management

[8] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and S. Avestimehr, "Federated Learning for Internet of Things: Applications, Challenges, and Opportunities," *CoRR*, vol. abs/2111.07494, 2021. [Online]. Available: https://arxiv.org/abs/2111.07494

[9] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation," in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, and T. van Walsum, Eds., vol. abs/1810.04304. Cham: Springer International Publishing, 2019, pp. 92–104. [Online]. Available: http://arxiv.org/abs/1810.04304

[10] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai *et al.*, "Federated Learning for Predicting Clinical Outcomes in Patients with COVID-19," *Nature Medicine*, vol. 27, pp. 1735–1743, 2021. [Online]. Available: https://doi.org/10.1038/s41591-021-01506-3

[11] N. Surantha, P. Atmaja, David, and M. Wicaksono, "A review of wearable internet-of-things device for healthcare," *Procedia Computer Science*, vol. 179, pp. 936–943, 2021, 5th International Conference on Computer Science and Computational Intelligence 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921001149

[12] G. Xu, "Iot-assisted ecg monitoring framework with secure data transmission for health care applications," *IEEE Access*, vol. 8, pp. 74 586–74 594, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9068222

[13] G. Long, Y. Tan, J. Jiang, and C. Zhang, *Federated Learning for Open Banking*. Cham: Springer International Publishing, 2020, pp. 240–254. [Online]. Available: https://doi.org/10.1007/978-3-030-63076-8_17

[14] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "FFD: A Federated Learning Based Method for Credit Card Fraud Detection," in *Big Data – BigData 2019*, K. Chen, S. Seshadri, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 18–32. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-23551-2_2

[15] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *CoRR*, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813

[16] S. A. Beiker, "Legal aspects of autonomous driving," *Santa Clara L. Rev.*, vol. 52, p. 1145, 2012. [Online]. Available: https://heinonline.org/hol-cgi-bin/get_pdf.cgi?handle=hein.journals/saclr52&section=36

[17] M. Alawadhi, J. Almazrouie, M. Kamil, and K. A. Khalil, "A systematic literature review of the factors influencing the adoption of autonomous driving," *International Journal of System Assurance Engineering and Management*, vol. 11, pp. 1065–1082, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s13198-020-00961-4

[18] A. Nguyen, T. Do, M. Tran, B. X. Nguyen, C. Duong, T. Phan, E. Tjiputra, and Q. D. Tran, "Deep Federated Learning for Autonomous Driving," *CoRR*, vol. abs/2110.05754, 2021. [Online]. Available: https://arxiv.org/abs/2110.05754

[19] J.-P. Giacalone, L. Bourgeois, and A. Ancora, "Challenges in aggregation of heterogeneous sensors for autonomous driving systems," in *2019 IEEE Sensors Applications Symposium (SAS)*, 2019, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8706005

[20] M. Anderson, "Technology Device Ownership: 2015," May 2020, accessed on April 28, 2023. [Online]. Available: https://www.pewresearch.org/internet/2015/10/29/technology-device-ownership-2015/

[21] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," *CoRR*, vol. abs/1811.03604, 2018. [Online]. Available: http://arxiv.org/abs/1811.03604

[22] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human Activity Recognition Using Federated Learning," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018, pp. 1103–1111. [Online]. Available: https://ieeexplore.ieee.org/document/8672262

[23] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated Learning of Deep Networks using Model Averaging," *CoRR*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[24] A. Fallah, A. Mokhtari, and A. E. Ozdaglar, "Personalized Federated Learning: A Meta-Learning Approach," *CoRR*, vol. abs/2002.07948, 2020. [Online]. Available: https://arxiv.org/abs/2002.07948

[25] Y. Chen, Y. Ning, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices," *CoRR*, vol. abs/1911.02134, 2019. [Online]. Available: http://arxiv.org/abs/1911.02134

[26] X. Jin, P. Luo, F. Zhuang, J. He, and Q. He, "Collaborating Between Local and Global Learning for Distributed Online Multiple Tasks," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 113–122. [Online]. Available: https://doi.org/10.1145/2806416.2806553

[27] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6296535

[28] E. Bisong, *Google Colaboratory*. Berkeley, CA: Apress, 2019, pp. 59–64. [Online]. Available: https://doi.org/10.1007/978-1-4842-4470-8_7

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," 2019. [Online]. Available: http://arxiv.org/abs/1912.01703

[30] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the Convergence of Federated Optimization in Heterogeneous Networks," 2018. [Online]. Available: http://arxiv.org/abs/1812.06127

[31] "Asynchronous Federated Learning on Heterogeneous Devices: A Survey," 2021. [Online]. Available: https://arxiv.org/abs/2109.04269

[32] J. Xu, L. Shi, Y. Shi, C. Fang, and J. Xu, "An asynchronous federated learning optimization scheme based on model partition," in *Wireless Algorithms, Systems, and Applications*, L. Wang, M. Segal, J. Chen, and T. Qiu, Eds. Cham: Springer Nature Switzerland, 2022, pp. 367–379. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-19211-1_31

[33] U. of New Brunswick, "IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018), Version: 1," Retrieved 2023-05-12 from https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv, 02 2018.

