Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

A Feasibility Study For the Implementation of Machine Learning Models using Edge Federated Learning in Healthcare Domain

John Joy, Bachelor of Technology

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Meriel Huggard

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

John Joy

August 19, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

John Joy

August 19, 2022

A Feasibility Study For the Implementation of Machine Learning Models using Edge Federated Learning in Healthcare Domain

John Joy, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: Meriel Huggard

The usage of smart monitoring devices has increased drastically. It has been instrumental in generating rich data to enable the use of Artificial Intelligence and Data Science in healthcare. The data generated is sensitive in nature. Federated Learning(FL) has been gaining considerable attention, since it could train machine learning models without uploading data onto a cloud, keeping in mind the privacy of users especially the patients in healthcare. The Federated Averaging algorithm proposed for aggregation in Federated Learning, could add a significant overhead with the presence of large number of devices in the network. A novel paradigm, Edge Federated Learning(EdgeFed) was introduced with Edge Computing as an inspiration to better adapt for the upcoming future of AI and networking. This work details the usage of Edge Federated Learning, for the training of two different models, a linear model - Logistic Regression and a Neural Network model, using the diabetes data. Data was manipulated to simulate a real-world scenario and distributed to clients for training. The evaluation of the trained models are discussed in detail which proves that it is feasible to use Edge Federated Learning in healthcare. The results portray comparable results with conventional Federated Learning and conventional central training of ML models. Finally, it also establishes that Edge Federated learning can significantly reduce the number of global communication compared to Federated Learning.

Keywords: Federated Learning, Edge Computing, Edge Federated Learning, Flower Framework, Machine Learning.

Acknowledgments

I would like to thank my Appa and Amma for the love and support, for being the reason for what I am today, and all my friends for being the inspiration and being part of all the good times, with special mention to 185 boys and Helen for proof reading my dissertation and to all the closest of closest friends you-know-who.

I would also like to specially thank my supervisor Meriel Huggard for being the constant motivator throughout the course, and being the cool supervisor. Thanks to you for all the support, guidance throughout the dissertation.

John Joy

University of Dublin, Trinity College
August 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the information age and technology advancements, the capability to generate, store, and process data has increased massively. The world has experienced a data generation of nearly 62 zettabytes in the year 2020. The current situation with pandemic has made the data generation prediction in the coming years to be more than 180 zettabytes [21]. The volume of data has been instrumental in further advancements in AI and Data Science, enabling researchers and engineers to develop systems that could aid human beings in their day-to-day life. Machine learning solutions have helped in changing everything such as the user interfaces, weather predictions, healthcare, logistics management, and optimizing various things in our day-to-day lives. However, this has raised concerns regarding the privacy of the users. Conventional Machine learning systems collect data from millions of devices into a central server giving dangerous powers to the data collectors. Incidents have been identified and widely discussed about how users data have been exploited especially the biggest technology companies that has motivated to create toughest of the data privacy such as GDPR.

Federated learning, a decentralized machine learning technique, was introduced to mitigate such issues. Modern-day devices and smartphones have a high computational capacity, aiding such an implementation of distributed computational tasks. Additionally, it has enabled devices to work collaboratively to train models without sharing the data with the central cloud, thereby giving the user, control over the data. Federated learning gained attention for its ability to share global models as well as its capability to personalize models over rich but sensitive data.

Edge computing is one of the novel architectures that could bring a huge impact with the introduction of ultra fast fifth generation of networks.The number of devices getting connected to the internet are increasing day by day especially with the introduction of Internet of Things (IoT). Some of the client devices on the edge are capable of doing

better computational tasks and some incapable of doing any computational tasks but can only be used for data collection. In such cases if all the data is send over the internet to a central cloud the load on the internet would drastically increase. Edge computing paradigm has been introduced to mitigate the such issues by bringing in provision to offload tasks on to the edge of the network, there by reducing the load.

1.1 Motivation

Inspired by the novel architecture of edge computing, the future in distributed computing, and federated learning, a novel technique in machine learning for model training in a collaborative decentralized approach, edge federated learning was proposed by the paper [26].

