Titre: Title:	Optimizing Federated Learning for Remote Patient Monitoring Systems
Auteur: Author:	Negin Keshavarz
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Keshavarz, N. (2024). Optimizing Federated Learning for Remote Patient Monitoring Systems [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/62473/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/62473/
Directeurs de recherche: Advisors:	Samuel Pierre
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Optimizing Federated Learning for Remote Patient Monitoring Systems

NEGIN KESHAVARZ

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie informatique

Décembre 2024

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Optimizing Federated Learning for Remote Patient Monitoring Systems

présenté par **Negin KESHAVARZ**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de:

Martine BELLAICHE, présidente
Samuel PIERRE, membre et directeur de recherche
Ranwa AL MALLAH, membre et codirectrice de recherche
Alejandro QUINTERO, membre

DEDICATION

This thesis is dedicated to my beloved parents, Navaz and Shekoufeh,

My cherished family and my dear professor, Samuel Pierre.

Their unwavering support, guidance, and belief in me have made completing my graduate studies possible.

RÉSUMÉ

Les soins de santé jouent un rôle essentiel dans nos vies, et les avancées dans ce domaine progressent à un rythme rapide chaque jour. Cette dissertation explore l'optimisation de l'apprentissage fédéré pour la surveillance à distance des patients. L'apprentissage fédéré, une approche collaborative de l'apprentissage automatique qui maintient localement les données sensibles reliées à la santé, permet d'entraîner des modèles d'intelligence artificielle sur des systèmes décentralisés. Cette recherche vise à améliorer ce processus quant à l'efficacité, la précision et la minimisation des coûts de communication. Ce sont des indicateurs essentiels pour les applications en surveillance de la santé. Cela a été réalisé en créant et testant une architecture complète d'apprentissage fédéré basée sur le jeu de données MHEALTH, qui contient des données physiologiques d'envergure. L'étude a débuté par le prétraitement des données, incluant l'encodage des données catégorielles, la standardisation des données en entrée et l'équilibrage du jeu de données pour garantir des données d'entrée de haute qualité pour les modèles d'apprentissage automatique. Au cœur du système se trouve un modèle BiLSTM, un réseau de neurones récurrent sophistiqué conçu pour les données de séries temporelles, qui a démontré une précision élevée dans la reconnaissance des activités humaines avec des taux de précision dépassant 99%. Les techniques d'apprentissage fédéré, en particulier l'outil Flower avec la technique de moyenne fédérée pour l'agrégation de modèles, ont permis un entraînement décentralisé sur plusieurs clients. Cette configuration aboutit à un modèle global ayant une précision impressionnante entre les nœuds. De plus, une optimisation multi-objectifs de Pareto a été appliquée pour avoir un équilibre entre les différents indicateurs à l'étude, obtenant un compromis optimal entre précision, temps et volume de transmission des données. Cette recherche souligne le potentiel de l'apprentissage fédéré pour transformer la surveillance à distance des patients en permettant un traitement des données de santé efficace et précis. En exploitant un traitement des données optimisé, un modèle BiLSTM performant et une optimisation multi-objectifs ciblée, cette approche d'apprentissage fédéré ouvre la voie à l'échange d'informations de santé précises et fiables.

ABSTRACT

Healthcare plays an essential role in our lives, and advancements in this field are happening at a rapid pace every day. This dissertation explores the optimization of Federated Learning (FL) for Remote Patient Monitoring (RPM). Federated learning, a collaborative machine learning approach that keeps sensitive health data local, enables the training of AI models across decentralized systems. This research aims to improve the FL process by focusing on efficiency, accuracy, and minimizing communication costs, all essential for real-world applications in health monitoring. This was accomplished by creating and testing a thorough federated learning architecture on the MHEALTH dataset, which includes extensive physiological data. The study started by addressing data preprocessing, which included encoding categorical data, standardizing features, and balancing the dataset to guarantee high-quality input for the machine learning models. At the core of the system is a BiLSTM model, a sophisticated recurrent neural network designed for time-series data, which demonstrated high precision in recognizing human activities with accuracy rates surpassing 99%. Federated learning techniques, specifically the Flower framework with Federated Averaging (FedAvg) for model aggregation, allowing for decentralized training across multiple clients. This setup results in a global model with impressive accuracy across nodes. Furthermore, multi-objective Pareto optimization was applied to balance key metrics, achieving an optimal trade-off between accuracy, time, and data transmission volume. This research underscores the potential of federated learning to transform remote patient monitoring by enabling efficient and accurate health data processing. By leveraging optimized data handling, a high-performing BiLSTM model, and targeted multi-objective optimization, this federated learning approach paves the way for delivering precise and reliable health insights.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
DL	Deep Learning
EHR	Electronic Health Records
FedAvg	Federated Averaging
FedDL	Federated Deep Learning
FL	Federated Learning
GCP	Google Cloud Platform
HAR	Human Activity Recognition
IDE	Integrated Development Environment
IoMT	Internet of Medical Things
LSTM	Long Short-Term Memory
ML	Machine Learning
MOO	Multi-Objective Optimization
PFL	Personalized Federated Learning
ROC	Receiver Operating Characteristic
RNN	Recurrent Neural Network
RPM	Remote Patient Monitoring
SNN	Spiking Neural Networks
SVM	Support Vector Machine
VSCode	Visual Studio Code

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Federated learning has been a breakthrough in improving the performance of healthcare applications since Google first introduced it [13]. It is an innovative approach where different systems work together to train a shared model by combining their models, all without sharing sensitive data. This means there is no need to send large datasets across networks, helping to reduce the amount of data exchanged and minimize delays. In the context of smart cities, one important application for health services is remote patient monitoring [14]. This system collects medical and health data from patients and then transmits it to healthcare providers for evaluation and recommendations. This dissertation focuses on optimizing federated learning in health monitoring systems to make healthcare services more efficient. The primary objective of this research is to propose a multi-objective approach to optimize federated learning in healthcare monitoring. We aim to design a more efficient federated learning process that improves performance, focusing on reducing time and the amount of data exchanged and addressing the challenge of high communication costs. We then evaluate, on a realistic dataset, the effectiveness of federated learning-based models in classifying physical activities through monitoring systems. In this chapter, we will start by explaining the key definitions and concepts of federated learning in healthcare. Then, we will outline the main problems we aim to solve, followed by the goals of the dissertation. Finally, we will provide an overview of the structure of the dissertation.

1.1 Basic Definitions and Concepts

In this section, we will review the basic definitions and the main concepts regarding the application of federated learning in healthcare systems. We will explore how it applies to Remote Patient Monitoring systems and Human Activity Recognition (HAR). Moreover, we will explain how federated learning can enhance the performance of health monitoring systems.

1.1.1 Federated Learning

The use of Machine Learning (ML) with health data is becoming increasingly popular for its potential to enhance healthcare decision-making. However, training high-quality ML models requires access to diverse and comprehensive datasets, which are often difficult to obtain due to the sensitive nature of patient medical data. In this context, FL offers a valuable solution by enabling the distributed training of ML models on remotely hosted datasets without the need to

centralize or share sensitive raw data. This approach aligns better with regulatory standards, enhancing the trustworthiness and data sovereignty of ML-based healthcare systems. FL has emerged as a powerful approach for collaborative training across multiple clients. The client updates collectively represent the insights gained from local data, which play a crucial role in the performance of the federated learning process. Moreover, there is a trade-off between model performance and efficiency in federated learning. To improve model performance, frequent updates are needed, which increases communication costs. However, improving efficiency by reducing communication can decrease model accuracy. Finding the right balance between these two elements is essential, especially in resource-constrained environments. In a federated learning system, a central entity distributes a shared training algorithm to all participating data holders. Each participant uses this algorithm to train a local model on their dataset. The clients only transmit the resulting model parameters back to the central entity. The central entity then applies an aggregation algorithm to merge these local parameters, creating a unified global model that reflects the combined insights from all participants [2].

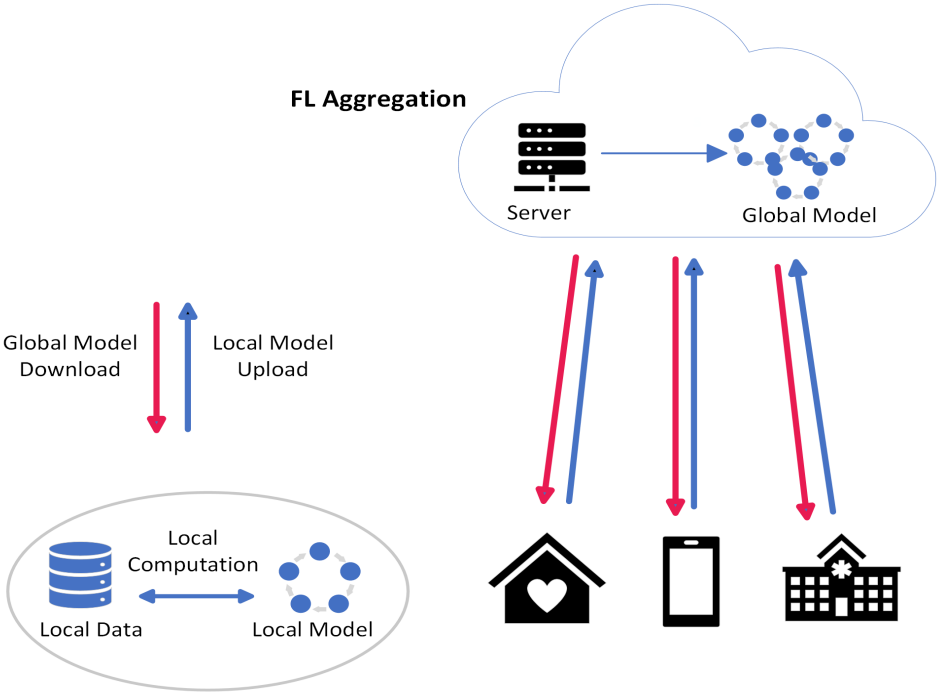

Figure 1.1 FL process.

1.1.2 Healthcare System

In healthcare, systems are designed to deliver services through a coordinated network of stakeholders, including hospitals, clinics, healthcare professionals, and medical devices [15]. These systems aim to ensure accurate diagnosis, effective treatment, continuous monitoring, and comprehensive care for individuals. In recent years, technological advancements have dramatically transformed healthcare delivery, particularly through the integration of the Internet of Medical Things (IoMT) [15]. IoMT connects medical devices and wearable technologies, allowing real-time data collection from patients. This has paved the way for smart healthcare systems, which offer advanced services such as remote patient monitoring, personalized treatment plans, and improved healthcare management, significantly enhancing the quality and efficiency of care. Emerging technologies such as FL further revolutionize healthcare systems by addressing key challenges, particularly in data privacy. FL enables the training of AI models on decentralized data, preserving patient privacy while harnessing insights from distributed datasets. This approach allows for more accurate decision-making across the healthcare system without the need to centralize sensitive medical information, which is crucial in maintaining data privacy and personalized care at scale [11][12].

1.1.3 Remote Patient Monitoring Systems

The integration of Artificial Intelligence (AI) in healthcare is advancing at an unprecedented pace, with RPM emerging as a vital application. RPM facilitates the continuous monitoring of patients with chronic conditions, elderly individuals receiving home care, and even hospitalized patients. It offers healthcare professionals a means of extending their care beyond the traditional confines of hospitals, particularly in general medical and surgical wards. RPM has been employed to remotely monitor patients through wearable devices or sensors. These systems track vital signs and other physiological parameters, such as HAR, which aid clinicians in making informed clinical decisions or developing treatment plans, particularly for conditions like movement disorders [3]. The incorporation of ML and AI into RPM enhances healthcare providers' ability to assess patient health by analyzing vital signs and recognizing patterns in physical activity. These intelligent systems enable the visualization of patient health status, aiding in diagnosis, prognosis, and clinical decision-making. This fusion of AI and RPM strengthens the precision and timeliness of care, ultimately improving patient outcomes [4].

1.1.4 Human Activity Recognition

Human Activity Recognition is an increasingly active research area that involves the processing of sensor data to identify human activities in real-time. This data can be collected from various multimodal wearable sensors, such as accelerometers, gyroscopes, and magnetometers commonly found in smartphones and smartwatches [6]. These signals are analyzed using different techniques to develop robust classification algorithms that accurately identify Activities of Daily Living (ADLs). By recognizing human activities, HAR systems contribute to a deeper understanding of the behavior and context of users, enabling the development of intelligent systems that can adapt to their needs in real-time. This has widespread applications in healthcare, smart environments, and personalized technology, making HAR a key component in the advancement of context-aware computing. One of the main challenges of HAR refers to the privacy concerns brought by the classical centralized approach that is used to develop the algorithms to recognize the desired activities. This approach needs to gather a large amount of data from users' wearable sensors to train, test, and validate the HAR algorithms. The centralized approach also implies a communication cost due to the transfer of large amounts of data. FL can address issues of sharing and centralizing the required data to develop HAR algorithms [5][6].

1.1.5 Overview of Machine Learning Techniques in FL and Related Concepts

Federated learning has increasingly become integrated into machine learning approaches due to its ability to protect data privacy while maintaining model efficiency. Among the models commonly supported by FL are linear models, decision trees, and neural networks, each offering unique benefits depending on the task.

- **Linear Models:** Linear models are a key part of FL due to their simplicity and ease of implementation. The most common types include linear regression, ridge regression, and lasso regression. These models work well in federated environments because they require relatively low computational resources and are straightforward to deploy. The ability to train these models in a distributed manner while achieving similar accuracy to centralized approaches makes them highly valuable. Despite their simplicity, linear models remain an effective choice for federated learning tasks, especially when the goal is to combine privacy and scalability [7].

- **Tree Models:** Federated learning is increasingly being used to train decision tree models, including popular algorithms like Gradient Boosting Decision Trees (GBDT) and random forests. GBDT has gained attention for its excellent performance in both classification and regression tasks. It effectively combines multiple decision trees to make more accurate predictions, which makes it well-suited for federated learning environments. In federated learning, decision trees can be trained across multiple data owners. Aggregation methods can be applied to combine the results from different trees into a unified model. Furthermore, frameworks like SecureBoost have been developed to facilitate training GBDT models on both horizontally and vertically partitioned data [17]. These systems enable organizations to leverage federated learning while making it easier for users to implement powerful machine learning models across different data sets [7].
- **Neural Network Models**: In the context of federated learning, the use of deep neural networks has gained substantial interest due to their capacity to handle distributed data. One of the key advantages of federated learning for neural networks is its ability to aggregate locally trained models from different nodes or clients into a global model. Neural network algorithms have already demonstrated encouraging results in classification tasks, making them a powerful tool in the realm of machine learning. The rapid advancement of machine learning demands ongoing innovation in federated learning to keep up with emerging needs [7].

1.2 Problem Statement

As federated learning continues to gain traction and find applications across diverse industries, it faces significant challenges that must be addressed to ensure efficient and accurate implementation in real-world scenarios. One of the most pressing issues is the communication costs. This challenge can severely impact the feasibility of deploying HAR systems, particularly in environments where numerous users or devices are involved. The complexity of communication costs in FL stems from the necessity to send model updates from many devices to a central server. This requirement introduces several key challenges.

- **Communication overhead**: When dealing with sophisticated models, such as those based on deep learning, model updates can be quite large. The frequent transmission of such

substantial updates can lead to high data transfer requirements**,** which may hinder the overall performance of the system. Moreover, the sheer volume of data exchanged can lead to inefficiencies and increased latency, making it difficult to meet the stringent training latency requirements often required in healthcare contexts. [8][9].

- **Data sensitivity**: The health data often contains sensitive information related to users' behaviors and personal circumstances. The sharing of such data, even in a federated context, poses risks of unintended exposure, highlighting the need for robust privacy measures [8][9].

Considering these challenges, our primary research question is the following: What optimization can be done to the federated learning process in health monitoring to ensure efficient and accurate training? From this main question, we can derive the following three secondary questions:

1- What AI methods are capable of classifying patients' physical activities?
2- What approach can be adopted to minimize resource utilization and address communication costs during the model update while maximizing model accuracy?
3- What are the performance metrics for estimating the effectiveness of AI models used in the proposed deployment?

1.3 Research Objectives

The main objective of this dissertation is to propose a multi-objective approach to optimize federated learning in healthcare monitoring.
This research specifically aims to:

1- Design an optimized federated learning process to enhance performance in terms of accuracy, time, and the amount of exchanged data.
2- Address the communication costs incurred by the FL process for the training of AI models for the task of HAR.
3- Evaluate the effectiveness of AI methods for the classification of physical activities.

1.4 Dissertation Plan

The rest of the dissertation is structured as follows. Chapter Two provides a comprehensive literature review, focusing on two key areas: healthcare systems, particularly in the realm of human

activity recognition, and the use of federated learning in healthcare applications. The chapter also emphasizes the crucial role of RPM systems in modern healthcare. A critical review of various machine learning models used in HAR is conducted to understand their application in classifying human activities. The chapter explores how federated learning can transform healthcare by enabling decentralized model training, thereby enhancing patient outcomes while maintaining data privacy. However, challenges associated with federated learning are highlighted as key areas for research and optimization. Chapter 3 details the proposed optimization method for federated learning and provides an in-depth explanation of the technologies adopted throughout the optimization process. It also covers the data preparation steps and offers a comprehensive overview of the datasets utilized. The chapter outlines the federated learning techniques and optimization strategies applied in HAR, along with a discussion of the proposed architecture. Additionally, it explains the configuration of the experimental testbed and presents the methodological procedures with detailed clarity. The results and evaluation of the system will be presented in Chapter 4, while Chapter 5 concludes the dissertation by summarizing the key contributions, acknowledging limitations, and offering suggestions for future research directions.

CHAPTER 2 LITERATURE REVIEW

In this chapter, we will provide an extensive literature review focusing on healthcare systems, human activity recognition, and the application of federated learning in the healthcare sector. By examining existing techniques and methods utilized in federated learning for healthcare, we aim to identify their limitations and propose robust solutions that align with the objectives of this dissertation. Through this detailed analysis, we aim to contribute to the development of more efficient and effective healthcare solutions.

2.1 Healthcare Systems

This section aims to conduct a thorough review of relevant studies on healthcare systems. The goal of this review is to highlight significant findings and contributions to health monitoring systems specifically. The significance of health in our lives is undeniable, and we continually witness rapid advancements in this domain. Notably, the integration of IoMT with healthcare systems stands out as a pivotal aspect of this progress. In recent times, IoMT has facilitated the gathering of extensive health data, enabled remote healthcare services, and facilitated advanced monitoring systems. Despite these advancements, several challenges have emerged, including concerns regarding processing time, training accuracy, the amount of transmitted data, and data privacy. The utilization of these innovative health systems presents unique opportunities for enhancement, such as the implementation of optimized algorithms for optimizing parameters like time in intelligent healthcare systems.

2.2 Remote Patient Monitoring Systems

Remote patient monitoring is an essential component of modern healthcare, allowing continuous observation and management of patients outside traditional clinical settings. This approach is particularly valuable for managing chronic diseases, post-operative care, and elderly patients, enabling timely interventions, and reducing the need for hospital visits. Health monitoring is one of the primary areas within IoMT healthcare services, with plenty of articles addressing this topic. In this section, we will examine a selection of these articles within this field. Due to the rapid advancement of the IoMT, big data, and artificial intelligence, significant progress has been made in health monitoring systems. These advancements aim to assist doctors in remotely monitoring patients and offering a sense of security, particularly for elderly individuals living alone [35].

The authors in [22] developed an enhanced RPM system integrating robust client privacy measures and applied it to real-world scenarios. The system was designed to accurately capture vital signs and natural body movements of multiple mobile patients, thereby facilitating remote healthcare management. One notable aspect of the study was the introduction of a novel heterogeneous stacked federated learning architecture. This architecture aimed to optimize the collaborative learning process while preserving data privacy across multiple clients. To evaluate its effectiveness, the framework was implemented using the MHEALTH dataset, a rich source of mobile health sensor data.