[34] G. Carl, G. Kesidis, R. Brooks, and S. Rai, "Denial-of-service attack-detection techniques," *IEEE Internet Computing*, vol. 10, no. 1, pp. 82–89, Jan 2006. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1580418

[35] J. K. Chahal, A. Bhandari, and S. Behal, "Distributed denial of service attacks: A threat or challenge," *New Review of Information Networking*, vol. 24, no. 1, pp. 31–103, 2019. [Online]. Available: https://doi.org/10.1080/13614576.2019.1611468

[36] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang, "A survey of distributed denial-of-service attack, prevention, and mitigation techniques," *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. 1550147717741463, 2017. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/1550147717741463

[37] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A Survey on Missing Data in Machine Learning," *Journal of Big Data*, vol. 8, no. 1, p. 140, 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00516-9

[38] M.-P. Fernando, F. Cèsar, N. David, and H.-O. José, "Missing the Missing Values: the Ugly Duckling of Fairness in Machine Learning," *International Journal of Intelligent Systems*, vol. 36, no. 7, pp. 3217–3258, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.22415

[39] L. Xing, "Cascading Failures in Internet of Things: Review and Perspectives on Reliability and Resilience," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 44–64, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9174628

[40] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 64–76. [Online]. Available: https://ieeexplore.ieee.org/document/7446054

[41] J. Dziura, L. Post, Q. A. Zhao, Z. Fu, and P. Peduzzi, "Strategies for Dealing With Missing Data in Clinical Trials: From Design to Analysis," *The Yale journal of biology and medicine*, vol. 86, no. 3, pp. 343–358, Sep 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3767219/

[42] R. Joseph, J. Sim, R. Ogollah, and M. Lewis, "A Systematic Review Finds Variable Use of the Intention-To-Treat Principle in Musculoskeletal Randomized Controlled Trials with Missing Data," *Journal of Clinical Epidemiology*, vol. 68, no. 1, pp. 15–24, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895435614003461

[43] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple Imputation by Chained Equations: What Is It and How Does It Work?" *International Journal of Methods in Psychiatric Research*, vol. 20, no. 1, pp. 40–49, Mar 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/mpr.329

[44] P. Li, E. A. Stuart, and D. B. Allison, "Multiple Imputation: A Flexible Tool for Handling Missing Data," *JAMA*, vol. 314, no. 18, pp. 1966–1967, Nov 2015. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/26547468/

[45] J. Wulff and L. Ejlskov, "Multiple Imputation By Chained Equations in Praxis: Guidelines and Review," *Electronic Journal of Business Research Methods*, vol. 15, pp. 2017–2058, Apr 2017. [Online]. Available: https://academic-publishing.org/index.php/ejbrm/article/view/1355

[46] J. N. Wulff and L. E. Jeppesen, "Multiple imputation by chained equations in praxis: guidelines and review," *Electronic Journal of Business Research Methods*, vol. 15, no. 1, pp. 41–56, 2017. [Online]. Available: https://vbn.aau.dk/ws/files/257318283/ejbrm_volume15_issue1_article450.pdf

[47] Y. Deng, C. Chang, M. S. Ido, and Q. Long, "Multiple imputation for general missing data patterns in the presence of high-dimensional data," *Scientific reports*, vol. 6, no. 1, p. 21689, 2016. [Online]. Available: https://www.nature.com/articles/srep21689

[48] D. W. Hodge, S. E. Safo, and Q. Long, "Multiple imputation using dimension reduction techniques for high-dimensional data," 2019. [Online]. Available: https://arxiv.org/abs/1905.05274

[49] H. Abdi and L. J. Williams, "Principal Component Analysis," *WIREs Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.101

[50] M. Kenyhercz and N. Passalacqua, "Chapter 9 - missing data imputation methods and their performance with biodistance analyses," in *Biological Distance Analysis*, M. A. Pilloud and J. T. Hefner, Eds. San Diego: Academic Press, 2016, pp. 181–194. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128019665000093

[51] M. Zhu and X. Cheng, "Iterative knn imputation based on gra for missing values in tplms," in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 01, Dec 2015, pp. 94–99. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7490714

[52] A. Gkillas and A. S. Lalos, "Missing Data Imputation for Multivariate Time Series in Industrial IOT: A Federated Learning Approach," in *2022 IEEE 20th International*

Conference on Industrial Informatics (INDIN), 2022, pp. 87–94. [Online]. Available: https://ieeexplore.ieee.org/document/9976093

[53] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, ser. 11/04 16:50:48 GMT. Wiley, 2000, no. pt. 1. [Online]. Available: https://books.google.ca/books?id=_sO4DwAAQBAJ

[54] H. Hegde, N. Shimpi, A. Panny, I. Glurich, P. Christie, and A. Acharya, "MICE Vs PPCA: Missing Data Imputation in Healthcare," *Informatics in Medicine Unlocked*, vol. 17, p. 100275, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352914819302783