The motivation for study comes with gaining popularity of federated learning in a medical community as discussed in various papers and conferences, with notable ones discussed are [22] ,[12]. There has been a huge scope discussed for federated learning overall, especially in the healthcare, due to the sensitive nature of the data. Although it is understood that the data is sensitive it is still required for the training machine learning model in conventional techniques, since there is a huge value added by data science and AI in the field of healthcare. Hence the solution, federated learning in healthcare.With the introduction of the Internet of things (IoT) for monitoring health of individuals different forms of devices such smart monitors, smart watches, smart bands, smart phones have been introduced.This can immensely increase the number of devices connected to the internet. Rather than a simple federated learning setup, edge federated learning could aid immensely in the working of devices over the internet, if necessary, with lower latency, better accuracy of machine learning models and better privacy. This could change the future of health care by bringing in better AI to the edge.

The paper [26] discuss about MNIST data which is the dataset for handwritten numbers and also the implementation of a Convolutional Neural Network on the Edge Federated Learning. However, these results cannot be compared to a healthcare data and the nature of models that will be used frequently in health care. The motivation of the dissertation comes from the idea of edge federated learning and to assess it's capability in training a machine learning model using a healthcare data.

1.2 Research Problem

The objective of the dissertation is to design, implement and closely evaluate the model using edge federated learning and compare the results with federated learning and centrally trained models. Following the research objectives set as the scope for this dissertation .

- Obtaining a dataset in healthcare and to do necessary manipulation to the data so that it better simulates the real world scenario for the study.
- Designing and implementing an edge federated learning setup that could be used for simulation, to better study the architecture, and at the same time should be capable of extending the same to a real world scenario.
- Training variety of machine learning models using the proposed architecture and to compare and study the results with the existing federated learning and central machine learning techniques of training, especially using the metrics that are relevant in machine learning when used in healthcare. The primary aim is to understand if the metrics of model generated using edge federated learning are comparable to that of the models generated using existing techniques.

1.3 Dissertation Outline

Following are brief description for upcoming chapter in the dissertation.

- **Chapter 2:** Background and Literature review - This chapter discusses about the Literature review of papers and theory for concept like federated learning, edge computing, Edge federated Learning, Various Frameworks and it's theory and machine learning models, required for the understanding of design and implementation of the dissertation.
- **Chapter 3:** Design and Implementation - This chapter put out the details about data simulation, especially for a data not designed for training on edge federated learning ,the design of the proposed architecture, it's implementation using flower framework, it's working and the challenges faced during the implementation.
- **Chapter 4:** Evaluation and Result - This chapter discusses about the evaluation methodology and the results of the model trained using the healthcare dataset and its critical analysis. It also establishes that the edge federated learning is a feasible paradigm in healthcare.

- **Chapter 5:** Conclusion and Future works - This chapter concludes the dissertation with the reflections and lays out the scope for future work, the notable one - study of scaled system using large number of clients and large dataset.

Chapter 2

Background and Literature Review

This chapter explains in detail the theory that is necessary for the design of a system to assess the feasibility of a edge federated learning as a paradigm in healthcare. Initially the aim is to discuss federated learning, a novel machine learning technique that serves as the underlying technique for the whole study. Further the chapter discusses about the edge computing that is gaining attention with the latest generation of networks and a necessity to manage the data driven age. Combining federated learning and edge computing, it further discusses about an architecture of edge federated learning that has the combined benefits of both the technology. Consequently it discusses about different frameworks that aids in the implementation and simulation of the entire architecture and the possible challenges related to it. Then it goes on about discussing the different machine learning models that are required for inferencing tasks to study the feasibility of machine learning models trained over edge federated learning network in the field of healthcare. Most importantly it discusses in combination with the about the underlying literature that has significantly contributed for the evolution of such a study in the dissertation.

2.1 Federated Learning

Federated learning (FL)[17] is a decentralized machine learning approach where the devices participating, learns collaboratively from the data distributed over the edge devices. In this paradigm, these devices learns a shared global model by training the model locally from the rich sensitive data and then aggregating without having to upload any data to a central cloud. The process is orchestrated by a central server which initially shares a central model to the participating clients, and this model is locally trained using the data on the device after which only the resultant model is sent to the central server for collective aggregation there by updating the global model and is illustrated in the figure(Figure

2.1). This architecture thereby limits the issues related to data privacy and attacks on the user data, since no real data is sent over to the cloud. Additionally, the costs associated with the data storage and data transfers can now be drastically reduced.