Particularly, the research in [5] involved training three distinct artificial intelligence models Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Bidirectional Long Short-Term Memory (Bi-LSTM) on individual subject data. Notably, the CNN model was leveraged to classify eight physical activities, utilizing data from sensors placed strategically on the subject's chest, right wrist, and left ankle. The dataset encompassed 12 natural activities, each assigned a numerical label ranging from 0 to 12. While nine subjects were treated as individual clients, one subject's data served to train the global model. The communication protocol between local models and the global model, crucial in FL, was orchestrated through the innovative FedStack algorithm proposed in the study. Subsequently, the proposed architecture was deployed on the mobile health sensor data from ten distinct subjects to classify the 12 routine activities. The performance evaluation of the model was based on various metrics, including balanced accuracy, precision, recall, and F1-score. However, it's worth noting that the study acknowledged potential privacy risks inherent in the FL process, particularly highlighting the possibility of privacy breaches through reverse engineering techniques. Furthermore, the assumption of an equal number of classification labels for all ten clients underscores the need for careful consideration of data distribution and model generalization in such collaborative learning frameworks.

2.3 Human Activity Recognition

Human Activity Recognition is an interdisciplinary field that focuses on identifying and understanding human actions using sophisticated data analysis techniques [36]. This technology has gained significant attention due to its wide range of applications in health monitoring, sports analysis, and human-computer interaction. HAR systems convert raw sensor data into meaningful insights, thus bridging the gap between raw data and the understanding of human behavior. This transformation is crucial for making the data actionable in real-world applications. The core

components of HAR begin with sensors, which can be broadly categorized into wearable and environmental types. Wearable sensors, such as accelerometers, gyroscopes, and magnetometers, are typically embedded in smartphones, smartwatches, or specialized fitness trackers. These sensors capture data related to the movement and orientation of the human body. Feature extraction is a critical step where specific attributes or patterns indicative of particular activities are identified from the preprocessed data. The extracted features are then used by activity recognition algorithms to classify the activities accurately [36].

2.3.1 Machine Learning Models in HAR

Machine learning models, including Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and neural networks, are often employed in HAR. These models are trained on labeled datasets to recognize various activities. In recent years, deep learning models such as convolutional neural networks and Recurrent Neural Networks (RNNs) have gained prominence accuracy in human activity recognition due to their ability to automatically learn features from raw data [37]. The classification process involves the model categorizing the extracted features into predefined activity classes, such as walking, running, and sitting. Real-time classification is particularly useful in applications requiring immediate feedback, such as fall detection systems for elderly care. As sensor technology, machine learning, and data analytics continue to advance, HAR is poised to become even more accurate and versatile, opening up new possibilities and applications in the future.

The authors in [30] propose a biological brain-optimized neuromorphic Convolution Neural Network model (biCNN-HAR). This innovative model applies a classification algorithm to the features it extracts, which are then used to recognize twelve distinct HAR activities. The training of the model is specifically designed to enhance its ability to accurately identify these activities. The performance of biCNN-HAR is evaluated using the MHEALTH dataset. This evaluation involves experimenting to thoroughly assess the model's efficiency on the MHEALTH dataset. Key performance metrics such as accuracy, precision, recall, and f-measure are used to determine the model's effectiveness. Implementation of the model is carried out on the neuromorphically optimized framework Nengo. This framework is instrumental in reducing communication latency and execution cost, all of which are critical factors when capturing and analyzing human activities. In the study, they face potential accuracy issues due to diverse sensor data quality. The flexibility

of this model suggests that it can be expanded in the future to recognize more complex activities, thereby enhancing its applicability in various domains of human activity recognition.

Authors in [29] propose a CNN architecture with sensor fusion and random forest classification. Their research delves into the effects of the Null class, a common occurrence in health-related datasets, which can significantly impact model performance. The proposed method employs CNN with sensor fusion in conjunction with recurrent neural network models. They integrate various processing steps, including pre-processing, processing, and post-processing, throughout the training and testing phases. These trained models are utilized to classify the human activities of the subjects, offering insights into their performance under different conditions. The evaluation encompasses the model sizes, training times, and testing times, providing a comprehensive analysis of their efficiency. The authors face potential accuracy issues due to diverse sensor data quality. To benchmark the performance, they employ the MHEALTH dataset, aiming to identify the model that excels in terms of accuracy, precision, recall, F-score, and computation time. Additionally, they can use other datasets that have not yet been incorporated into their experiments, suggesting a broader applicability of their approach. They also could emphasize the importance of active learning, which can facilitate future data collection processes and encourage users to provide labels for the most critical data.

2.4 Federated Learning in Healthcare

Federated learning represents a transformative approach in healthcare, addressing the critical challenge of leveraging vast amounts of medical data while preserving model accuracy. Traditional centralized machine learning model, as shown in Figure 2.1, requires data aggregation, which poses significant risks related to data breaches. In contrast, federated learning allows for decentralized model training across multiple institutions without the need to transfer sensitive patient data. This approach facilitates collaboration across various healthcare providers, enabling the development of more robust and generalized models [38].

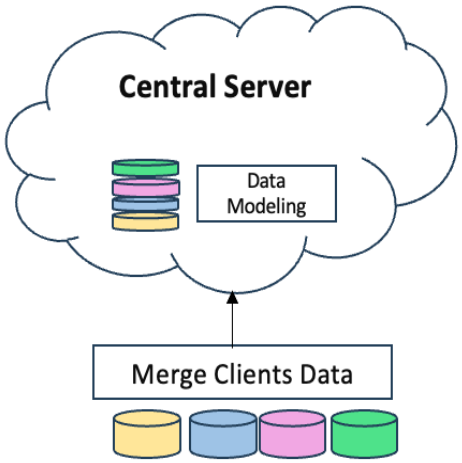

Figure 2.1 Centralized traditional machine learning.

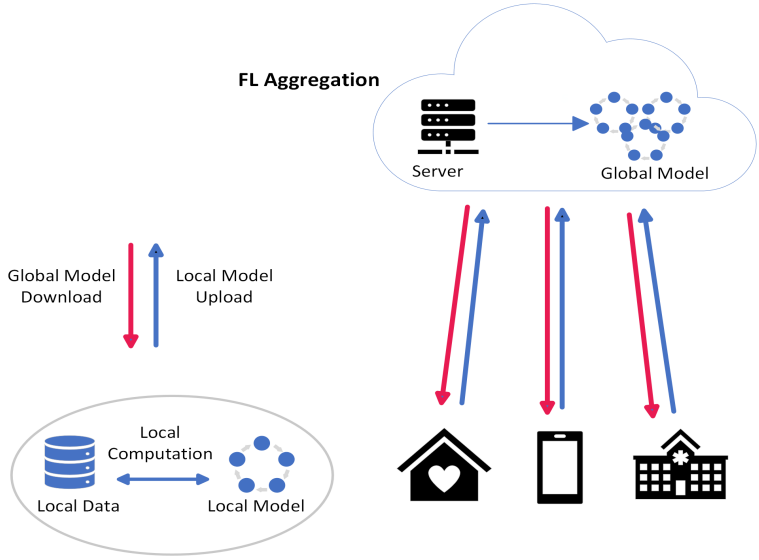

Figure 2.2 Federated learning.

In healthcare, the heterogeneity of data sources—ranging from Electronic Health Records (EHRs) to wearable device data outputs—presents a unique challenge. Federated learning effectively manages this by allowing each participating institution to train the model on its local dataset. The local updates are then aggregated to improve the global model. One of the significant advantages of federated learning in healthcare is its ability to improve patient outcomes. By enabling the

integration of diverse datasets, federated learning models can capture a wide range of patient demographics and clinical conditions, leading to more accurate predictions and personalized treatment plans. For instance, federated learning can help in the early detection of diseases by analyzing patterns across vast datasets, which would be infeasible for individual institutions to achieve independently.

Moreover, federated learning promotes equitable access to advanced machine learning technologies. Smaller healthcare providers and institutions, which may lack extensive datasets or computational resources, can contribute to and benefit from the collective intelligence of federated learning. This decentralization of AI technology ensures that advancements in healthcare are not confined to well-funded institutions but are accessible to a broader spectrum of the healthcare ecosystem.

Authors in [28] conducted a comprehensive analysis of Federated Learning, contrasting it with traditional centralized methods. In their study, they employed the FedAvg method for the model aggregation step, which facilitated the combination of locally trained models from multiple clients into a single global model. This approach included various communication rounds to progressively enhance the performance of the global model. The trained models underwent evaluation using data from a single subject, with the evaluation process repeated multiple times to ensure the reliability of the evaluation metrics for each communication round. For the experiments, the researchers utilized the MHEALTH dataset, and the neural network architecture chosen was the Multi-Layer Perceptron (MLP), which was trained and tested within the FL framework. Beyond the initial experiments, they could extend their analysis to assess the performance of FL on other HAR datasets. They could also explore other types of FL, such as vertical and transfer learning, to provide a broader understanding of FL's capabilities. Additionally, they could study other neural networks and machine learning techniques, further illustrating the applicability of FL in various contexts. One notable aspect of their research is that they did not study the processing time of local training on clients' nodes. This aspect was identified as an area for future research, highlighting a potential direction for further optimizing the FL process.

2.4.1 Core Technologies and Algorithms in Federated Learning

The exploration of core federated learning technologies and algorithms has highlighted significant contributions, underlining the essential role of machine learning and artificial intelligence in advancing federated learning applications. The literature has extensively explored the potential of distributed machine learning algorithms, such as FedAvg and its variants, to execute complex tasks while maintaining data privacy. FedAvg, also referred to as Local-SGD, is a foundational federated learning algorithm. In FedAvg, clients perform several local SGD steps independently before sending their updates to a central server for aggregation [39].

To address specific challenges in federated learning, such as varying computational capabilities of clients and communication constraints, several advanced variants of FedAvg have been developed. FedProx incorporates a proximal term in the local optimization process to restrict the extent to which local updates can deviate from the global model. Traditional federated learning algorithms, like FedAvg, operate under the assumption that aggregating local client updates will yield a single global model that performs adequately for all clients. However, this assumption falters when faced with non-IID data, resulting in suboptimal model performance and convergence issues. To overcome these challenges, Personalized Federated Learning (PFL) has been proposed. PFL aims to create personalized models tailored to each client's specific data distribution while still benefiting from the collective knowledge shared across all clients. These advancements significantly enhance the robustness and scalability of federated learning systems. Furthermore, advancements in AI and ML methodologies are continuously being integrated into federated learning frameworks. Techniques such as transfer learning and meta-learning are leveraged to improve model generalization across diverse clients [40].

2.4.2 Federated Learning Applications in Healthcare

Federated learning has emerged as a ground-breaking approach for various healthcare applications, offering a pathway to leverage large-scale, diverse datasets while maintaining accuracy and privacy standards. Federated learning is instrumental in advancing remote patient monitoring systems, which are becoming increasingly important in managing chronic diseases and post-operative care. Wearable devices and remote monitoring tools generate vast amounts of data that can be used to predict patient health trends. Federated learning models can analyze this distributed data to detect

early warning signs of health deterioration, allowing healthcare providers to intervene promptly [54].

One of the primary benefits of integrating federated learning into RPM is the enhancement of predictive capabilities. Wearable devices, mobile health apps, and home monitoring systems generate vast amounts of data, including vital signs, activity levels, and other health metrics. Federated learning allows models to be trained locally on devices or within healthcare facilities where the data resides. These local models then share their learned parameters with a central server, which aggregates them to update the global model. This process ensures that sensitive health data never leaves the patient's device or healthcare provider's system.

Federated learning enhances the RPM systems' ability to detect subtle changes in patient health by leveraging a broader range of data. For instance, data from patients with similar conditions across different geographic locations and demographics can be used to improve the accuracy of predictive models. This collaborative approach allows for more comprehensive pattern recognition, enabling healthcare providers to identify early warning signs of health issues and intervene promptly [22].

Despite its advantages, implementing federated learning in RPM systems poses several challenges. The heterogeneity of data from various sources can complicate model training, requiring advanced algorithms capable of handling diverse data types and qualities. Another consideration is the computational and communication overhead associated with federated learning. For instance, devices used in RPM, such as wearables and smartphones, may have limited processing power, making efficient model training and communication strategies essential. Research is ongoing to develop federated learning algorithms that can operate effectively in such constrained environments.

The authors in [20] Propose FedHealth, a framework for wearable healthcare to monitor physical activities. The main objective is to achieve accuracy in federated learning that is close to or better than that of conventional learning. Using the UCI Smartphone dataset for human activity recognition, they adopted deep neural networks to learn both the cloud and user models. They performed a transfer learning method to make the model. To construct the problem situation in FedHealth, they extracted five subjects and regarded them as isolated users. Data from the remaining 25 users were used to train the cloud model. They applied FedHealth in the diagnosis of

Parkinson’s Disease by developing a smartphone application to collect patients' acceleration data during symptom tests. For each test, symptoms were ranked into five levels, from normal to severe. The authors struggled with accuracy due to heterogeneous data sources, and finally, they evaluated the classification accuracy and macro F1 score on the collected dataset. The framework can be extended with various transfer learning methods. It is designed to work with multiple users and a single server but can be extended to accommodate more users. It does not explore the challenges and requirements of different healthcare domains.

The authors in [21] developed a federated deep learning algorithm aimed at protecting the privacy of health data, particularly focusing on heart activity. This work serves as a case study in stress recognition, where a global model capitalizes on data sourced from diverse institutions. The preprocessing of heart activity data is carried out using a dedicated module written in MATLAB programming language. Feature extraction is facilitated through the utilization of MATLAB's built-in toolboxes. Following this, the federated deep learning algorithm is applied to the heart activity data, with 80% allocated for training and 20% for testing. It's important to note that the communication burden of federated learning scenarios was not assessed, and it's recognized that varying data qualities from wearable devices may impact the system's performance.

The authors in [36] proposed the Federated Learning Framework for Elderly Healthcare Using Edge-IoMT (FEEL), which aims to develop an efficient edge-IoMT-based smart healthcare system for elderly people. This system is designed to continuously monitor body vitals, process data to update local models, communicate with neighboring nodes if any unusual conditions arise, and send alerts in case of emergencies. By providing continuous and proactive health monitoring, FEEL ensures timely medical interventions, which are crucial for the elderly population. The input data for the FEEL system comes from various sources, including vital signs, mobility-related data, and contextual information about the surroundings. This comprehensive data collection allows for a holistic understanding of the individual's health status. The computation and storage of the global model are managed using the Google Cloud Platform (GCP), providing a reliable and scalable infrastructure [53]. To detect abnormal activities, the FEEL framework employs an advanced algorithm for activity data analysis. Daily activities, which consist of numerous sub-activities, are meticulously monitored. These sub-activities are unit-level activities identified through various sensors. The system learns the normal sequences of these activities using an adaptive and stacked long short-term memory method, which helps in recognizing deviations from typical patterns. The

accuracy of activity recognition within the FEEL framework is measured using precision, recall, and F1-measure metrics, ensuring the reliability of the monitoring system. Despite its innovative design and robust capabilities, the FEEL framework has certain limitations. It requires the integration and analysis of data from a broader range of modalities to fully understand and interpret complex human activities. Additionally, the age range of individuals that the system can effectively monitor is limited.

The authors in [33] propose FedDL, a novel federated learning system designed specifically for human activity recognition. They adopt convolutional neural networks for the HAR task, leveraging the powerful feature extraction capabilities of CNNs to improve recognition accuracy. FedDL addresses the need for efficient learning models that can be deployed across multiple devices. To demonstrate the feasibility and effectiveness of their approach, they designed and implemented a FedDL instance on Amazon Compute Cloud. This cloud-based implementation allows for efficient training across multiple devices, facilitating a more robust evaluation of the system. The distributed nature of FedDL ensures that data remains localized to the user's device, thus maintaining compliance with data protection regulations.

The authors implement FedDL and evaluate it using a self-collected LiDAR dataset, as well as four public real-world datasets. This comprehensive evaluation ensures that the system is tested across a variety of conditions and data types, providing a thorough assessment of its performance. One of the key aspects of their evaluation is the comparison of the overall accuracy of FedDL with some baseline methods, such as FedAvg, which is the standard FL method. This comparison highlights the advantages of FedDL in terms of both accuracy and efficiency, demonstrating its potential as a superior alternative to existing FL approaches. The reduced communication overhead of FedDL is particularly significant in scenarios where the amount of data that can be transferred is limited or costly. In addition to the cloud-based implementation, the authors could deploy FedDL on devices like smartphones to evaluate the system's overhead. This deployment is crucial for assessing the practical applicability of FedDL in real-world scenarios. Furthermore, the authors could investigate different use cases beyond human activity recognition through domain adaptation techniques.

In their research, the authors in [35] propose a Federated Learning-based Person Movement Identification (FL-PMI) system. This innovative approach employs bidirectional long short-term

memory to extract features from sensor data, coupled with deep reinforcement learning for effective data classification. To rigorously evaluate the performance of the FL-PMI system, the authors adopted a systematic approach by splitting the dataset into five equal subsets. In each iteration of their evaluation, one subset was designated for testing, while the remaining four subsets were utilized for training. This cross-validation method ensures a thorough and reliable assessment of the system's performance across different segments of the data. The study presents a detailed comparison of the FL-PMI system's performance against existing solutions, focusing on key metrics such as accuracy, F1-score, computation cost, and transmission data. This comparative analysis underscores the efficiency of the FL-PMI system in identifying personal movements based on sensor data. The results demonstrate that FL-PMI not only meets but often exceeds the performance of current state-of-the-art methods, highlighting its potential for widespread adoption. Despite its promising performance, the authors recognize that FL-PMI is vulnerable to privacy issues and security threats, which could undermine its practical utility. To address these concerns, they can propose the integration of blockchain technology into the FL-PMI system. Blockchain's decentralized and secure nature can significantly enhance user privacy and protect against malicious attacks. Embedding blockchain technology not only mitigates security risks but also paves the way for broader acceptance and implementation of federated learning-based systems in sensitive and privacy-focused domains.

Authors in [27] propose FedHome, a novel cloud-edge-based federated learning framework designed for in-home health monitoring. This innovative framework aims to achieve accurate health monitoring without compromising user data privacy. The researchers devised a unique network model that is collaboratively trained on both the cloud and edge devices, ensuring robust performance and adaptability. The user's local model is trained and refined with a class-balanced dataset generated from their data. FedHome employs the FedAvg algorithm for efficient cloud-edge collaborative training, leveraging the strengths of both computing environments. The main architecture of the model is based on a convolutional neural network chosen for its effectiveness in handling complex patterns and large datasets. Extensive experiments are conducted using a realistic human activity dataset, demonstrating the framework's practical applicability. The results highlight the average precision, recall, and F1-score for each activity across all 30 users, showcasing FedHome's potential to deliver high-quality health monitoring services. Despite its promising results, the application of the FedHome framework to more realistic and varied in-home

health monitoring environments remains unexplored, pointing to an avenue for future research and development.

The integration of federated learning into RPM is a promising area of research with significant potential for future advancements. Developing more robust and scalable federated learning frameworks will be crucial for widespread adoption.

2.4.3 Optimization in Federated Learning

Federated Learning presents significant challenges in terms of time efficiency, model accuracy, and the amount of data transferred. Time efficiency in FL is paramount, given the iterative nature of the training process, which involves frequent communication between clients and a central server. Furthermore, constraints on model accuracy and the amount of data exchanged pose a significant challenge in FL, especially in environments with limited connectivity.

Researchers in [24] propose a resource allocation algorithm aimed at determining the optimal amount of data to be transferred by each device participating in a federated learning system. Their approach employs a weighted sum method to formulate the optimization problem, which is then solved using CVX, a software package for convex optimization. The algorithm introduces two weight parameters to effectively manage the trade-offs between the amount of data exchanged and transmission power control among the devices. To evaluate the performance of the proposed algorithm, the researchers utilize floating-point operations to measure the time complexity. They conduct extensive simulations by running the algorithm 100 times and compute the average values of time consumption to ensure robust and reliable results. This repeated testing provides a comprehensive understanding of the algorithm's efficiency and resource management capabilities in a federated learning context. However, the authors do not delve into the analysis of the optimal timing for executing the federated learning process. This omission leaves a potential area for further research, focusing on identifying the most efficient moments for federated learning updates to enhance overall system performance.

Authors in [23] propose an innovative algorithm designed to adapt local training and model sharing to suit the specific applications, profiles of IoT devices, and the unique environmental contexts in which these devices operate. This approach ensures that the algorithm is highly adaptable and can function effectively across a diverse range of scenarios and device configurations. To enhance the

security and efficiency of the FL process, they optimize the algorithm by fine-tuning its parameters to account for the varying characteristics of the communication network. This optimization is crucial for maintaining robust security measures and ensuring seamless data exchange within the network. The problem is modeled as a Multi-Objective Optimization (MOO) problem, which allows for the balancing of multiple competing objectives, such as minimizing latency and maximizing throughput. By employing a MOO framework, the proposed solution can achieve an optimal trade-off among these various objectives, thereby improving overall system performance. To validate their approach, the authors conduct extensive simulations that demonstrate the efficiency of their algorithm in diverse conditions. These simulations provide valuable insights into the algorithm's performance metrics and its ability to handle real-world complexities. Despite the promising results, the authors acknowledge certain limitations in their work. They note the vulnerability of the proposed setting to cyberattacks, highlighting the need for additional security measures to mitigate potential threats. Moreover, they recognize that their study does not extend the application of the optimization to more realistic and advanced environments. This omission indicates an area for future research where further refinement and testing could enhance the algorithm's applicability and robustness in even more challenging and dynamic settings.

Authors in [25] present a comprehensive approach to designing and implementing a privacy-preserving health forecasting system. Their work focuses on exploring efficient learning models suitable for edge devices, aiming to develop a novel time-series forecasting system for user health data streams. This system demonstrates superior efficiency compared to the existing state-of-the-art LSTM-based counterparts. A key aspect of their approach involves the use of Spiking Neural Networks (SNNs) for time-series forecasting. The proposed forecasting model is integrated into an end-to-end health data stream forecasting system leveraging clustered FL. This integration aims to enhance the privacy and efficiency of the health data forecasting process. To validate their model, they conduct a thorough comparison between the SNN model and its LSTM-based counterpart. They evaluate the models based on criteria such as model size, number of trained parameters, and training time. These evaluations are carried out using an augmented real-world Fitbit dataset. The experiments are repeated multiple times to ensure the reliability of the results, with average outcomes computed for each configuration. However, the SNN-based model must be compared with other state-of-the-art machine learning techniques to establish its relative performance and efficiency.

Researchers in [26] propose a novel federated learning framework named FedEx. They conduct FL experiments on two public datasets, utilizing the Dirichlet method to create the datasets. In their experiments, they develop a bi-level optimization algorithm to enhance the performance of FedEx. The performance evaluation of FedEx is carried out using a simulated network environment and standard public datasets. Notably, accuracy may be impacted by device variability and non-IID data, and FedEx only supports distributed machine learning in settings without communication infrastructure.

Researchers in [31] propose an Energy-aware Multi-Criteria Federated Learning (EaMC-FL) model, which addresses the critical issues of communication in FL. Their work delves into controlling and optimizing the communication processes within FL to enhance the sustainability of the system. In their research, the MHEALTH dataset is utilized to simulate a classification problem for HAR. This dataset serves as a practical example to investigate the balance between the clients' local model performance and the energy consumed. By focusing on these trade-offs, the model aims to optimize the overall performance of the system. To facilitate this analysis, experimental datasets are built for all clients. A significant part of their study involves comparing the traditional FedAvg algorithm with the proposed EaMC-FL algorithm. This comparison is based on the average budget across several evaluation scenarios, highlighting the strengths and potential improvements offered by the EaMC-FL model. However, the evaluation of the EaMC-FL algorithm could be extended to much larger scale real-world datasets. This extension is crucial for understanding the scalability and applicability of the model in diverse and more complex environments. By doing so, the researchers aim to demonstrate the potential and robustness of their algorithm in real-world scenarios. About the limitations of this work, the proposed EaMC-FL algorithm is not compared with other state-of-the-art FL models. This comparison serves to benchmark the efficiency of the EaMC-FL model.

Authors in [32] design a distributed learning system with federated learning to support healthcare services. They propose an optimized solution for resource allocation over FL, ensuring that computational and communication resources are used efficiently. The authors formulate an optimization problem that considers the challenges of imbalanced data and the constraints of communication resources. The proposed system is evaluated comprehensively in terms of communication and computation time and latency using real-world datasets. This evaluation helps

to demonstrate the effectiveness of the system in real-world healthcare environments where resource constraints are a significant concern. They assess the performance of the proposed FL framework compared to other state-of-the-art techniques. To further enhance the system's efficiency, the authors could develop algorithms to select a subset of outstanding users that can accelerate the FL process while minimizing communication overheads. This user selection process reduces the overall training time and ensures that the most reliable and fastest nodes contribute to the model updates. Despite the advancements, the framework currently lacks tight synchronization among the nodes, which poses a challenge for its deployment in highly dynamic scenarios. The authors suggest that further research is needed to address this issue, making the framework more adaptable to environments with rapidly changing conditions.

In their study, the authors in [34], propose a novel Dispersed Federated Learning (DFL) framework aimed at optimizing resource utilization. This framework addresses the challenges of resource optimization by formulating an integer linear optimization problem, with the primary objective of minimizing the federated learning cost within the DFL framework. The authors decompose this optimization problem into two distinct sub-problems: the association problem and the resource allocation problem. The association problem focuses on the optimal assignment of learning tasks to different nodes. This task allocation is crucial for maintaining balance and preventing any single node from becoming a bottleneck, thus enhancing the overall system efficiency. Meanwhile, the resource allocation problem deals with the optimal distribution of available resources among these nodes. Furthermore, the study could delve into the application of this resource optimization strategy in real-world scenarios, particularly within the context of smart industry. By addressing both the association and resource allocation problems, the framework ensures that the distributed learning process is both cost-effective and highly performant.

The authors in [46] address the need for efficient machine learning model training in the context of FL. They propose an approach that harmonizes cost optimization with strategic model selection. The authors introduce a model selection method that prioritizes local model uploads, thereby addressing substantial challenges associated with data gathering, processing, and training. The method of model aggregation varies across different algorithms, with federated averaging being a notable example, where an average of all local models is used to create the global model. They introduce a model selection strategy to refine traditional aggregation methods, and significantly

reduce the number of iterations required for training. Moreover, they highlight three key areas for cost reduction in federated learning. Firstly, Algorithm Optimization is addressed through the federated averaging algorithm, which increases the number of local training epochs and sends updates after several iterations. This method effectively reduces the number of communication rounds required, thereby lowering communication costs. Secondly, the authors focus on Encoding and Data Compression. They propose using structured and sketched updates to compress local updates before transmission. This compression technique decreases the amount of data that needs to be sent, further reducing communication expenses. Lastly, Decentralized Training is emphasized through the implementation of a layered architecture. This approach utilizes an edge server positioned between the cloud and the clients. The edge server aids in expediting model convergence and reducing communication costs by managing a portion of the computational load locally.

The study analyzes the cost in FL, emphasizing the importance of modeling and analyzing costs to ensure client participation. This includes time cost analysis, determining the optimal number of local iterations, and optimizing CPU frequency and the amount of data exchanged. They propose a data allocation strategy that aims to maximize the total amount of data contributed by clients during training. The Particle Swarm Optimization (PSO) algorithm is chosen to solve the problem of allocating the optimal amount of data to be exchanged. The article identifies limitations, such as the potential production of inferior models by some clients due to small datasets or transmission errors. Incorporating such subpar models into the aggregation process can slow down convergence rates and increase costs. To mitigate this, the research introduces a selective model integration strategy, incorporating only high-quality models into the aggregation process. The research focuses on improving the FL algorithm. The authors assess the efficacy of their approach and compare it with other algorithms like CMFL, which is a different algorithm in the realm of federated learning. Despite these advancements, the article did not explore further optimization, highlighting an area for future research. While the study primarily focuses on cost reduction in federated learning, such as minimizing communication rounds and compressing data to reduce resource consumption, our work goes beyond this by adopting a multi-objective optimization approach. We balance efficiency with the need for high precision in tasks like symptom detection or diagnostics, where accuracy is critical.

The authors in [47] address the challenge of optimizing communication overhead in federated learning by modeling it as a multi-objective problem. In their work, they employ the MNIST

dataset to simulate a server-client architecture involving four clients, where both fully connected and convolutional neural network models are trained. The communication overhead in federated learning is formulated as a bi-objective optimization problem, with the two key objectives being to minimize communication overhead while maximizing model accuracy. To solve this multi-objective problem, the authors apply the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). This algorithm is particularly effective in exploring the trade-offs between the two competing objectives, enabling the generation of a Pareto front that represents a set of optimal solutions. While their approach shows promise, the authors suggest that future work could involve testing the proposed solution on physically distributed devices and larger benchmark datasets, allowing for further validation and refinement of the results.

In their work, the authors in [48] propose a novel approach to optimize both data transfer and convergence time within the context of federated learning applied to a face recognition task. Their method aims to balance the trade-off between reducing training time and maintaining or enhancing model accuracy. To achieve this, they employ the NSGA-II optimization algorithm and assess the recognition accuracy of their proposed solution in comparison to other existing algorithms. While the study demonstrates the potential for improved performance, the authors suggest that integrating additional data sources and exploring alternative optimization algorithms could further enhance the accuracy and efficiency of the system. However, certain aspects remain unexplored, such as the scalability of the system under various network conditions and configurations, as well as the system's robustness against potential data tampering. Additionally, accuracy is still impacted by device and data variability. These limitations highlight areas for future research and optimization.

In their study, the authors in [49], focused on optimizing the structure of the global model in federated learning to reduce communication overhead while preserving model accuracy. They conducted experiments using the MNIST dataset, setting the batch size and training epochs to 50 and 5, respectively, and defining five communication rounds. The experiments were carried out under different model structures, including both Multi-Layer Perceptron (MLP) and Convolutional Neural Network, to evaluate performance. To achieve optimization, the authors employed the MOEA/D and NSGA-II algorithms. Through these optimization techniques, they were able to identify a set of Pareto-optimal solutions that effectively reduced model complexity without significantly impacting accuracy. However, the study had certain limitations. Due to constrained computing resources, the authors were forced to limit the values of some key parameters, such as

the number of communication rounds and the number of participating clients. Additionally, the model structures used in the experiments were relatively simple, which may have limited the generalizability of the results to more complex models. These constraints point to potential areas for further exploration in future research.

In [50], a robust method for HAR is presented, with a strong focus on maintaining user privacy throughout the process. The method leverages LSTM networks for both server and client-side model generation, as LSTMs are well-suited for capturing temporal dependencies in sequential data. For the model evaluation, the dataset was split into two subsets, with 70% allocated for training and 30% reserved for testing. The impact of the number of epochs was examined, with values increasing from 3 to 10 across five communication rounds and three clients. For the limitations, the authors could utilize advanced LSTM architectures, such as stacked and bidirectional LSTMs, to effectively model complex temporal relationships and detect intricate activity patterns. The investigation did not include adaptive methods that could dynamically adjust parameters like the number of communication rounds or learning rates to optimize the training process and improve accuracy. Additionally, feature engineering techniques that could extract more relevant features from raw sensor data were not explored in this work.

2.5 Research Challenges and Issues

This section highlights the key challenges and obstacles encountered in optimizing federated learning for healthcare systems. We summarize the findings from our literature review and categorize them based on their weaknesses and similarities in Table 2.1. The main limitations identified include optimizing time efficiency, improving accuracy, reducing the amount of exchanged data, and enhancing data privacy in federated learning.

Table 2.1 Summary of the literature review on federated learning in health monitoring.

Proposed Method	Limitations
IoMT-based smart healthcare and wearable frameworks for elderly and general healthcare [36],[20],[22],[35]	Struggles with accuracy due to heterogeneous data sources. Needs to analyze information from more various modalities, does not explore the challenges and requirements of different healthcare domains
Optimization algorithms and resource allocation in federated learning systems [23][24][31][32][34][36][47][48][49]	Lack of data privacy, ignoring optimal timing for FL process, no tight synchronization among nodes, comparison with other state-of-the-art models needed, overlook real-time performance and scalability, unexplored Further optimization in more dynamic scenarios, increased cost, accuracy impacted by device and data variability
Privacy-preserving federated learning algorithms for health monitoring and forecasting [4][25]	Ignoring communication burden, performance affected by data quality of devices, comparison with other state-of-the-art ML techniques needed.
Accurate activity monitoring and HAR using optimized federated learning [22][28][29][30][33][35]	Lack of data privacy, ignoring local training processing time, requires active learning to facilitate future data collection in the most critical data, potential accuracy issues due to diverse sensor data quality.
Evaluation tools and simulation for federated learning [26]	Supports distributed ML in limited scenarios, accuracy may vary depending on the simulation's ability to emulate real-world conditions, and accuracy could be impacted by non-IID data.

Optimizing Time Efficiency:

Optimizing time efficiency in the federated learning process is crucial to enhance the overall performance of the system. Federated learning involves multiple clients performing local computations and then communicating updates to a central server. This iterative process can be time-consuming, especially when dealing with large-scale data and complex models. One key factor that affects time efficiency is the duration of local computations. The time taken for each client to complete its local training can vary significantly depending on the computational resources available. Ensuring that clients have optimized local training procedures is essential to reduce overall training time. Communication latency between clients and the server is another significant factor, and the round-trip time for data transmission can introduce delays. Additionally, the synchronization of clients plays a vital role in determining the speed of the federated learning process. The authors in [42] discuss how asynchronous communication methods can help mitigate delays caused by waiting for all clients to synchronize. By allowing clients to operate independently and update the server asynchronously, the overall training process can be expedited. Understanding these factors and their implications on time efficiency is vital for developing faster and more practical federated learning systems.

Improving accuracy:

In federated learning, achieving high accuracy remains a significant challenge due to the inherent complexities of working with distributed data and a variety of participating devices. The data utilized in FL systems is often non-IID (non-Independent and Identically Distributed), meaning that data characteristics vary widely across different clients. This diversity can lead to a phenomenon called model drift, where the global model struggles to generalize across diverse datasets. Consequently, the aggregated model may have difficulty capturing the unique patterns in each client's data, which can decrease overall accuracy as it fails to effectively integrate these nuances. Additionally, the variability in device capabilities poses further obstacles to accuracy. Devices participating in the network may differ significantly in terms of computational power and processing capacity. These disparities can impact the quality of local models produced by each device, as lower-capacity devices may not be able to train as effectively. When aggregated, these varied local models can lead to a decline in the global model's accuracy. Moreover, differences in

data quality among devices can introduce further inconsistencies, making it challenging to sustain high accuracy across the distributed network [51][52].

Optimizing the amount of data exchanged:

Optimizing the amount of exchanged data is crucial in federated learning to ensure efficient communication between clients and the central server. Performing local computations and sending updates to a server can put a significant strain on the FL network, especially when dealing with large datasets and complex models. One major challenge is the massive amount of data being exchanged between clients and the server. The authors in [43] highlight that transmitting model updates from numerous devices can quickly result in a large amount of data transfer. This is particularly problematic in environments with limited network resources, where high data traffic can lead to network congestion and slow down the entire system. As noted by the authors in [44], another important consideration is the frequency of communication between devices and the server. The frequency of updates can significantly impact the amount of data transferred. Frequent communication can overwhelm the network, causing delays and affecting the efficiency of the federated learning process.

Understanding these challenges is essential for making federated learning systems more efficient in terms of the amount of data transferred. Acknowledging these complexities helps ensure that federated learning can be implemented effectively, even in environments with limited network resources.

Enhancing data privacy:

Enhancing data privacy is a top priority in federated learning, where sensitive information from multiple users is used to train machine learning models without being directly shared. In this process, each device performs local computations and only shares model updates with a central server, not the actual data. The diverse and decentralized nature of federated learning introduces challenges in maintaining consistent privacy standards across all participating devices. Authors in [45] note that the varying levels of security on different devices can make it difficult to guarantee the same level of data protection for all users, underscoring the need for robust privacy measures throughout the system.

CHAPTER 3 PROPOSED OPTIMISED FEDERATED LEARNING ARCHITECTURE

In this chapter, we introduce the architecture of our proposed model aimed at enhancing the efficiency of federated learning within a remote patient monitoring system. We then provide an in-depth analysis of each parameter targeted for optimization and discuss their influence on the overall performance of the federated learning process. Additionally, we explore how recent advancements and emerging technologies have contributed to shaping and improving our optimization approach.

3.1 Data Collection and Processing

In this section, we introduce the dataset used in our research. The MHEALTH dataset was selected to develop and evaluate the federated learning model in the remote patient monitoring system. The MHEALTH dataset was designed to serve as a benchmark for developing and evaluating techniques aimed at human activity recognition through multimodal body sensing. This dataset is particularly relevant for studies in computer science that focus on multivariate time-series data and classification tasks. The MHEALTH dataset captures a wide range of physiological and motion-related features. It consists of recordings from ten volunteers while they engaged in various physical activities, as shown in Table 3.1. Each participant's body motion and vital signs were recorded using wearable sensors placed on three specific body locations, which are the chest, right wrist, and left ankle, while performing 12 distinct physical activities, as illustrated in Table 3.2. The dataset can be used to develop and evaluate advanced models for activity recognition and human behavior analysis [55].

Table 3.1 Number of rows in dataset log files.

Log file	Number of data rows
1	161280
2	130561
3	122112
4	116736
5	119808
6	98304
7	104448
8	129024
9	135168
10	98304

Table 3.2 Dataset classes.

No.	Activity
1	Climbing stairs
2	Cycling
3	Frontal Elevation of Arms
4	Jogging
5	Jump Front and Back
6	Lying down
7	Running
8	Sitting and Relaxing
9	Standing
10	Waist bends Forward
11	Walking
12	Knees bending

Additionally, the structure of the MHEALTH dataset is designed to capture various types of physiological data, including motion, rotational movement, and orientation. These measurements are recorded across three axes, along with labels that indicate the specific activity being performed, as illustrated in Figure 3.1. Although the dataset includes additional columns related to heart

function, such as 2-lead ECG data, these were excluded from our analysis as they were collected for future investigations into the effects of exercise on heart function. For this study, we focused solely on the motion-related features [55].

```
Column 1: acceleration from the chest sensor (X axis)
Column 2: acceleration from the chest sensor (Y axis)
Column 3: acceleration from the chest sensor (Z axis)
Column 4: acceleration from the left-ankle sensor (X axis)
Column 5: acceleration from the left-ankle sensor (Y axis)
Column 6: acceleration from the left-ankle sensor (Z axis)
Column 7: gyro from the left-ankle sensor (X axis)
Column 8: gyro from the left-ankle sensor (Y axis)
Column 9: gyro from the left-ankle sensor (Z axis)
Column 10: magnetometer from the left-ankle sensor (X axis)
Column 11: magnetometer from the left-ankle sensor (Y axis)
Column 12: magnetometer from the left-ankle sensor (Z axis)
Column 13: acceleration from the right-lower-arm sensor (X axis)
Column 14: acceleration from the right-lower-arm sensor (Y axis)
Column 15: acceleration from the right-lower-arm sensor (Z axis)
Column 16: gyro from the right-lower-arm sensor (X axis)
Column 17: gyro from the right-lower-arm sensor (Y axis)
Column 18: gyro from the right-lower-arm sensor (Z axis)
Column 19: magnetometer from the right-lower-arm sensor (X axis)
Column 20: magnetometer from the right-lower-arm sensor (Y axis)
Column 21: magnetometer from the right-lower-arm sensor (Z axis)
Column 22: Label (0 for the null class)
```

Figure 3.1 Dataset structure.

3.1.1 Data Preprocessing

In preparation for the machine learning model used in federated learning architecture, multiple preprocessing techniques were applied to ensure the consistency, balance, and suitability of the MHEALTH dataset for time-series classification.