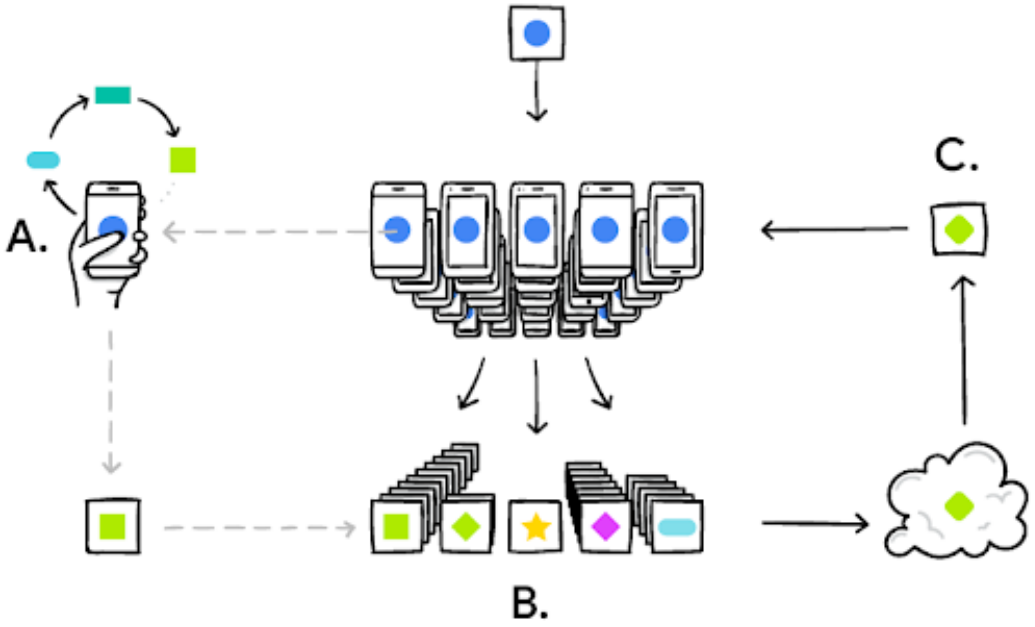


Figure 2.1: The figure gives an overview about the federated learning in devices A. The devices are locally updating the model that was initialized and shared globally using the locally generated data. B. The locally trained models are aggregated to form a shared model. C the shared model is then again send to client. The process is repeated further again [17]

As an effort to comply with data privacy laws, such as GDPR, researchers Brendan McMahan et al [18]. from Google introduced this technique in the year 2016 which has led to introducing federated learning into varieties of applications, a notable advancement - Gboard from google[17], where all android devices rely on a federated learning technique to train next word prediction model. All the sensitive text data generated from the keyboard is used to train the model locally and the resultant model alone are shared for aggregation. This comes with advantages such as, not having to share sensitive data ,allowing a new user to have an intelligent keyboard, and the current users to have a personalised trained model from large dataset.

One of the main technical contributions in federated learning obtained from the paper from the researchers at Google was introduction of FederatedAveraging algorithm ([18]), which is a combination of stochastic gradient descent on client for the training of the

model locally and then performs model averaging on the server. The basic assumption in machine learning is that the dataset should be independent and identically distributed (iid data) however the data generated on various edge devices would not follow the same distribution of independent and identically distributed data (non iid data). FederatedAveraging algorithm however, works robustly even with the unbalanced and non-iid data and has been proved to work efficiently with 10x-100x times less rounds of communication compared naive federated approach. In effect the algorithm has contributed to a robust approach with better data privacy for the user in an efficiently.

2.1.1 Stages of Federated Learning

Federated learning could follow slight changes in step depending on the architecture, but in a generalizing FL technique as such following are the steps in Federated Learning: [16]

- Based on different conditions, with the goal that the device's primary tasks are not affected the devices are selected randomly by the central server who is responsible for orchestrating the whole training process. Conditions could be,for example a smartphone when not in use or put on charge, are the times when the user experience is least impacted. In terms of IoT devices, devices might be offline most of the time, especially the ones deployed in remote areas, and when these devices comes online, these could be considered as clients that could participate in the training.

- A global model is initialized and the global parameters are shared with the selected clients which would act as the starting point for the local training of the model on the devices. Instead of assigning random parameters in the central training(weights and biases) these values are initialised from the common global model

- Based on the conditions specified by the central server, using the data that is generated on the local devices, it is used execute the training process. Parameters such as the number of rounds, strategy for training etc are mentioned and orchestrated by the central server to the clients and clients makes use of models designed to be light for edge devices to train lower number of epochs.