- **Data Cleaning**: To address potential inconsistencies, column names were stripped of extraneous whitespace, as suggested by common data cleaning practices [56]. This step ensures clarity and prevents issues during later stages of model training.

- **Data Resampling**: A key aspect of preparing the dataset for model training involved addressing the issue of class imbalance. Specifically, the dataset exhibited a significant overrepresentation of the null activity class, labeled as activity 0. This imbalance can make

the model favor the majority class, making it less accurate at predicting the minority classes. Recent studies emphasize that class imbalance, if left untreated, can severely impact the overall performance of machine learning models, particularly in multi-class classification tasks. In our dataset, activity 0, representing periods of no activity, appeared far more frequently than other activity classes. This overabundance of the null activity neglects other, more meaningful activity types. To resolve this, we applied a well-established technique called random down sampling. Random down sampling is a method widely used in handling imbalanced datasets by reducing the size of the majority class [57].

- **Data Splitting**: the dataset was split into the predictor and the output variables. The target variable, activity, represents the class label, while the predictor variables are the features used to predict these activity classes. Splitting the data into features and target variables is a foundational step in any machine learning process, as it ensures that the model is trained on input data to predict the desired output. The dataset was divided into 75% for training and 25% for testing, which is a commonly accepted ratio in machine learning for model evaluation. This split ensures that the model is trained on a large portion of the data while still allowing for testing on separate data to assess the model's performance.
- **Data Scaling**: Next, we used the standardization method, which scales the data to have a mean of 0 and a standard deviation of 1, ensuring that all features are on a similar scale. The standardization method is crucial for deep learning models because it helps the model converge more quickly and accurately by ensuring that all features contribute equally, without any one feature dominating due to its larger scale [58]**.**
- **One-hot encoding**: Finally, the target variable was converted to a categorical format using one-hot encoding, a technique that transforms the class labels into binary vectors. This transformation is important for multi-class classification problems, as it allows the model to learn from each class independently, improving its predictive accuracy across different categories [59].

These preprocessing steps are crucial in ensuring that the data is ready for machine learning techniques. They follow well-established techniques that enhance the model's learning ability and improve its generalization on unseen data.

3.2 System Architecture

We propose an innovative federated learning framework based on Flower in which the federated learning system is designed to ensure efficiency in healthcare services. The architecture focuses on the optimization of three parameters: accuracy, time, and amount of data transferred, as shown in Figure 3.2.

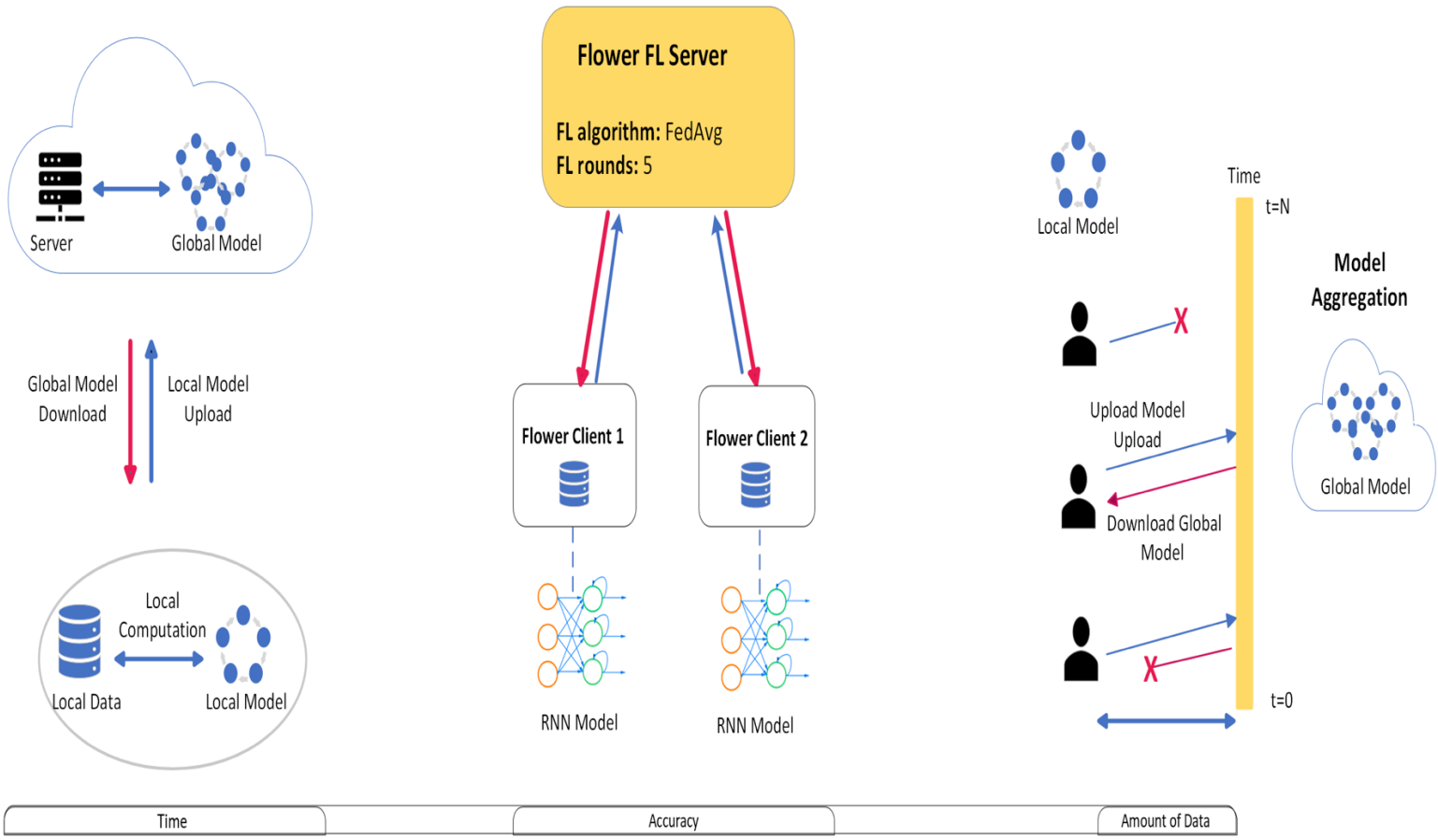

Figure 3.2 Proposed FL architecture.

In this section, we propose a federated learning model designed to deliver an efficient and accurate federated learning process for health monitoring by integrating advanced machine learning techniques. The proposed architecture centers around a federated learning framework, which enables collaborative model training across two decentralized nodes. In this setup, an FL server

coordinates the training process by exchanging models with clients, each containing local data. In this system, each node first downloads the global model from the FL server, trains it locally using its dataset, and subsequently uploads the updated local model back to the server. The server then aggregates these updates to refine the global model, which is then redistributed to all participating nodes for more improvement. The timeline on the right side of the figure shows how the server and nodes synchronize at specific time intervals. On the right side of Figure 3.2, The timeline shows how the server and clients synchronize at specific time intervals. Moreover, if there is a high amount of data load, the models may not transfer, so we need to optimize the amount of data being transferred. Several preprocessing steps were carried out to prepare the input data for further analysis. The preprocessing steps include data cleaning, resampling, splitting, scaling, and one-hot encoding. This ensures the input data is standardized and ready for analysis. A BiLSTM network processes input data in both forward and backward directions, allowing the model to consider both preceding and following context within time series data. This bidirectional approach is particularly valuable for accurately identifying physical activities. Batch Normalization (BN) is also used to help more standardization of the output of the BiLSTM layer. Additionally, the dense layers in the network refine the outputs of the model, enhancing its ability to make precise classifications, thereby improving the overall effectiveness of the system [63][64]. In our setup, the FL process is repeated over five communication rounds, and the FL strategy used is federated averaging. We utilize Flower, an innovative federated learning framework, in our FL architecture. Besides the BiLSTM, which achieved the highest model accuracy, we also experimented with an LSTM architecture to further explore time and bandwidth optimization. This LSTM model is less complex than the BiLSTM model.

3.3 Recurrent Neural Networks in Time-series Data Analysis

In this section, we introduce the proposed RNN neural network model and its concepts, along with the preprocessing techniques applied to prepare the data as model input. In recent years, RNNs have become increasingly popular for time-sequential applications due to their unique ability to model relationships over time. Unlike traditional deep learning models, which process data as independent inputs, RNNs can handle sequences, providing a distinct advantage in predictive tasks where context and historical information are key. Activities that unfold over time often carry a temporal dimension, meaning that the sequence and timing of events are crucial for accurate

recognition. This characteristic makes RNNs particularly well-suited for such tasks because the output of an RNN depends on prior elements within the sequence, allowing it to remember past inputs, as shown in Figure 3.3. This ability to loop the sequence back into the network as an input offers discriminative power over other deep learning approaches, making RNNs particularly effective in domains such as healthcare [60].

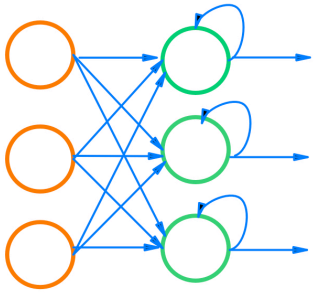

Figure 3.3 RNN structure.

For this study, RNN is adopted due to its capability to handle sequence-based information. This model has recurrent connections within its hidden units, which allows it to maintain relationships between past and present states effectively, embedding a form of memory into the network.

3.3.1 Long Short-Term Memory

LSTM is a specialized type of recurrent neural network architecture that excels at capturing and representing sequential data, such as time-series information. While traditional RNNs often struggle with retaining long-range dependencies due to challenges like vanishing gradients and other optimization problems, LSTMs were specifically designed to overcome these issues [61]. They achieve this by holding onto information across multiple time steps, which makes them particularly effective for tasks that require remembering past inputs over extended periods. An LSTM cell operates through three primary components known as gates [61][62]:

1. Input Gate: The input gate determines which new information should be incorporated into the cell's memory. It assesses the relevance of the current input along with the hidden state

from the previous time step. Based on this evaluation, the input gate updates the cell state, allowing only the most pertinent new information to be added for further processing.

$$i_t = \sigma\,(W_i * [h_{t-1}, x_t) + b_i\,) \tag{3.1}$$

$$C_t = \tanh\,(W_c * [h_{t-1}, x_t] + b_f\,) \tag{3.2}$$

2. Forget Gate: The forget gate determines which information from previous steps should be discarded. It uses the current input and the hidden state from the last time step to produce a value between 0 and 1 for each piece of information. A value of 0 indicates that the information should be forgotten, while a value of 1 means it should be retained for further processing.

$$f_t = \sigma\,(\,W_f * [h_{t-1}, \chi_t] + b_f\,) \tag{3.3}$$

f_t is the forget gate vector at time step *t*.

σ is the sigmoid activation function, which makes the output values range between 0 and It plays a vital role in determining how much information should be preserved and how much should be omitted.

W_f is the weight matrix for the forget gate and $[h_{t-1}, \chi_t]$ is the concatenation of the previous hidden state h_{t-1} and the current input χ_t. It merges the data from the prior time step with the current input.

b_f is the bias term for the forget gate.

3. Output Gate: The output gate determines which information from the current cell state should be passed on to the next hidden state. It filters and processes the cell's data to produce the current output, ensuring that only the most relevant information is carried forward in the sequence.

$$o_t = \sigma(W_0 * [h_{t-1}, x_t] + b_0) \tag{3.4}$$

$$h_t = O_t * \tanh(C_t) \tag{3.5}$$

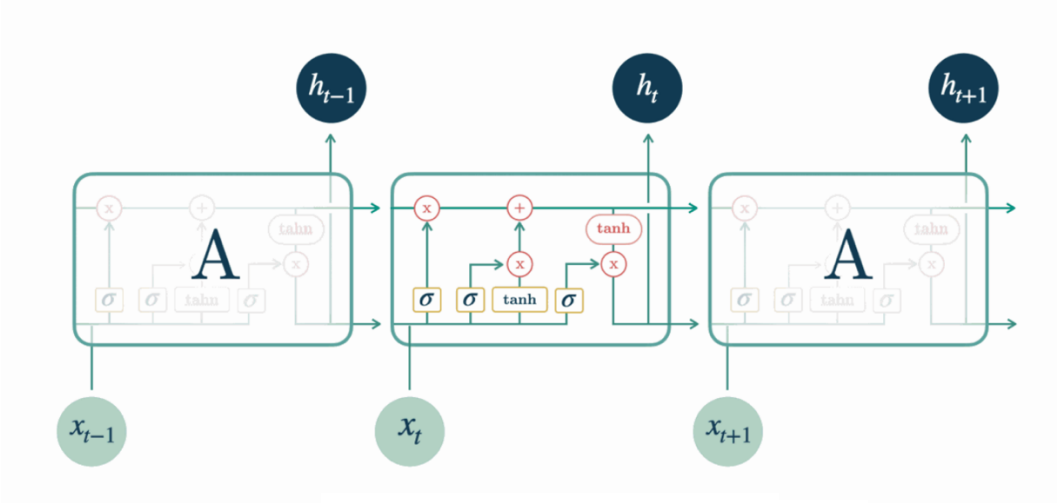

Figure 3.4 LSTM architecture [65].

Cell State: The cell state is updated through the combined actions of the forget and input gates. First, the output of forget gate is multiplied by the previous cell state, deciding what information to retain. Then, this product is added to the output of input gate. The sum is then used to calculate the hidden state within the output gate.

$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t \tag{3.6}$$

The LSTM architecture resembles that of an RNN, but it replaces the standard feedback loop with an LSTM cell. In each layer, a sequence of LSTM cells processes information, with each cell receiving the output from the previous one. This allows the cell to retain prior inputs and sequence details. Within each LSTM cell, a series of cyclic steps occur [61[[62]:

- The Input gate value is determined.
- The Forget gate is calculated.
- The Cell state is updated based on these two results.
- Finally, the output is computed through the output gate.

3.3.2 Bidirectional Long Short-Term Memory

In this section, we introduce BiLSTM for classification on the MHEALTH dataset. BiLSTM builds on the LSTM model by processing the input sequence in two directions: forward from past to future

and backward from future to past. This bidirectional approach is particularly effective for tasks that benefit from understanding the entire sequence context. The BiLSTM network consists of the following [63][64]:

- **Forward LSTM:** One LSTM layer processes the input sequentially from the beginning to the end of the sequence, moving from the first token to the last.
- **Backward LSTM:** Another LSTM layer handles the same input but in reverse, starting from the last token and moving back towards the first. This backward pass allows the model to incorporate future context that may impact the interpretation of current action.
- **Outputs:** The outputs from both the forward and backward LSTM layers are then combined. By integrating information from the entire sequence, the model gains a comprehensive view that enhances its understanding of the sequence.

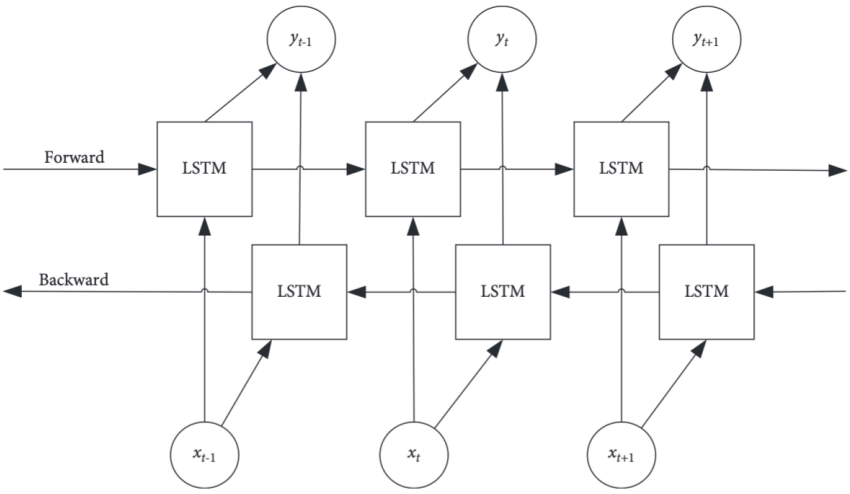

Figure 3.5 Bi_LSTM architecture [66].

We created two RNN model architectures, the first and main proposed model architecture, which has the highest accuracy we achieved, is a BiLSTM model, while the second model architecture is an LSTM model. The second model architecture, which is much less complicated than the first

one, was created to introduce more variety in the dataset that we will create, aiming for better optimization results across all three parameters: time, data amount, and accuracy.

3.3.3 Batch Normalization Layer

Batch Normalization (BN) is a technique widely employed in neural network models to standardize the inputs of each layer. This normalization process adjusts the inputs so that their mean is close to zero and their standard deviation is close to one. By doing so, batch normalization can significantly expedite the convergence of neural networks, as it helps to stabilize the learning process. The output distributions of each layer can vary considerably throughout training. Such discrepancies increase the model's computational demands, as each layer needs to handle diverse distributions of input data. Consequently, the network can become more intricate, making it vulnerable to overfitting and slower convergence. Thus, batch normalization not only enhances training efficiency but also contributes to the overall robustness and generalizability of neural network models [67]. Following each layer, Batch Normalization adjusts the feature distribution to maintain a mean of zero and a variance of one. This normalization process has a noteworthy effect on the activation functions within the network. For instance, certain activation functions, such as the sigmoid, can encounter issues like the vanishing gradient problem, especially when inputs approach the function's extreme values. This problem arises when gradients become very small, hindering the model's ability to learn effectively during training. BN helps to address this by ensuring that the features remain within a range where the gradients are sufficiently large, which helps in maintaining gradient magnitude. As a result, BN not only supports the preservation of useful gradients but also accelerates the convergence process. This stabilization ultimately enhances the reliability and speed of training, which is particularly valuable for deep neural networks. It also means that the learning convergence speed is faster so that the speed and stability of training can be greatly improved. The formula is as follows [71][72]:

For an output *x* of a given layer, BN normalizes the data:

$$\hat{x}_i = \frac{x - mean}{\sqrt{variance + \epsilon}} = \frac{x_i - \mu_B}{\sqrt{Var_{x_i} + \epsilon}} \tag{3.7}$$

Here, ϵ is a small constant added for numerical stability, ensuring the denominator does not approach zero.

Finally, the normalized value is scaled and shifted using two parameters, γ and β, which are learned during training.

$$y_i = \gamma_{\hat{x}_i} + \beta \tag{3.8}$$

3.3.4 Dense Layer

A dense layer, also referred to as a fully connected layer, serves as a fundamental component in neural networks. In this layer, each neuron is linked to every neuron in both the preceding and following layers, enabling the model to capture complex patterns within the data [68].

$$y = \sigma(W * x + b) \tag{3.9}$$

In this context, x represents the input vector, while W denotes the weight matrix that links the input neurons to the output neurons. The term b stands for the bias vector. The symbol σ refers to the activation function, such as ReLU or sigmoid, which incorporates non-linearity into the model. By introducing non-linearity, the activation function allows the model to learn complex, non-linear relationships between inputs and outputs.

3.4 Flower Framework

Flower is an innovative federated learning framework designed to support both algorithmic and systems-related challenges in FL [73]. As an open-source project under the Apache 2.0 License, Flower has been widely adopted by leading research institutions across academia and industry. This framework provides a robust language and machine learning framework, enabling researchers to implement and experiment with new ideas. Flower allows a rapid transition of existing ML training into an FL setup to evaluate their convergence properties and training time in a federated setup. Flower is also fully extendable and can incorporate emerging algorithms and training strategies. Flower is widely applicable across various federated learning settings and believed to

have the potential to be transformative in narrowing the gap between federated learning research and practical, real-world systems [73].

3.4.1 Federated Averaging: The training strategy that we use in our Flower federated learning framework is the FedAvg algorithm [77]. FedAvg is arguably the most popular and effective FL algorithm that figures out the global model by averaging the model updates from all the participating clients. Mathematically, it can be explained as the updated global model W_{t+1} at time t + 1 is obtained by averaging the model parameters from all the clients participating in FL [74]:

$$W_{t+1} = \frac{1}{N}\sum_{i=0}^{N} W_{i,t} \tag{3.10}$$

Where N is the total number of participating clients and $W_{i,t}$ represents the model parameters from client i at time t.

3.5 Multi-Objective Optimization

A significant portion of typical modern machine learning procedures involves hyperparameter optimization This arises from the fact that machine learning techniques and corresponding preprocessing steps often only produce optimal performance when hyperparameters are properly tuned. However, in many applications, we are not only interested in optimizing ML pipelines solely for predictive accuracy; additional metrics or limitations must be considered while figuring out the best configuration, creating a multi-objective optimization problem. For many ML uses, single-objective tasks that only measure prediction performance are no longer up to date, because models now have to meet certain standards for other goals as well. Experts in machine learning will have to deal with new problems when it comes to optimization and choosing algorithms. However, the right approaches can give the user a set of optimal choices to help them choose a good model [76].

3.5.1 Pareto Optimality Approach

Pareto optimization, a concept introduced by the engineer and economist Vilfredo Pareto, has been foundational in the study of multi-objective optimization [75]. Pareto discovered that solutions to

problems involving multiple objectives could be organized in a partial order without requiring any specific preference choices beforehand. The solutions that arise from applying Pareto's principle are now known as Pareto-optimal solutions. This concept implies that a solution is considered Pareto-optimal if there is no other feasible solution that is superior. In other words, a Pareto-optimal solution cannot be improved upon in any objective without compromising at least one other. In a Pareto optimization framework, you try to balance these objectives. When dealing with conflicting objectives, the result is not a single optimal solution but a set of solutions that collectively form a surface. This surface, known as the Pareto front or Pareto frontier, represents all the possible optimal trade-offs among the objectives. Pareto optimization thus provides a framework for identifying these solutions while leaving the task of selecting a final compromise to the decision-maker [69][70].

An array of several objectives $c \in \mathbb{R}^m$(Pareto) dominates another array of multiple objectives $\acute{c}$,

Written as $c < \acute{c}$, if and only if [22]:

$$\forall_i \in \{1, \dots, m\}: c_i \leq \acute{c}_i \ And \qquad (3.11)$$
$$\exists_j \in \{1, \dots, m\}: c_j < \acute{c}_j$$

m is the number of objectives, $\mathbb{R}^m$ is the objective space of the problem, and $c_i, i \in \{1\ , 2\ , \dots, m\}$ are objective points.

Maximizing an objective function is equivalent to minimizing its negative.

After conducting several experiments on the proposed FL framework by modifying hyperparameters and testing both the LSTM and BiLSTM models, we collected data on the parameters of accuracy, time, and data volume during the FL process. We then applied Pareto optimality to the collected dataset to optimize these parameters.

3.6 Summary

This chapter presented our proposed Federated Learning model for a remote patient monitoring system. The architecture includes three primary components: accuracy, time, and amount of data transferred. Data cleaning, data resampling, splitting, scaling, and one-hot encoding were applied to prepare the input data for the model.

In the training phase, we applied the RNN neural network to classify physical activities with high accuracy. We tested different hyperparameters and applied two distinct RNN architectures on our Flower federated learning system to collect data on training time, accuracy, and data amount which was used to generate the dataset. Then, we applied multi-objective Pareto optimization on our dataset to develop a set of Pareto optimal solutions that trade-off between maximizing accuracy and minimizing time and amount of data transferred.

CHAPTER 4 IMPLEMENTATION AND RESULTS

In Chapter 3, we presented the architecture of the proposed Federated Learning model for enhancing patient health monitoring within the healthcare system, emphasizing its three primary optimization focuses accuracy, amount of data transferred, and time optimization. In this chapter, we present the implementation of the FL model to evaluate its performance. We present our testbed setup and configuration of the testing environment. In addition, we present our findings and results about the proposed architectures and the augmented dataset we created to apply multi-objective optimization to it. Moreover, we demonstrate the efficiency of the proposed neural network architecture for classification in human activity recognition and the multi-objective optimization method for identifying optimal solutions in FL optimization.

4.1 Testbed

Our testbed was deployed on a machine using Visual Studio Code (VSCode). VSCode was selected because of its adaptability and support for extensive federated learning and machine learning frameworks and libraries, making it suitable for handling the distributed machine learning and data-intensive workloads required by this project, as follows:

- Development environment: The VSCode IDE enabled the use of Jupyter, which is an interactive development environment for notebooks, code, and data. The development of machine learning and data mining processes is increasingly carried out on Jupyter notebooks, mainly because of the reproducibility and the advantages and convenience in visualization [78].
- Software Specifications:
  a) Programming Language: We chose Python language because of its several types of FL frameworks like Flower and TensorFlow Federated [79].
  b) Operating System: We utilized the macOS operating system, which provided a stable environment for the federated learning frameworks to function and VSCode with add-ons to connect to the additional features available for testing and developing models.

  The testbed environment for all testing was carried out on a MacBook Pro ((14-inch, 2023) with macOS Sonoma (Version 14.1). The system's specifications are as follows:

- **Processor**: We used an Apple silicon M3 Pro chip for our processor, which has enough computing power to execute the federated learning task and model evaluations.
- **Memory**: The system has 36 GB of RAM, which allows us to analyze data and make model training operations easier and faster.
- **Storage**: Our storage had a 994.66 GB SSD, which enabled swift data access and run operations, which are crucial for handling large datasets and rapid model iterations.

We implemented the FL architecture using Python code in VSCode, running locally on a macOS system. This architecture processes medical data and integrates multiple components of federated learning within a health monitoring system to classify physical activities and optimize the learning process.

4.2 Selection of the Dataset

In Chapter 3 (cf. Chap. 3.1), we presented our selected dataset for training the model. We utilized a large dataset called the MHEALTH dataset. This dataset covers various physical activities and has extensive data based on multimodal body sensing. In the MHEALTH dataset, the data collected for each subject is stored in a different log file. The activity class or label is represented by the final column, "activity," which is one of the 22 columns in each file and the other columns contain sensor data. Figure 4.1 illustrates the initial rows and shape of the MHEALTH dataset for a single sample log file [48].

```
Data Head:
    acc_ch_x  acc_ch_y  acc_ch_z  acc_la_x  acc_la_y  acc_la_z  gyr_la_x  \
0   -9.8184  0.009971   0.29563    2.1849   -9.6967   0.63077  0.103900
1   -9.8489  0.524040   0.37348    2.3876   -9.5080   0.68389  0.085343
2   -9.6602  0.181850   0.43742    2.4086   -9.5674   0.68113  0.085343
3   -9.6507  0.214220   0.24033    2.1814   -9.4301   0.55031  0.085343
4   -9.7030  0.303890   0.31156    2.4173   -9.3889   0.71098  0.085343

   gyr_la_y  gyr_la_z  mag_la_x  ...  acc_rw_x  acc_rw_y  acc_rw_z  gyr_rw_x  \
0  -0.84053  -0.68762 -0.370000  ...   -8.6499   -4.5781  0.187760  -0.44902
1  -0.83865  -0.68369 -0.197990  ...   -8.6275   -4.3198  0.023595  -0.44902
2  -0.83865  -0.68369 -0.374170  ...   -8.5055   -4.2772  0.275720  -0.44902
3  -0.83865  -0.68369 -0.017271  ...   -8.6279   -4.3163  0.367520  -0.45686
4  -0.83865  -0.68369 -0.374390  ...   -8.7008   -4.1459  0.407290  -0.45686

   gyr_rw_y  gyr_rw_z  mag_rw_x  mag_rw_y  mag_rw_z  activity
0   -1.0103  0.034483  -2.35000 -1.610200 -0.030899         0
1   -1.0103  0.034483  -2.16320 -0.882540  0.326570         0
2   -1.0103  0.034483  -1.61750 -0.165620 -0.030693         0
3   -1.0082  0.025862  -1.07710  0.006945 -0.382620         0
4   -1.0082  0.025862  -0.53684  0.175900 -1.095500         0

[5 rows x 22 columns]
Data Shape: (161280, 22)
```

Figure 4.1 MHEALTH dataset structure: sample data and shape.

4.3 Implementation of the RNN Model

In Chapter 3 (cf. Chap. 3.3.2), we explained that the first proposed model for the MHEALTH dataset is the BiLSTM model. In this section, we implement the BiLSTM model which is both an effective and highly accurate model for classifying physical activities. We also discuss our findings for the second model architecture, an LSTM model that resulted in a lower accuracy. All components are implemented using Python programming language.

4.3.1 Preprocessing of the Dataset

In this chapter, we began by loading the dataset, which we obtained as a CSV file. Upon loading, we removed any extraneous whitespace from the column names to ensure adequate access during subsequent analysis. Additionally, we reduced the raw log files to 22 columns by removing two columns related to vital signs, as our focus in this study is on classifying body activities. The Scikit-Learn framework was used to standardize and clean the dataset. The dataset included a substantial amount of data labeled as null activity, which could distort the performance of the model if left unaddressed. To address this, we implemented a targeted resampling strategy. Specifically, we reduced the instances of activity null. Following the resampling process, the dataset was divided

into predictor variables and the target variable. All features except activity were considered predictors, with activity serving as the target variable. To standardize the scale of the predictor variables, we applied a StandardScaler, a preprocessing technique in scikit-learn library, which is crucial in ensuring that all features contributed equally to the learning process. We then split the standardized dataset into training and testing subsets, allocating 75% of the data for training and the remaining 25% for testing. For compatibility with the RNN model, we reshaped the predictor data into a three-dimensional format, with each sample representing a single timestep including all feature inputs, meaning:

- **Batch size**: Number of independent sequences (observations).
- **Timesteps**: `1`, as each sample represents a single time step.
- **Features**: All feature inputs per time step.

We converted the target variable into a categorical format through the One-hot encoding technique, as illustrated in Figure 4.2, to better support the multi-class classification problem and set it up for efficient training and assessment in the ensuing modeling stages. In this plot, each sample only has one active class. The red blocks indicate the specific class each sample belongs to, while the blue background indicates the absence of other classes for each sample.

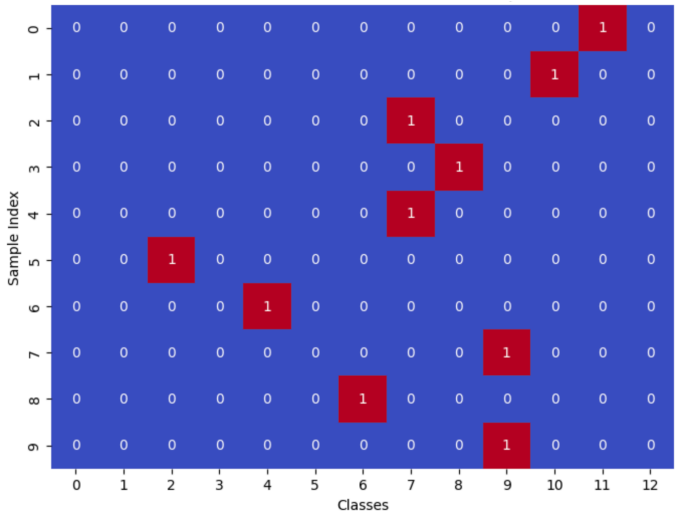

Figure 4.2 One-hot encoded labels - first ten samples.

4.3.2 Model 1 - BiLSTM Model

In this section, we implemented the BiLSTM model as a primary model for recognizing human body activities. The BiLSTM model aims at capturing temporal patterns in the sequential data. The model architecture begins with two bidirectional LSTM layers. The temporal structure is maintained and transferred to the following layer by the 128-unit initial BiLSTM layer, which is intended to return sequences. A dropout rate of 30% is applied after this layer to prevent overfitting by randomly dropping a portion of the neurons during training [80]. The second BiLSTM layer, with 64 units, is configured to return a single output instead of a sequence. A 20% dropout is applied to enhance the model's generalization capabilities. Batch normalization is incorporated after this layer to stabilize and accelerate the training process by normalizing the output of the previous layer. Following this, a dense layer with 256 neurons and a ReLU activation function is used to introduce non-linearity, enabling the model to learn more complex representations of the data [81]. Another dropout rate of 40% is applied at this stage to reduce overfitting risks. Finally, the model's output layer is a dense layer with softmax activation. The softmax function, often used in the final layer of a neural network model for classification tasks ensures that the output is a probability distribution corresponding to the different categories [82]. The model is compiled using the Adam optimizer and categorical cross-entropy as the loss function, which is appropriate for multi-class classification tasks [83]. This BiLSTM architecture is well-suited for sequential data, as it efficiently and accurately learns temporal dependencies and achieves the highest accuracy in this study.

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 1, 256)	153,600
dropout (Dropout)	(None, 1, 256)	0
bidirectional_1 (Bidirectional)	(None, 128)	164,352
dropout_1 (Dropout)	(None, 128)	0
batch_normalization (BatchNormalization)	(None, 128)	512
dense (Dense)	(None, 256)	33,024
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 13)	3,341

Total params: 354,829 (1.35 MB)
Trainable params: 354,573 (1.35 MB)
Non-trainable params: 256 (1.00 KB)

Figure 4.3 Model summary of BiLSTM.

Results from our model are encouraging, as Figure 4.4 and Table 4.1 illustrate. The model is gradually getting better, as we can see from the left graph, where the accuracy for both training and validation has increased, and the right graph, where the corresponding loss for both training and validation has steadily decreased, indicating that the model is learning efficiently while remaining stable.

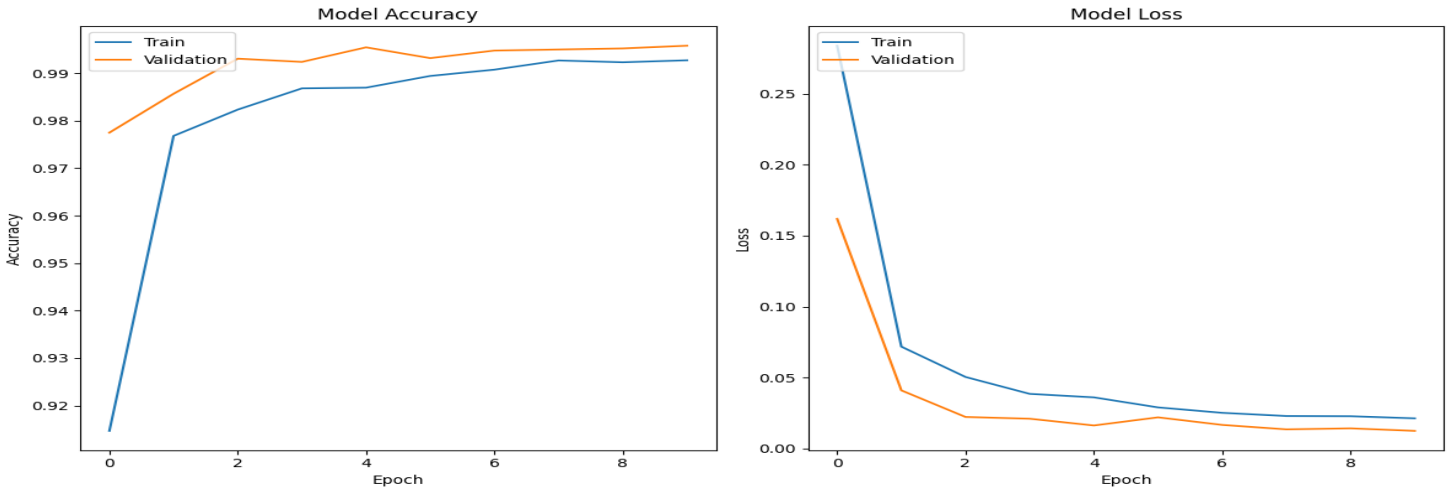

Figure 4.4 The accuracy and loss of BiLSTM.

Table 4.1 Accuracy of the proposed BiLSTM model.

Dataset	Accuracy
Train Data	99.2
Validation Data	99.72
Test Data	99.73

4.3.3 Model 2 - LSTM Model

The second RNN model leverages the LSTM architecture which is useful for tasks where understanding the temporal relationships within data is required. The model begins with an LSTM layer consisting of 64 units, which processes the input data sequentially. The LSTM layer enables the model to retain important information from earlier inputs while selectively forgetting less relevant details, making it ideal for handling dependencies in time series data. Following the LSTM layer, a Batch Normalization layer is applied to help the model converge more quickly. A Dropout layer is then introduced with a dropout rate of 0.2, which randomly disables 20% of the neurons during training to prevent overfitting. Next, a dense layer with 64 units and a ReLU activation function is added. This layer enables the model to learn more representations of the input. Another Dropout layer follows, applying the same dropout rate to further combat overfitting. Finally, the output layer is a dense layer, where the number of units corresponds to the number of classes in the target variable. It uses the softmax activation function, ensuring the output of the model is a probability distribution over the possible classes. The model is compiled with the Adam optimizer and the categorical cross-entropy loss function, which is suitable for multi-class classification tasks. Additionally, an early stopping mechanism is integrated to halt training if the performance on the validation set does not improve for five consecutive epochs. Finally, since the accuracy of the LSTM model was lower than that of the proposed BiLSTM model, we decided to not focus as much on its accuracy output summary. Particularly, it required less training time, and its model parameters have a smaller data volume. This by itself is relevant for the purpose of our study

because we needed to build a dataset of different models running under different settings to collect different characteristics. The LTSM represented a suitable candidate.

4.4 Federated Learning Setup

The federated learning structure is implemented with the FLOWER framework (cf. Chap. 3, §3.4) using a Python custom code. The implementation is divided into two parts server side and client side. This setup consists of two clients and a central server that adapts the model training process. Each client trains the model locally on its own data log file. Once training is complete on each client, the locally updated model parameters are sent to the central server. The server then aggregates these parameters to update the global model. This updated global model is sent back to the clients for local training in the next round. The model improves over multiple iterations, leveraging data from the sources.

4.4.1 Client-Side Implementation

In our federated learning system, the two clients are responsible for local model training and periodically communicating with the FLOWER server to synchronize the parameters of the model. Using the Flower framework, which enables smooth integration between local model updates and global coordination, the client-side script is designed to facilitate these tasks. At the core, the client is initialized with specific configurations, such as the number of training epochs and batch size. The key components of the client side are broken down as follows:

- **Model Parameter Management:** Each iteration begins by receiving the current global model parameters from the server. These parameters are immediately set in the client's local model to ensure that the clients start from the same baseline.
- **Training Phase:** The local model is then trained on the data log file of the client. Throughout the training phase, critical metrics, such as global model accuracy, are tracked to assess performance.
- **Evaluation Phase:** Once training is complete, the client evaluates the model on its local test set. The evaluation phase measures key performance indicators like loss and accuracy, and the results are shared with the central server. This allows the server to assess the effectiveness of the global model based on client evaluations.

4.4.2 Server-Side Implementation

The Flower federated learning server was started with a configuration specifying that five rounds of training would be conducted. No round timeout was set during these rounds, allowing the clients to take as long as needed to complete their tasks. The server initialization is logged, and the process begins by requesting initial model parameters from a randomly selected client. Once the initial parameters are obtained, the server evaluates them before the first round of training. After completing the training phase, the server evaluates the global model by selecting both clients for evaluation and aggregating their results. This process repeats for all five rounds. The server calculates a weighted average of accuracy, which is the accuracy of the global model, from multiple client models [84]. The federated learning algorithm employed is federated averaging, which aggregates client models based on the average of their parameters. As shown in Table 4.2, the global model improves as the rounds progress, demonstrated by a reduction in loss values over time and a corresponding increase in accuracy. After five rounds are completed, a summary of the results is presented in the server output. This includes a history of the loss values and accuracy metrics throughout the five rounds. The steady improvement in both metrics demonstrates the effectiveness of the federated learning approach in refining the global model with the contributions from the distributed clients.

Table 4.2 The accuracy and loss of the FL global model.