- The locally trained model from the client devices is send to the central server for aggregation. There are different attacks associated with model transfer that could have a impact on the aggregation, and hence different transfer protocols are enabled between the client and server for transfer. Additionally, different compression techniques are employed for reducing the bandwidth required on the network to transfer the model.

- The shared model from all the clients are aggregated on the central server to update the global shared model, and for the next set of training the aggregated model is shared with the devices locally for inferencing and training.

2.1.2 Federated Averaging Algorithm

To train a machine learning model the underlying idea is to use gradient descent.It then calculates the gradient in each step by evaluating gradient using all the available data from the dataset. However, with the increase in the size of the training data gradient descent can prove to be a computationally expensive and time consuming task. Stochastic Gradient Descent (SGD) was introduced as an optimization to the normal gradient descent where a single data point in randomly chosen to calculate the gradient leading to a less computationally expensive convergence. Apart from the batch gradient descent that takes all the training data into consideration, and true SGD which takes in consideration a random data point, another common form of approximation is mini-batch Stochastic gradient descent where instead of a single random data point chosen to compute gradient, a small batch of data points are randomly selected to calculate the gradient in each step of the training [24].

SGD being common choice of approximation for a training of a model these days, due to the faster convergence, it has been selected as the starting point for the federated optimizations. [18]. Here in a federated setting, SGD is applied in terms selection of a set of clients, whereas with the data inside the client, a batch gradient descent takes place. Then randomly selected set of clients collectively calculates the gradient over specified number of rounds. This a synchronous training activity where the parameter C controls the number of clients to be selected.i.e. when C=1 all the clients are selected and this is said to have a promising result in data centers.[14]. This approach is considered the base-line and is termed as the federatedSGD or FedSGD [18]

A typical case of federated SGD assuming that a client k is selected, gradient is calculated by the client k is $g_k = \nabla F_k(w_t)$ where w_t is the current weight of the model. The local model is then updated as $w_{t+1} = w_t - \eta.g_k($ η is the learning rate) which is the model that is send to central server for aggregation. The central server aggregates the received updated model based on weighted average to create a resultant shared global model.

$$w_{t+1} = w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n}.g_k$$

where k is selected client η is the learning rate, and the gradient calculated by the client g_k and W_{t+1}^k is the updated weight from the client. This could be termed as the federated

averaging which has significantly optimized naive federated learning as mentioned in the paper [18].

2.2 Edge Computing

With the improvement in the technology especially in the field of electronics, newer domains such as Internet of Things (IoT) has come into existence where there are multitude of edge devices such as smartphones, intelligent and autonomous devices, monitoring devices,etc. These devices are now connected to the internet that has lead to the increase in data generation, more consumption of bandwidth of network, increased response time, especially poor privacy for the data generated when uploaded to the cloud . Edge computing was introduced to mitigate such issues that was faced on a typical cloud computing setup. In this paradigm, data processing happens to the edge of the network, rather than a central cloud there by making the entire architecture light. When distributed the architecture becomes a lighter since it only requires less storage especially since most of the devices today are far superior and has surplus computational capacity,and of all that can be utilized to optimize rather than overloading central entities to rely on storage and computation.[13].

Edge computing comes with variety of advantages such as real-time processing of data,especially in situations where immediate results are required edge computing could be really useful, rather than sending data to a cloud for inference, when considered on a scale could put an immense load on the whole network that could even delay the results. Consequently, the privacy and security involved in the cloud computing is really weak since the real data transfers occur it is more prone to attack. On edge computing these data since it resides within the devices or the edge network the plane for attack is now limited. Finally one of the most relevant topic would be energy efficiency between edge computing and cloud computing. When more devices are connected in a cloud computing setup more energy is required when compared to edge computing.

2.2.1 Components of Edge Computing

Edge computing layer consists of three layers, terminal layer, boundary layer and cloud layer and its functions are as follows [13] It is illustrated in the figure2.2

- Terminal Layer - Terminal layer is predominantly terminal devices such as Internet of Things(IoT) devices that are responsible for data gathering. These devices themselves aren't computationally very capable but are the main sources for collecting

the data. These devices are capable of doing smaller computation or data processing before sending on to the boundary layer. The data collected are uploaded and processed on the upper layer.