FL Round	Accuracy	Loss
1	0.718	1.062
2	0.930	0.206
3	0.908	0.206
4	0.963	0.098
5	0.978	0.058

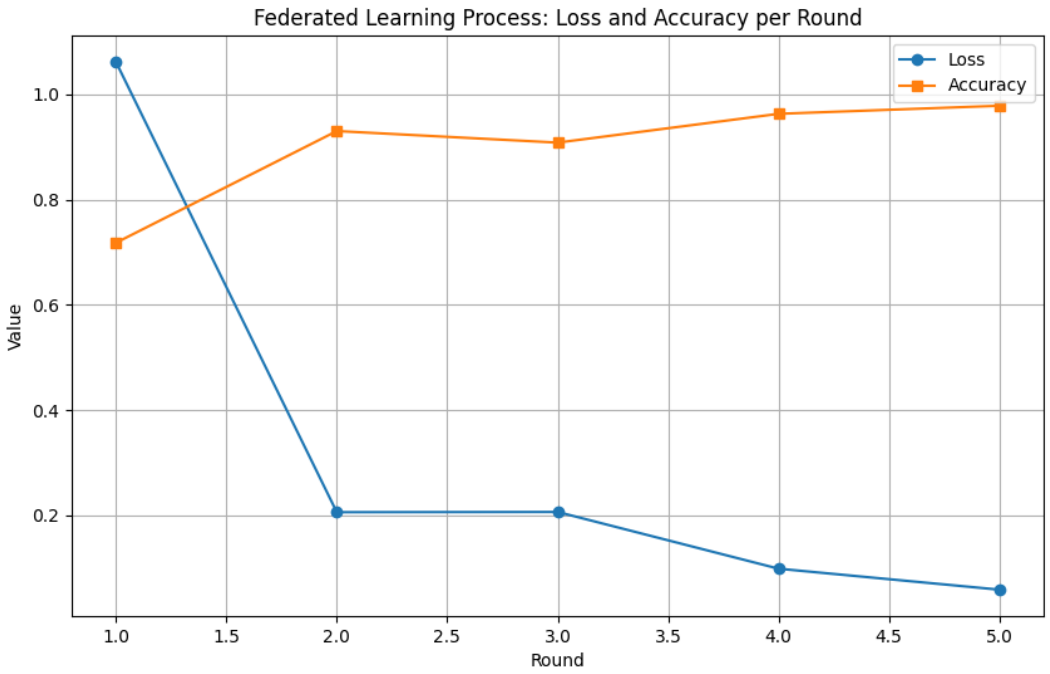

Figure 4.5 The accuracy and loss of the FL global model.

4.5 Multi-Objective Optimization

As discussed in Chapter 3, to apply multi-objective Pareto optimality, we needed to create an augmented dataset. This dataset was created by running the proposed federated learning setup and collecting the results in a file.

4.5.1 Creating the Augmented Dataset

To build a dataset that reflects a balanced trade-off between accuracy, time, and data volume, several experimental scenarios were implemented. These scenarios involve varying key hyperparameters and RNN model architectures to analyze their impact on performance metrics. The goal is to identify the optimal configuration that achieves the best balance between accuracy, time, and data volume during training. The following describes the scenarios intended to produce the enhanced dataset, which incorporates the three crucial factors of accuracy, processing time, and data volume:

- **Modifying the values of hyperparameters:** To correctly train the model to be able to accurately classify body activities, numerous hyperparameters need to be adjusted; these hyperparameters will affect the performance of the network along its time to convergence.
    a) **Change in the number of epochs:** In the context of machine learning, an epoch is a single cycle of model training. After each epoch, the internal parameters of the model are adjusted. The number of full runs necessary to efficiently build an algorithm is expressed in terms of epochs. By increasing the number of epochs, each client goes through more training cycles, which often improves model accuracy. However, this comes at the cost of increased training time. Recent studies have shown that the trade-off between accuracy and training duration becomes significant, necessitating careful tuning to avoid high computational costs [85].
    b) **Change in the batch size:** One of the main hyperparameters that need to be tuned is the batch size, which defines the number of training samples processed before the model updates its weights once. It is a critical hyperparameter that significantly impacts a neural network's performance, efficiency, and generalization ability. Large batch sizes frequently lead to degraded generalization if it is not fine-tuned. Setting this hyperparameter too high can make the network take too long to achieve convergence. However, if the batch size is too low, it will make the network bounce back and forth without reaching a satisfactory level of performance. Additionally, the nature of the dataset can have an impact on the batch size, especially the medical dataset, because of its complexity [86].
- **Modifying the model architecture:** Recurrent neural networks are a popular choice for sequential data, but their performance can vary depending on the depth of the architecture and the number of hidden layers. More complex models tend to capture intricate patterns more effectively but often come with larger parameter sizes and increased computation time. This can result in greater resource demands, reducing efficiency in terms of processing time and data transfer. In addition to the BiLSTM model that we developed, we also created another RNN model, specifically an LSTM model. The purpose of creating the second model was to reduce complexity, as simpler models typically result in a smaller

volume of data being transferred between the client and server. This was done to introduce variation in the dataset in terms of data volume transferred, helping to optimize the amount of data through multi-objective optimization. In designing the second model, we aimed to maximize accuracy while working within the constraints of its simpler architecture, striking a balance between efficiency and performance at this level of complexity. This scenario focuses on experimenting with the two proposed model architectures.

Table 4.3 illustrates the dataset derived from various experiments of the proposed FL framework. It provides a detailed breakdown of configurations tested within the proposed FL framework to evaluate their impact on performance metrics such as data transfer, processing time, and model accuracy. All the information in the dataset pertains to the final communication round, which is iteration five, where the accuracy and time of training the global model are determined. Model 1, which is the BiLSTM model, offers the best accuracy but at a higher computational cost and data transfer volume. Model 2, which is the LSTM model, provides a more efficient option in terms of time and data transfer, albeit with a slight compromise in accuracy. These insights are valuable for selecting configurations that best meet specific requirements for accuracy, processing time, and data efficiency in real-world applications.

Table 4.3 Dataset derived from the proposed FL framework.

Iteration	Number of epochs	Batch size	Model architecture	Data volume	Time	Global model accuracy
5	10	32	Model 1	2840388	113.65	0.97867
5	15	16	Model 1	2840388	212.60	0.97562
5	10	16	Model 1	2840388	202.27	0.97516
5	10	64	Model 1	2840388	70.01	0.97313
5	5	64	Model 1	2840388	47.96	0.97126
5	15	64	Model 1	2840388	78.81	0.96566

5	5	16	Model 1	2840388	128.26	0.96340
5	5	32	Model 1	2840388	74.69	0.96278
5	15	32	Model 1	2840388	122.41	0.96238
5	15	16	Model 2	219218	37.16	0.96798
5	10	16	Model 2	219218	36.74	0.96753
5	15	32	Model 2	219218	22.17	0.96730
5	10	64	Model 2	219218	15.97	0.96640
5	10	32	Model 2	219218	21.43	0.96385
5	15	64	Model 2	219218	14.34	0.96266
5	5	64	Model 2	219218	11.52	0.96057
5	5	32	Model 2	219218	16.57	0.95819
5	5	16	Model 2	219218	26.98	0.93817

4.5.2 Implementation of Pareto Optimization

In this section, we employed a dataset analysis and Pareto optimization approach to identify optimal data points that balance three key metrics: accuracy, total training time, and data volume. Rows containing missing or invalid values in these critical columns were removed to ensure a clean dataset. The next step is the identification of Pareto optimal points. In our case, the goal was to find points that maximize accuracy while minimizing total time and amount of data transferred in FL. The relevant columns, which are accuracy, total time, and data volume were extracted and converted into an array to facilitate computational operations. We implemented a custom function to identify the Pareto front. This function iterates through each data point, comparing it to others based on three criteria: higher accuracy is preferred, lower time is favored, and lower amount of data is prioritized. For each point, if another data point has greater accuracy and equal or better performance on the other two metrics, the point is marked as dominated and excluded from the

Pareto front. The approach returns a set of non-dominated or Pareto optimal points. The final set of Pareto optimal points represents the solutions that offer the best trade-offs between accuracy, processing time, and data amount. This approach enables the identification of the most efficient configurations of the system in terms of accuracy and resource use, which is critical for balancing performance and efficiency in applications. Table 4.4 illustrates the set of Pareto optimal solutions in blue color. Pareto optimization provides a framework for identifying these solutions, leaving the task of selecting a final compromise to the decision-maker.

Table 4.4 Pareto optimal solutions in the dataset.

Iteration	Number of epochs	Batch size	Model architecture	Amount of data	Time	Global model Accuracy
5	10	32	Model_1	2840388	113.65	0.97867
5	15	16	Model_1	2840388	212.60	0.97562
5	10	16	Model_1	2840388	202.27	0.97516
5	10	64	Model_1	2840388	70.01	0.97313
5	5	64	Model_1	2840388	47.96	0.97126
5	15	64	Model_1	2840388	78.81	0.96566
5	5	16	Model_1	2840388	128.26	0.96340
5	5	32	Model_1	2840388	74.69	0.96278
5	15	32	Model_1	2840388	122.41	0.96238
5	15	16	Model_2	219218	37.16	0.96798
5	10	16	Model_2	219218	36.74	0.96753
5	15	32	Model_2	219218	22.17	0.96730
5	10	64	Model_2	219218	15.97	0.96640
5	10	32	Model_2	219218	21.43	0.96385

5	15	64	Model_2	219218	14.34	0.96266
5	5	64	Model_2	219218	11.52	0.96057
5	5	32	Model_2	219218	16.57	0.95819
5	5	16	Model_2	219218	26.98	0.93817

In Figure 4.6, we observe a three-dimensional plot showcasing the trade-offs between accuracy, time, and data amount in our FL model configurations. Each red triangle represents a model configuration, plotted based on its respective values for accuracy, time, and data volume.

The blue dots highlight the Pareto optimal solutions, those configurations that achieve an optimal balance by maximizing accuracy while minimizing time and data requirements. These solutions form the Pareto front, a set where improving one objective would necessarily compromise another objective. Thus, these points represent the most efficient trade-offs, leaving the final selection to the decision-maker based on project goals and constraints.

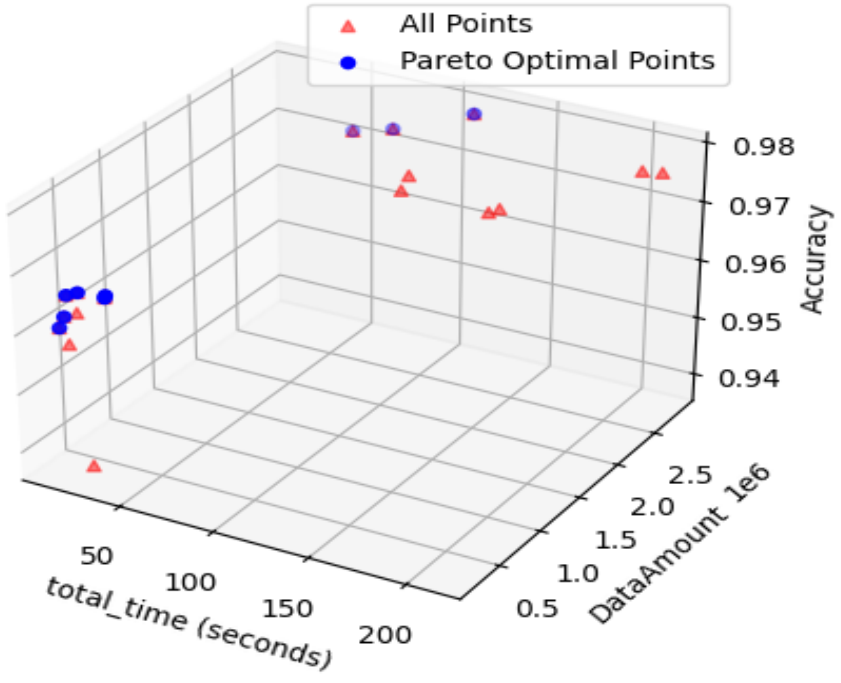

Figure 4.6 Pareto front of optimal solutions.

4.6 Summary

This chapter presented the implementation of our proposed FL model. First, we established our testbed in one machine. Then, we discussed the implementation of the preprocessing methods on the MHEALTH dataset. Data was divided into training dataset, validation dataset and evaluation to robust model training. We implemented the bidirectional LSTM model to train data in FL clients. In our experiments, we ran the FL framework in different scenarios and collected data on time, accuracy, and amount of data. Finally, we applied the Pareto approach to find a set of optimal solutions. We discussed our implementation to provide FL in real-time monitoring, which is done by federated learning and Python libraries. Table 4.5 provides an overview of the proposed FL model, the technologies, and the enhancements depending on each component in our model based on both chapters 3 and 4.

Table 4.5 Summary of the proposed FL model.

Components	Module	Technologies/ Dataset	Task	Achievements
Data Collection and Processing	Data Preprocessing	MHEALTH dataset	Preprocess healthcare data through cleaning, resampling, splitting, scaling, and one-hot encoding	Enhanced feature selection and classification in the activity recognition task.
RNN model	RNN for time-series data	BiLSTM, LSTM	Recognizing physical body activities from time series data using BiLSTM model	Improve accuracy of human activity recognition with BiLSTM model
FL techniques	FL libraries for distributed training	Flower framework, FedAvg algorithm	Manage decentralized training and coordination of the federated learning process.	Established a robust and accurate federated learning model facilitating collaborative learning

Multi objective optimization	Pareto optimization	Pareto optimality approach and Pareto frontier	Identify optimal solutions and visualize it in Pareto frontier plot	Achieved optimal configurations

CHAPTER 5 CONCLUSION

This dissertation presented a framework to enhance the federated learning process in remote patient monitoring systems. The dissertation comes to an end with this chapter. Our work is summarized in the first section. The limits of our work are then described in depth. The future efforts are finally discussed in the last section.

5.1 Summary of Works

The importance of healthcare in our lives is clear, and this field is advancing significantly every day. In this research, we developed an FL model to address the growing need for the efficiency of distributed learning in health monitoring systems. The BiLSTM model was built for accurate human activity recognition and the Pareto optimality was employed to find the set of optimal solutions.

By conducting the literature review on FL in the remote patient monitoring systems and human activity recognition, we identify several weaknesses and shortcomings in existing systems, including a lack of high accuracy in recognizing human body activities in FL, insufficient data privacy, neglect of training processing time, and oversight of communication burden. Therefore, several questions come to mind, What optimization can be done to the federated learning process in health monitoring to ensure efficient and accurate training? What AI methods are capable of classifying patients' physical activities? What approach can be adopted to minimize resource utilization and address communication costs during the model update while maximizing model accuracy?

To address the weaknesses mentioned above, this dissertation proposes a multi-objective approach to optimize federated learning in healthcare monitoring. To do so, we design an optimized federated learning process to enhance performance in terms of accuracy, time, and the amount of exchanged data. The aim is to address the communication costs incurred by the FL process for the training of AI models for the task of HAR and to evaluate the effectiveness of AI methods for the classification of physical activities.

To accomplish our objectives, we developed BiLSTM, a model able to accurately recognize physical body activities, and we used Pareto optimization to identify optimal configurations that

balance time, accuracy, and data transfer efficiency. The findings of the suggested FL model for health monitoring systems are encouraging. In the paragraphs that follow, we will go over each system component, its results, and how it helps us reach our goals.

Our proposed FL model for health monitoring consists of several key components. In the first component of this architecture, data collection and processing, we focus on preparing raw healthcare data for effective use in a machine learning environment. For this, we utilize the MHEALTH dataset, a comprehensive collection of health-related data, alongside various Python libraries tailored for data manipulation and analysis. This preprocessing phase is critical, as it ensures that the data is clean and consistent for the needs of our model. The preprocessing process involves several key steps: cleaning to remove any irrelevant data, resampling to balance the dataset, splitting to allocate portions for training and testing, scaling to bring all data to a common scale, and one-hot encoding to transform categorical labels into a format suitable for machine learning. These steps not only streamline the data but also enhance its quality, making it more conducive for accurate feature selection and classification in our activity recognition task. We create a solid basis for the model to produce accurate, dependable results in later phases.

The second component of our model is dedicated to recognizing physical activities from time-series data through recurrent neural network models. The proposed BiLSTM model is a highly precise method for recognizing human body activities, and capturing temporal patterns in sequential data. The model architecture comprises two bidirectional LSTM layers, with the first layer maintaining the temporal structure and the second layer returning a single output. Batch normalization is used to stabilize the training process. A dense layer with 256 neurons and a ReLU activation function introduces non-linearity, reducing overfitting risks. The output layer is dense with softmax activation, ensuring a probability distribution corresponding to different categories. The model is compiled using the Adam optimizer and categorical cross-entropy as the loss function. This BiLSTM architecture is ideal for sequential data, efficiently learning temporal dependencies and achieving high accuracy. The results show an accuracy of 99.2% for the training set, 99.72% for the validation set, and 99.73% for the test set, highlighting the strong performance of our BiLSTM model. We also experimented with a second RNN model that used the LSTM architecture for temporal relationships in data. It included an LSTM layer, followed by batch normalization, dropout, dense, and output layers. The model used the Adam optimizer and

categorical cross-entropy loss function for multi-class classification tasks. Although this model has lower accuracy than the BiLSTM model, it requires less training time and has a smaller data volume for model parameters.

The third component of our model is focused on applying federated learning techniques to enable decentralized training across multiple data sources. In this setup, we use specialized libraries, including the Flower framework, implemented entirely in Python. FedAvg strategy is also used for federated aggregation and model training. With federated learning, each client trains the model on its local data, and only the learned parameters are shared with the server. The server then aggregates these parameters to update the global model, creating a collaborative learning process that preserves data privacy. This decentralized training approach leverages a diverse range of health data, making the model more robust and generalizable. Through effective management of the federated learning process, we achieved a resilient and highly accurate global model, reaching an accuracy of 97.8% after five communication rounds.

The fourth component of our model is focused on Multi-Objective Optimization to balance key performance metrics such as accuracy, training time, and data volume. For this, we employ a Pareto optimization approach, which identifies configurations that achieve the best trade-offs between these objectives, ensuring our model operates efficiently within practical constraints. To implement this, we create an augmented dataset by running various experimental scenarios within our federated learning framework. Each scenario varies critical parameters of the number of epochs, batch size, and model architecture. By doing so, we collect valuable data on how these factors impact accuracy, processing time, and data volume. The resulting dataset is analyzed to identify Pareto optimal solutions that are configurations where improving one metric would necessitate a compromise in others. A Pareto frontier plot is then used to display these ideal configurations. This approach not only gives decision-makers a clear framework for determining the optimal settings of our model, but it also gives them the freedom to select configurations that best suit particular objectives or resource constraints. Finally, multi-objective optimization enables us to fine-tune the model for real-world applications, where achieving high performance must be balanced with practical considerations.

5.2 Limitations

Our proposed model has several limitations that have identified areas for potential enhancement. One of the limitations is scalability, particularly since the federated learning model was implemented with a limited number of clients. This simplifies coordination and communication but does not fully capture the complexities that would arise in a larger-scale deployment. Moreover, an expanded client network requires more advanced techniques to manage communication costs effectively. To ensure scalability for real-world applications, especially in healthcare settings with diverse and distributed data sources, future work would need to address these factors. Additionally, while the augmented MHEALTH dataset provided substantial training data, its specificity to human activity recognition may limit the model's generalizability to broader healthcare monitoring tasks. The performance and generalization of the model could further benefit from a more diverse dataset, encompassing a broader range of activities or health-related parameters. A personal computer was used to test the models in a controlled environment. However, the testing environment was not representative of real-world conditions, as it included factors that could lower the model's performance, such as, concurrent user queries, and hardware limitations in lower-resource environments. To sum up, our project demonstrated excellent accuracy, efficiency, and usability, however, limitations in the dataset, scalability, and real-world testing conditions highlight areas that require further research to improve the resilience and applicability of the FL process in health monitoring systems.

5.3 Future Work

For future work, several enhancements could further strengthen the findings of this project. The first improvement in our proposed model would be expanding the number of clients, to better reflect the variability and distributed nature of real-world healthcare environments. Additionally, broadening the dataset beyond human activity recognition to include a wider range of health monitoring tasks could enhance the model's generalizability. This expansion could involve diverse health metrics, creating a more comprehensive and robust FL application in healthcare monitoring. Future work could also explore testing the model in more resource-constrained environments. Deploying the FL system on devices with varied computational capacities would provide insights into its performance under realistic conditions and identify potential optimizations for these

scenarios. Moreover, incorporating real-time data processing capabilities into the federated learning model represents an important area for future work. Real-time processing would enable the model to analyze and respond to incoming data continuously, providing timely insights essential for dynamic healthcare monitoring applications. To achieve real-time processing, the FL framework would need to be adapted to handle streaming data while maintaining synchronization across multiple clients [88]. This could involve implementing techniques such as incremental learning, where the model updates continuously with new data rather than relying on periodic training cycles [87]. Real-time data processing would enhance the responsiveness of the model and make it more practical for real-world applications, enabling healthcare providers to monitor patient conditions continuously and intervene promptly when needed. Moreover, instead of Pareto, reinforcement learning can be used to learn to react to the changes in the environment in real time. Through these steps, we can better evaluate and refine the resilience, scalability, and suitability of the FL framework for practical healthcare applications.

In sum, future developments should focus on deep learning methods, real-time data processing, scalability of the model, and dataset expansion. The effectiveness of the framework would be substantially increased by these advancements, making it a more reliable and adaptable instrument in the healthcare industry.

REFERENCES

[1]R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated Learning for Healthcare: Systematic Review and Architecture Proposal," ACM Transactions on Intelligent Systems and Technology, vol. 13, no. 4, pp. 1–23, Aug. 2022, doi: https://doi.org/10.1145/3501813.

[2] J. Shao et al., "A Survey of What to Share in Federated Learning: Perspectives on Model Utility, Privacy Leakage, and Communication Efficiency," arXiv.org, 2023. https://arxiv.org/abs/2307.10655 (accessed Sep. 25, 2024).

[3] T. Shaik et al., "FedStack: Personalized activity monitoring using stacked federated learning," Knowledge-Based Systems, vol. 257, p. 109929, Dec. 2022, doi: https://doi.org/10.1016/j.knosys.2022.109929.

[4] T. Shaik et al., "Remote patient monitoring using artificial intelligence: Current state, applications, and challenges," WIREs Data Mining and Knowledge Discovery, vol. 13, no. 2, Jan. 2023, doi: https://doi.org/10.1002/widm.1485.

[5] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," Computers & Industrial Engineering, vol. 149, p. 106854, Nov. 2020, doi: https://doi.org/10.1016/j.cie.2020.106854.

[6] S. Sanchez, J. Machacuay, and M. Quinde, "Federated Learning for Human Activity Recognition on the MHealth Dataset," Lecture Notes in Computer Science, pp. 215–225, Jan. 2023, doi: https://doi.org/10.1007/978-3-031-42505-9_19.

[7] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," Knowledge-Based Systems, vol. 216, p. 106775, Mar. 2021, doi: https://doi.org/10.1016/j.knosys.2021.106775.

[8] K. M. J. Rahman et al., "Challenges, Applications and Design Aspects of Federated Learning: A Survey," IEEE Access, vol. 9, pp. 124682–124700, 2021, doi: https://doi.org/10.1109/access.2021.3111118.

[9] A. Rahman et al., “Federated learning-based AI approaches in smart healthcare: concepts, taxonomies, challenges and open issues,” Cluster Computing, Aug. 2022, doi: https://doi.org/10.1007/s10586-022-03658-4.

[10] D. C. Nguyen et al., “Federated Learning for Smart Healthcare: A Survey,” ACM Computing Surveys, vol. 55, no. 3, pp. 1–37, Apr. 2023, doi: https://doi.org/10.1145/3501296.

[11] M. Ali, F. Naeem, M. Tariq, and G. Kaddoum, “Federated Learning for Privacy Preservation in Smart Healthcare Systems: A Comprehensive Survey,” IEEE Journal of Biomedical and Health Informatics, pp. 1–14, 2022, doi: https://doi.org/10.1109/jbhi.2022.3181823.

[12] S. Rani, A. Kataria, S. Kumar, and P. Tiwari, “Federated learning for secure IoMT-applications in smart healthcare systems: A comprehensive review,” Knowledge-Based Systems, p. 110658, May 2023, doi: https://doi.org/10.1016/j.knosys.2023.110658.

[13] H. Mcmahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera, “Communication-Efficient Learning of Deep Networks from Decentralized Data.” Accessed: May 13, 2022. [Online]. Available: https://arxiv.org/pdf/1602.05629

[14] Samuel Bonet Olivencia, Karim Zahed, Farzan Sasangohar, Rotem Davir, and A. Vedlitz, “Integration of Remote Patient Monitoring Systems into Physicians Work in Underserved Communities: Survey of Healthcare Provider Perspectives,” arXiv (Cornell University), Jun. 2022, doi: https://doi.org/10.48550/arxiv.2207.01489.

[15] M. Javaid, A. Haleem, and R. P. Singh, “Health informatics to enhance the healthcare industry’s culture: An extensive analysis of its features, contributions, applications and limitations,” Informatics and Health, vol. 1, no. 2, Jun. 2024, doi: https://doi.org/10.1016/j.infoh.2024.05.001.

[16] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, “Internet of multimedia things: Vision and challenges,” Ad Hoc Networks, vol. 33, pp. 87–111, Oct. 2015, doi: https://doi.org/10.1016/j.adhoc.2015.04.006.

[17] W. Chen, G. Ma, T. Fan, Y. Kang, Q. Xu, and Q. Yang, “SecureBoost+ : A High Performance Gradient Boosting Tree Framework for Large Scale Vertical Federated Learning,” arXiv (Cornell University), Jan. 2021, doi: https://doi.org/10.48550/arxiv.2110.10927.

[18] R. C. Walker, A. Tong, K. Howard, and S. C. Palmer, “Patient expectations and experiences of remote monitoring for chronic diseases: systematic review and thematic synthesis of qualitative studies,” International journal of medical informatics, vol. 124, pp. 78–85, 2019.

[19] S. Ghosh and S. K. Ghosh, “FEEL: FEderated LEarning Framework for ELderly Healthcare Using Edge-IoMT,” IEEE Transactions on Computational Social Systems, vol. 10, no. 4, pp. 1800–1809, Aug. 2023, doi: https://doi.org/10.1109/tcss.2022.3233300

[20] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, “FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare,” IEEE Intelligent Systems, vol. 35, no. 4, pp. 83–93, Jul. 2020, doi: https://doi.org/10.1109/mis.2020.2988604

[21] Y. S. Can and C. Ersoy, “Privacy-preserving Federated Deep Learning for Wearable IoT-based Biomedical Monitoring,” ACM Transactions on Internet Technology, vol. 21, no. 1, pp. 1–17, Jan. 2021, doi: https://doi.org/10.1145/3428152.

[22] T. Shaik et al., “FedStack: Personalized activity monitoring using stacked federated learning,” Knowledge-Based Systems, vol. 257, p. 109929, Dec. 2022, doi: https://doi.org/10.1016/j.knosys.2022.109929.

[23] D. López, B. Farooq, and Ranwa Al Mallah, “Adapting Detection in Blockchain-enabled Federated Learning for IoT Networks,” Jul. 2023, doi: https://doi.org/10.1109/iceccme57830.2023.10252819.

[24] X. Zhou, J. Zhao, H. Han, and C. Guet, “Joint Optimization of Energy Consumption and Completion Time in Federated Learning,” 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Jul. 2022, doi: https://doi.org/10.1109/icdcs54860.2022.00101.

[25] Muhammad Arsalan, Davide Di Matteo, S. Imtiaz, Z. Abbas, Vladimir Vlassov, and Vadim Issakov, “Energy-Efficient Privacy-Preserving Time-Series Forecasting on User Health Data Streams,” Dec. 2022, doi: https://doi.org/10.1109/trustcom56396.2022.00080.

[26] J. Bian, C. Shen, and J. Xu, “Federated Learning via Indirect Server-Client Communications,” arXiv (Cornell University), Feb. 2023, doi: https://doi.org/10.48550/arxiv.2302.07323.
[27]Q. Wu, X. Chen, Z. Zhou, and J. Zhang, “FedHome: Cloud-Edge based Personalized Federated

Learning for In-Home Health Monitoring," IEEE Transactions on Mobile Computing, pp. 1–1, 2020, doi: https://doi.org/10.1109/tmc.2020.3045266.

[28]S. Sanchez, J. Machacuay, and M. Quinde, "Federated Learning for Human Activity Recognition on the MHealth Dataset," Lecture Notes in Computer Science, pp. 215–225, Jan. 2023, doi: https://doi.org/10.1007/978-3-031-42505-9_19.

[29] M. Baltabay, A. Yazici, M. Sterling, and E. Ever, "Designing Efficient and Lightweight Deep Learning Models for Healthcare Analysis," Neural Processing Letters, Mar. 2023, doi: https://doi.org/10.1007/s11063-023-11246-9.

[30] P. Agarwal and M. Alam, "A Lightweight Neuromorphic CNN for Human Activity Recognition on Edge Device," May 2023, doi: https://doi.org/10.1109/iciccs56967.2023.10142696.

[31] A. A. Al-Saedi, Emiliano Casalicchio, and Veselka Boeva, "An Energy-Aware Multi-Criteria Federated Learning Model for Edge Computing," DiVA (Blekinge Institute of Technology), Aug. 2021, doi: https://doi.org/10.1109/ficloud49777.2021.00027.

[32] A. A. Abdellatif et al., "Communication-efficient hierarchical federated learning for IoT heterogeneous systems with imbalanced data," Future Generation Computer Systems, vol. 128, pp. 406–419, Mar. 2022, doi: https://doi.org/10.1016/j.future.2021.10.016.

[33] L. Tu, X. Ouyang, J. Zhou, Y. He, and G. Xing, "FedDL," Nov. 2021, doi: https://doi.org/10.1145/3485730.3485946.

[34] L. U. Khan, M. Alsenwi, I. Yaqoob, M. Imran, Z. Han, and C. S. Hong, "Resource Optimized Federated Learning-Enabled Cognitive Internet of Things for Smart Industries," IEEE Access, vol. 8, pp. 168854–168864, 2020, doi: https://doi.org/10.1109/ACCESS.2020.3023940.

[35] K. S. Arikumar et al., "FL-PMI: Federated Learning-Based Person Movement Identification through Wearable Devices in Smart Healthcare Systems," Sensors, vol. 22, no. 4, p. 1377, Feb. 2022, doi: https://doi.org/10.3390/s22041377.

[36]S. Gupta, "Deep learning based human activity recognition (HAR) using wearable sensor data," International Journal of Information Management Data Insights, vol. 1, no. 2, p. 100046, Nov. 2021, doi: https://doi.org/10.1016/j.jjimei.2021.100046.

[37] Ahmed Younes Shdefat, Mostafa, N., Zakwan Al-Arnaout, Yehia Kotb, & Alabed, S. (2024). Optimizing HAR Systems: Comparative Analysis of Enhanced SVM and k-NN Classifiers. the International Journal of Computational Intelligence Systems/International Journal of Computational Intelligence Systems, 17(1). https://doi.org/10.1007/s44196-024-00554-0

[38] Dasaradharami Reddy, K., & Gadekallu, T. R. (2023). A Comprehensive Survey on Federated Learning Techniques for Healthcare Informatics. Computational Intelligence and Neuroscience, 2023, 1–19. https://doi.org/10.1155/2023/8393990

[39] Yoo, J. H., Jeong, H., Lee, J., & Chung, T.-M. (2022). Open problems in medical federated learning. International Journal of Web Information Systems, 18(2/3), 77–99. https://doi.org/10.1108/ijwis-04-2022-0080

[40] FedMAP: Unlocking Potential in Personalized Federated Learning through Bi-Level MAP Optimization. (n.d.). Arxiv.org. Retrieved July 18, 2024, from https://arxiv.org/html/2405.19000v1#bib.bib2

[41]Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., & Chan, K. (2019). Adaptive Federated Learning in Resource Constrained Edge Computing Systems. IEEE Journal on Selected Areas in Communications, 37(6), 1205–1221. https://doi.org/10.1109/JSAC.2019.2904348

[42] Xie, C., Koyejo, S., & Gupta, I. (2020). Asynchronous Federated Optimization. ArXiv:1903.03934 [Cs]. https://arxiv.org/abs/1903.03934

[43] Konečný, J., McMahan, H. B., Ramage, D., & Richtárik, P. (2016). *Federated optimization: Distributed machine learning for on-device intelligence*. [Technical report]. University of Edinburgh and Google. https://arxiv.org/abs/1610.02527

[44] H. Brendan McMahan, Moore, E., Ramage, D., Hampson, S., & Blaise. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. International Conference on Artificial Intelligence and Statistics, 1273–1282.

[45] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., & Seth, K. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. https://doi.org/10.1145/3133956.3133982

[46] Chellapandi, Vishnu Pandi, L. Yuan, C. G. Brinton, S. H. Zak, and Z. Wang, “Federated Learning for Connected and Automated Vehicles: A Survey of Existing Approaches and Challenges,” arXiv.org, 2023. https://arxiv.org/abs/2308.10407 (accessed Sep. 27, 2024).

[47] José Ángel Morell, Zakaria Abdelmoiz Dahi, Chicano, F., Luque, G., & Alba, E. (2022). Optimising Communication Overhead in Federated Learning Using NSGA-II. Lecture Notes in Computer Science, 317–333. https://doi.org/10.1007/978-3-031-02462-7_21

[48] فلاح, مهدی, & صالحپور, پدرام. (2023). Optimizing Data Transfer and Convergence Time for Federated Learning based on NSGA II. *مجله مهندسی برق دانشگاه تبریز*, *53*(1), 61–67. https://doi.org/10.22034/tjee.2023.16064

[49] Zheng-yi Chai, Chuan-dong Yang & Ya-lun Li, Ed., “Communication efficiency optimization in federated learning based on multi-objective evolutionary algorithm,” https://link.springer.com/article/10.1007/s12065-022-00718-x, Apr. 12, 2022.

[50] Dheeraj Dayakaran, & Nalinadevi Kadiresan. (2024). Federated Learning Framework for Human Activity Recognition Using Smartphones. Procedia Computer Science, 235, 2069–2078. https://doi.org/10.1016/j.procs.2024.04.196

[51] S. Ji et al., “Emerging trends in federated learning: from model fusion to federated X learning,” International journal of machine learning and cybernetics, Apr. 2024, doi: https://doi.org/10.1007/s13042-024-02119-1.

[52] Z. Chai, C. Yang, and Y. Li, “Communication efficiency optimization in federated learning based on multi-objective evolutionary algorithm,” Evolutionary Intelligence, vol. 16, no. 3, pp. 1033–1044, Apr. 2022, doi: https://doi.org/10.1007/s12065-022-00718-x.

[53] B. Gupta, P. Mittal, and T. Mufti, “A Review on Amazon Web Service (AWS), Microsoft Azure & Google Cloud Platform (GCP) Services,” Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India, vol. 1, no. 2, 2021, Available: https://eudl.eu/pdf/10.4108/eai.27-2-2020.2303255

[54] X. Gu, F. Sabrina, Z. Fan, and S. Sohail, “A Review of Privacy Enhancement Methods for Federated Learning in Healthcare Systems,” International Journal of Environmental Research and Public Health, vol. 20, no. 15, p. 6539, Jan. 2023, doi: https://doi.org/10.3390/ijerph20156539.

[55] “UCI Machine Learning Repository,” archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/319/mhealth+dataset

[56] “Data Mining: Practical Machine Learning Tools and Techniques | ScienceDirect,” Sciencedirect.com, 2011. https://www.sciencedirect.com/book/9780123748560/data-mining-practical-machine-learning-tools-and-techniques

[57] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, “Self-Balancing Federated Learning With Global Imbalanced Data in Mobile Systems,” vol. 32, no. 1, pp. 59–71, Jan. 2021, doi: https://doi.org/10.1109/tpds.2020.3009406.

[58] W. Jiang, “Applications of deep learning in stock market prediction: Recent progress,” Expert Systems with Applications, vol. 184, p. 115537, Dec. 2021, doi: https://doi.org/10.1016/j.eswa.2021.115537.

[59] M. K. Dahouda and I. Joe, “A Deep-Learned Embedding Technique for Categorical Features Encoding,” IEEE Access, vol. 9, pp. 114381–114391, 2021, doi: https://doi.org/10.1109/ACCESS.2021.3104357.

[60] Md. Z. Uddin, M. M. Hassan, A. Alsanad, and C. Savaglio, “A body sensor data fusion and deep recurrent neural network-based behavior recognition approach for robust healthcare,” Information Fusion, vol. 55, pp. 105–115, Mar. 2020, doi: https://doi.org/10.1016/j.inffus.2019.08.004.

[61] Y. Imrana, Y. Xiang, L. Ali, and Z. Abdul-Rauf, “A bidirectional LSTM deep learning approach for intrusion detection,” Expert Systems with Applications, vol. 185, p. 115524, Dec. 2021, doi: https://doi.org/10.1016/j.eswa.2021.115524.

[62] J. P. Tan, A. Liza, M. V. Abante, R. L. Tadeo, and R. R. Lansigan, “A Performance Review of Recurrent Neural Networks Long Short-Term Memory (LSTM),” 2022 3rd International Conference for Emerging Technology (INCET), May 2022, doi: https://doi.org/10.1109/incet54531.2022.9824567.

[63] F. García, F. Guijarro, J. Oliver, and R. Tamošiūnienė, “Foreign Exchange Forecasting Models: LSTM and BiLSTM Comparison,” Jul. 2024, doi: https://doi.org/10.3390/engproc2024068019.

[64] Ameer Ali Ridha, Ihab Almaameri, László Blázovics, and Haider Mohammed Abbas, “Human Activity Recognition by BiLSTM Recurrent Neural Networks and Support Vector Machine,” Jul. 2023, doi: https://doi.org/10.1109/iiceta57613.2023.10351372.

[65] M. Filho, “Multiple Time Series Forecasting With LSTM In Python,” Forecastegy.com, Feb. 09, 2023. https://forecastegy.com/posts/multiple-time-series-forecasting-with-lstm-in-python/ (accessed Oct. 24, 2024).

[66] H. Wang, J. Wang, L. Cao, Y. Li, Q. Sun, and J. Wang, “A Stock Closing Price Prediction Model Based on CNN-BiSLSTM,” Complexity, vol. 2021, pp. 1–12, Sep. 2021, doi: https://doi.org/10.1155/2021/5360828.

[67] Z. Hu, “A web application for crowd counting by building parallel and direct connection-based CNN architectures,” pp. 47–82, Jan. 2022, doi: https://doi.org/10.1016/b978-0-12-824410-4.00012-x.

[68] “Dense layer - AI Wiki - Artificial Intelligence, Machine Learning Wiki and Guide,” aiwiki.ai. https://aiwiki.ai/wiki/Dense_layer

[69] J. Lampinen, “Multiobjective Nonlinear Pareto-Optimization,” ResearchGate, 2014. https://www.researchgate.net/publication/239560754_Multiobjective_Nonlinear_Pareto-Optimization (accessed Oct. 08, 2024).

[70] A. K. Tan, M. Tegmark, and I. L. Chuang, “Pareto-Optimal Clustering with the Primal Deterministic Information Bottleneck,” Entropy, vol. 24, no. 6, p. 771, May 2022, doi: https://doi.org/10.3390/e24060771.

[71] Q. Zhu, Z. He, T. Zhang, and W. Cui, “Improving Classification Performance of Softmax Loss Function Based on Scalable Batch-Normalization,” Applied Sciences, vol. 10, no. 8, p. 2950, Apr. 2020, doi: https://doi.org/10.3390/app10082950.

[72] C. Garbin, X. Zhu, and O. Marques, “Dropout vs. batch normalization: an empirical study of their impact on deep learning,” Multimedia Tools and Applications, vol. 79, no. 19–20, pp. 12777–12815, Jan. 2020, doi: https://doi.org/10.1007/s11042-019-08453-9.

[73] D. J. Beutel et al., “Flower: A Friendly Federated Learning Research Framework,” arXiv:2007.14390 [cs, stat], Apr. 2021, Available: https://arxiv.org/abs/2007.14390

[74] S. Dasari, “COMPARISON OF FEDERATED LEARNING STRATEGIES USING THE FLOWER FRAMEWORK ON EMBEDDED DEVICES,” 2024. Accessed: Oct. 11, 2024. [Online]. Available: https://trepo.tuni.fi/bitstream/handle/10024/157908/DasariSaiPoojith.pdf?sequence=2

[75] J. Yuan, H.-L. Liu, Y.-S. Ong, and Z. He, “Indicator-Based Evolutionary Algorithm for Solving Constrained Multiobjective Optimization Problems,” IEEE Transactions on Evolutionary Computation, vol. 26, no. 2, pp. 379–391, Jun. 2021, doi: https://doi.org/10.1109/tevc.2021.3089155.

[76] F. Karl et al., “Multi-Objective Hyperparameter Optimization in Machine Learning—An Overview,” ACM transactions on evolutionary learning, vol. 3, no. 4, pp. 1–50, Dec. 2023, doi: https://doi.org/10.1145/3610536.

[77] H. Mcmahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera, “Communication-Efficient Learning of Deep Networks from Decentralized Data.” Available: https://arxiv.org/pdf/1602.05629

[78] J. S.-P. Díaz et al., “Making Federated Learning Accessible to Scientists: The AI4EOSC Approach,” pp. 253–264, Jun. 2024, doi: https://doi.org/10.1145/3658664.3659642.

[79] H. Hilberger, S. Hanke, and M. Bödenler, “Federated Learning with Dynamic Model Exchange,” Electronics, vol. 11, no. 10, p. 1530, May 2022, doi: https://doi.org/10.3390/electronics11101530.

[80] [1] J. Brownlee, “Dropout Regularization in Deep Learning Models With Keras,” Machine Learning Mastery, Jun. 19, 2016. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[81] C. Banerjee, T. Mukherjee, and E. Pasiliao, “The Multi-phase ReLU Activation Function,” Apr. 2020, doi: https://doi.org/10.1145/3374135.3385313.

[82] Y. Gao, W. Liu, and F. Lombardi, “Design and Implementation of an Approximate Softmax Layer for Deep Neural Networks,” Oct. 2020, doi: https://doi.org/10.1109/iscas45731.2020.9180870.

[83] Manjunatha Veerappa, M. Anneken, N. Burkart, and M. F. Huber, “Explaining CNN classifier using association rule mining methods on time-series,” Elsevier eBooks, pp. 173–189, Jan. 2023, doi: https://doi.org/10.1016/b978-0-32-396098-4.00015-6.

[84] Z. Li, T. Lin, X. Shang, and C. Wu, “Revisiting Weighted Aggregation in Federated Learning with Neural Networks,” arXiv (Cornell University), Jan. 2023, doi: https://doi.org/10.48550/arxiv.2302.10911.

[85] Majid Kundroo and T. Kim, “Federated learning with hyper-parameter optimization,” Journal of King Saud University - Computer and Information Sciences, vol. 35, no. 9, pp. 101740–101740, Sep. 2023, doi: https://doi.org/10.1016/j.jksuci.2023.101740.

[86] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” ICT Express, vol. 6, no. 4, May 2020, doi: https://doi.org/10.1016/j.icte.2020.04.010.

[87] B. Li, S. Chen, and Z. Peng, “New Generation Federated Learning,” Sensors, vol. 22, no. 21, p. 8475, Nov. 2022, doi: https://doi.org/10.3390/s22218475.

[88] K. Guo, T. Chen, S. Ren, N. Li, M. Hu, and J. Kang, “Federated Learning Empowered Real-Time Medical Data Processing Method for Smart Healthcare,” IEEE/ACM Transactions on Computational Biology and Bioinformatics, pp. 1–12, 2022, doi: https://doi.org/10.1109/tcbb.2022.3185395.

APPENDIX A PREPROCESSING AND PROPOSED BILSTM MODEL

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Bidirectional
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dropout
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Load data
data = pd.read_csv('mHealth_subject1.csv', sep='\t')
data.columns = data.columns.str.strip()

# Resample activity 0 (null class) to 100 observations per EDA phase
data_activity_0 = data[data['activity'] == 0]
data_activity_else = data[data['activity'] != 0]
data_activity_0 = data_activity_0.sample(n=3, replace=True, random_state=1)
data = pd.concat([data_activity_0, data_activity_else])

# Split data between predictors and output variable
X = data.drop(['activity'], axis=1)
y = data['activity']

# Scale predictors
X_scaled = StandardScaler().fit_transform(X)

# Create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=1)

# Reshape data for LSTM (samples, time steps, features)
X_train_lstm = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Convert output variables to categorical
y_train_categorical = pd.get_dummies(y_train).values
y_test_categorical = pd.get_dummies(y_test).values
```

```python
# Define BiLSTM model
model = Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model.summary()
# Fit model
history = model.fit(X_train_lstm, y_train_categorical, epochs=10, validation_data=(X_test_lstm, y_test_categorical), batch_size=32,
                    callbacks=[early_stopping])

# Assuming 'history' is the object returned by model.fit()
def plot_learning_curve(history):
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()
```

```
# Call this function after training
plot_learning_curve(history)

# Evaluate model
_, accuracy = model.evaluate(X_test_lstm, y_test_categorical)
print("Accuracy: %.2f%%" % (accuracy * 100))

# Make predictions
y_pred_prob = model.predict(X_test_lstm)
y_pred = np.argmax(y_pred_prob, axis=1)

# Show results
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

APPENDIX B CLIENT.PY FOR BILSTM MODEL

```
import pickle
import tensorflow as tf
import flwr as fl
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
import time
import sys
import array
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.feature_selection import SequentialFeatureSelector
import tensorflow as tf
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from keras.models import Sequential
from keras.layers import Conv1D, Dropout, MaxPooling1D, Flatten, Dense, Bidirectional
from keras.layers import LSTM
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization
from keras.optimizers import Adam,SGD
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping



# Load data
data = pd.read_csv('mHealth_subject1.csv', sep='\t')
data.columns = data.columns.str.strip()

# Resample activity 0 (null class) to 3 observations per EDA phase
data_activity_0 = data[data['activity'] == 0]
data_activity_else = data[data['activity'] != 0]
data_activity_0 = data_activity_0.sample(n=3, replace=True, random_state=1)
data = pd.concat([data_activity_0, data_activity_else])
```

```
# Split data between predictors and output variable
X = data.drop(['activity'], axis=1)
y = data['activity']

# Scale predictors
X_scaled = StandardScaler().fit_transform(X)

# Create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=1)

# Reshape data for LSTM (samples, time steps, features)
X_train_lstm = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Convert output variables to categorical
y_train_categorical = pd.get_dummies(y_train).values
y_test_categorical = pd.get_dummies(y_test).values

# Updated LSTM model architecture
model = Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))

optimizer = Adam()
# Compile model
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Define early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

```python
# Define a custom Flower Client class inheriting from Flower's NumPyClient
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, epochs, batch_size):
        super().__init__()  # Initialize the base class
        self.iteration_count = 0  # Track number of training iterations
        self.epochs = epochs  # Set the number of epochs for training
        self.batch_size = batch_size  # Set the batch size for training
        self.model = model  # Define the model for training
        self.optimizer = optimizer  # Define the optimizer for training
        self.results = []  # List to store training results

    # Method to retrieve model parameters
    def get_parameters(self, config):
        return model.get_weights()  # Return the current model weights

    # Method to train the model with provided parameters
    def fit(self, parameters, config):
        self.iteration_count += 1  # Increment iteration count
        iteration_start_time = time.time()  # Record start time for training

        model.set_weights(parameters)  # Set model weights to received parameters
        # Train the model with the specified epochs, batch size, and validation data
        history = model.fit(
            X_train_lstm, y_train_categorical,
            validation_data=(X_test_lstm, y_test_categorical),
            epochs=self.epochs, batch_size=self.batch_size,
            callbacks=[early_stopping]
        )

        # Calculate and log time taken for this iteration
        iteration_end_time = time.time()
        training_time = iteration_end_time - iteration_start_time

        # Measure bandwidth usage (sent and received)
        sent_bandwidth = len(pickle.dumps(parameters))
        received_bandwidth = len(pickle.dumps(model.get_weights()))
        accuracy = history.history['accuracy'][-1]  # Track accuracy of the final epoch
        total_bandwidth = received_bandwidth + sent_bandwidth  # Total bandwidth used

        # Print iteration metrics for time and bandwidth
        print(f"Iteration {self.iteration_count} total time: {training_time} seconds")
        print(f"Iteration {self.iteration_count} total bandwidth: {total_bandwidth} bytes")
```

```
        # Prepare result dictionary for logging
        iteration_result = {
            'iteration': self.iteration_count,
            'clientID': "client_1",
            'dataSize': total_bandwidth,
            'training_time': training_time,
            'epochs': self.epochs,
            'batch_size': self.batch_size,
            'optimizer': type(self.optimizer).__name__,
            'model_architecture': "model_1"
        }

        # Save the result to a CSV file after each iteration
        self.append_results_to_csv(iteration_result)
        return model.get_weights(), len(X_train_lstm), {}  # Return updated weights and training data size

    # Method to evaluate model performance on test data
    def evaluate(self, parameters, config):
        evaluation_start_time = time.time()  # Record evaluation start time
        model.set_weights(parameters)  # Set model weights to received parameters
        loss, accuracy = model.evaluate(X_test_lstm, y_test_categorical)  # Evaluate the model
        evaluation_time = time.time() - evaluation_start_time  # Calculate evaluation time

        # Print evaluation metrics
        print(f"Iteration evaluation time: {evaluation_time}")
        print(f"Iteration evaluation accuracy: {accuracy}")
        return loss, len(X_test_lstm), {"accuracy": accuracy}  # Return loss, data size, and accuracy

    # Helper method to log results to a CSV file
    def append_results_to_csv(self, result):
        df_result = pd.DataFrame([result])  # Convert result to DataFrame
        file_exists = os.path.isfile('client_results.csv')  # Check if CSV file exists
        # Append results to CSV; add header if file doesn't exist
        df_result.to_csv('client_results.csv', mode='a', header=not file_exists, index=False)

# Create an instance of the FlowerClient with specified epochs and batch size
client_1 = FlowerClient(epochs=10, batch_size=32)

# Start the Flower client, connecting to the specified server address
fl.client.start_client(server_address="127.0.0.1:8080", client=client_1)
```

APPENDIX C CLIENT.PY FOR LSTM MODEL

```
import pickle
import tensorflow as tf
import flwr as fl
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
import time
import sys
import datetime
import array
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.feature_selection import SequentialFeatureSelector
import tensorflow as tf
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from keras.models import Sequential
from keras.layers import Conv1D, Dropout, MaxPooling1D, Flatten, Dense, Bidirectional
from keras.layers import LSTM
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization
from keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping

# Load data
data = pd.read_csv('mHealth_subject1.csv', sep='\t')
data.columns = data.columns.str.strip()

# Resample activity 0 (null class) to 3 observations per EDA phase
data_activity_0 = data[data['activity'] == 0]
data_activity_else = data[data['activity'] != 0]
data_activity_0 = data_activity_0.sample(n=3, replace=True, random_state=1)
data = pd.concat([data_activity_0, data_activity_else])

# Split data between predictors and output variable
X = data.drop(['activity'], axis=1)
y = data['activity']
```

```
# Scale predictors
X_scaled = StandardScaler().fit_transform(X)

# Create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=1)

# Reshape data for LSTM (samples, time steps, features)
X_train_lstm = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Convert output variables to categorical
y_train_categorical = pd.get_dummies(y_train).values
y_test_categorical = pd.get_dummies(y_test).values

# Updated LSTM model architecture
model = Sequential()
model.add(LSTM(64, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))

optimizer = Adam()
# Compile model
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


# Define early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

```
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, epochs, batch_size):
        super().__init__()
        self.iteration_count = 0
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = model
        self.optimizer = optimizer
        self.results = []

    def get_parameters(self, config):
        return model.get_weights()

    def fit(self, parameters, config):
        self.iteration_count += 1
        iteration_start_time = time.time()
        model.set_weights(parameters)
        history = model.fit(X_train_lstm, y_train_categorical, validation_data=(X_test_lstm, y_test_categorical),epochs=self.epochs, batch_size=self.batch_size,
                            callbacks=[ early_stopping] )

        iteration_end_time = time.time()
        training_time = iteration_end_time - iteration_start_time
        sent_bandwidth = len(pickle.dumps(parameters))
        received_bandwidth = len(pickle.dumps(model.get_weights()))
        # Track accuracy (or other performance metrics)
        accuracy = history.history['accuracy'][-1]


        total_bandwidth = received_bandwidth + sent_bandwidth


        print(f"Iteration {self.iteration_count} total time: {training_time} seconds")
        print(f"Iteration {self.iteration_count} total bandwidth: {total_bandwidth} bytes ")
        # Store results for Pareto optimization
        iteration_result={
            'iteration': self.iteration_count,
            'clientID' :"client_1",
            'bandwidth': total_bandwidth,
            'training_time': training_time,
            'epochs': self.epochs,
            'batch_size': self.batch_size,
            'optimizer': type(self.optimizer).__name__,
            'model_architecture' :"model_2"
        }
        # Append result to CSV after each iteration
        self.append_results_to_csv(iteration_result)
        return model.get_weights(), len(X_train_lstm), {}

    def evaluate(self, parameters, config):
        evaluation_start_time = time.time()
        model.set_weights(parameters)
        loss, accuracy = model.evaluate(X_test_lstm, y_test_categorical)
        evaluation_time = time.time() - evaluation_start_time
        print(f"Iteration evaluation time: {evaluation_time}")
        return loss, len(X_test_lstm), {"accuracy": accuracy}
    def append_results_to_csv(self, result):

        df_result = pd.DataFrame([result])
        file_exists = os.path.isfile('client_results.csv')
        df_result.to_csv('client_results.csv', mode='a', header=not file_exists, index=False)


client_1 = FlowerClient(epochs=15, batch_size=16)

fl.client.start_client(server_address="127.0.0.1:8080", client=client_1)
```

APPENDIX D SERVER.PY

```python
import csv
import flwr as fl
import time
def save_accuracy_to_csv(accuracy):
    try:
        with open('client_results.csv', mode='r', newline='') as file:
            lines = file.readlines()
    except FileNotFoundError:
        lines = []

    # Modify the last line and the previous line by appending the accuracy
    if len(lines) >= 2:
        # Append accuracy to both the last and second last lines
        lines[-2] = lines[-2].strip() + f",{accuracy}\n"  # Modify the second last line
        lines[-1] = lines[-1].strip() + f",{accuracy}\n"  # Modify the last line
    elif len(lines) == 1:
        # If there's only one line, modify it
        lines[-1] = lines[-1].strip() + f",{accuracy}\n"
    else:
        # If the file is empty, add the accuracy
        lines.append(f"{accuracy}\n")
    with open('client_results.csv', mode='w', newline='') as file:
        file.writelines(lines)

# Define a function to calculate a weighted average of accuracies from multiple clients
def weighted_average(metrics):
    # Calculate weighted accuracies by multiplying each client's accuracy by the number of examples they used
    accuracies = [num_examples * m["accuracy"] for num_examples, m in metrics]
    # Extract the number of examples from each client
    examples = [num_examples for num_examples, _ in metrics]
    # Calculate the overall accuracy by dividing the sum of weighted accuracies by the total number of examples
    accuracy = sum(accuracies) / sum(examples)
    # Save the calculated accuracy to a CSV file for logging or analysis
    save_accuracy_to_csv(accuracy)
    # Return the weighted average accuracy in a dictionary
    return {"accuracy": accuracy}

# Start the Flower server for federated learning
fl.server.start_server(
    server_address="0.0.0.0:8080",  # Specify the server address and port
    config=fl.server.ServerConfig(num_rounds=5),  # Set the server to run for 5 rounds of federated training
    strategy=fl.server.strategy.FedAvg(
        evaluate_metrics_aggregation_fn=weighted_average,  # Use the custom weighted_average function to aggregate accuracies
    )
)
```

APPENDIX E PARETO OPTIMALITY AND PARETO FRONT

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset from the provided CSV file
file_path = '/Users/neginkeshavarz/vsCode/client_results.csv'  # Update the path as needed
data = pd.read_csv(file_path)

# Convert columns to numeric where necessary
data['total_time'] = pd.to_numeric(data['total_time'], errors='coerce')
data['DataAmount'] = pd.to_numeric(data['DataAmount'], errors='coerce')
data['accuracy'] = pd.to_numeric(data['accuracy'], errors='coerce')

# Drop rows with NaN values in the critical columns
data = data.dropna(subset=['total_time', 'DataAmount', 'accuracy'])

# Print the data to check for diversity
print("Data after cleaning:")
print(data)

# Extract relevant columns for Pareto optimization
objectives = data[['DataAmount', 'total_time', 'accuracy']].to_numpy()

# Function to determine Pareto front
def pareto_front_maximize_accuracy(points):
    is_dominated = np.zeros(points.shape[0], dtype=bool)
    for i in range(points.shape[0]):
        for j in range(points.shape[0]):
            # Check if point j dominates point i based on three metrics:
            if (points[j][2] >= points[i][2] and  # j has higher or equal accuracy
                points[j][1] <= points[i][1] and  # j has lower or equal total time
                points[j][0] <= points[i][0]):    # j has lower or equal data amount

                # j dominates i if it's strictly better in one or more metrics
                if (points[j][2] > points[i][2] or points[j][1] < points[i][1] or points[j][0] < points[i][0]):
                    is_dominated[i] = True  # Mark i as dominated
                    break  # No need to check further for this point
    return ~is_dominated

# Determine Pareto optimal points
pareto_mask = pareto_front_maximize_accuracy(objectives)
pareto_points = data[pareto_mask]
```

```
print("Identified Pareto optimal points:") # Print the identified Pareto optimal points
print(pareto_points)

final_pareto_points = pd.DataFrame(columns=data.columns) # Further refine the points to ensure only non-dominated points are saved

for index, row in pareto_points.iterrows():
    # Check if the row is not dominated by any already in final_pareto_points
    if not any((final_pareto_points['accuracy'] >= row['accuracy']) &
               (final_pareto_points['total_time'] <= row['total_time']) &
               (final_pareto_points['DataAmount'] <= row['DataAmount'])):
        final_pareto_points = pd.concat([final_pareto_points, row.to_frame().T], ignore_index=True)

# Print the final Pareto optimal points
print("Final Pareto optimal points:")
print(final_pareto_points)

# Save the Pareto optimal points to a CSV file
pareto_output_path = 'pareto_optimal_points.csv'  # Specify the output file name
final_pareto_points.to_csv(pareto_output_path, index=False)
print(f"Pareto optimal points saved to {pareto_output_path}")

# Create a simple 3D scatter plot for accuracy vs. training time
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = data['total_time'] # Define axes for the plot
y = data['DataAmount']
z = data['accuracy']

ax.scatter(x, y, z, c='r', marker='^', alpha=0.5, label='All Points') # Plot all points for reference

# Plot the final Pareto optimal points if any exist
if not final_pareto_points.empty:
    x_pareto = final_pareto_points['total_time']
    y_pareto = final_pareto_points['DataAmount']
    z_pareto = final_pareto_points['accuracy']
    ax.scatter(x_pareto, y_pareto, z_pareto, c='b', marker='o', label='Pareto Optimal Points')

# Set labels for the axes
ax.set_xlabel('total_time (seconds)')
ax.set_ylabel('DataAmount')
ax.set_zlabel('Accuracy')

ax.legend() # Add legend

plt.show() # Show the plot
```