- Boundary Layer - This layer has better computational capacity and the added advantage of being closer to the user, the data source. Otherwise called the edge layer it could consist of components in the network such as servers, switches, routers, gateways, access points or base stations and it acts as the core layer in the whole paradigm. It is capable of conducting intelligent tasks and real time analysis on the data. Consequently it comes with the added advantage of being connected to the multitude of edge devices as well as it has higher bandwidth capacity to communicate with the cloud layer of the network which is an added advantage since most of edge devices might not be capable of better connection with the cloud layer. It is also responsible for storing sensitive data.

- Cloud Layer - Since trivial task do not require cloud could it could be easily offloaded to the boundary layer there by reducing the overload on the network, however cloud layer still is the most powerful layer where there are more capable servers and data centers capable of high intensity tasks and much needed information can still be stored. It can also act as the layer where all the data could be permanently stored if deemed necessary and for wider analysis cloud layer serves as a key layer.

2.3 Edge Federated Learning

Edge federated learning a combination of edge computing and federated learning is the novel paradigm that is aimed to study in this dissertation. With concerns in data privacy at it's peak researchers working on newer architectures keeping privacy is mind, federated learning is gaining massive attention since the data is never transferred but only results are passed. Although it doesn't completely comply with the GDPR laws for privacy it still helps increasing the privacy of the users. Additionally, another architecture that is gaining attention is edge computing. With the introduction of 5G, researchers and engineers are seeing immense opportunities with the combination of both the technologies [4]. 5G, an extremely fast network aims on achieving 1ms latency to any ping request made in the network. [5]. Edge computing will turn out to be an essential factor when it comes to fifth generation networks. With the amount of devices, especially in the era of IoT, connected to the internet can slow down the internet. Edge Computing could the key solution when introduced since it can drastically reduce latency since it is capable of

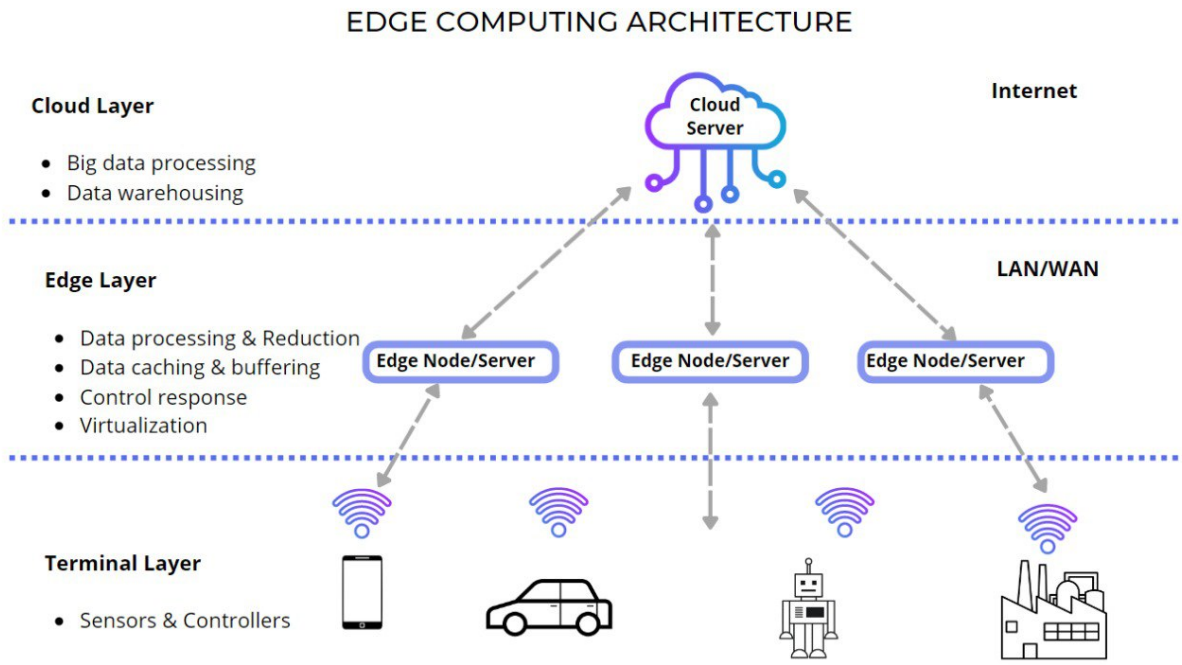


Figure 2.2: This figure illustrates the function of different layers in edge computing

offloading majority of the tasks over internet to edge devices. It was discussed in section 2.2. Hence any novel paradigm in the present day approach would require edge computing to be necessary component in designing the architecture. Inspired by it, edge federated learning has been proposed in [26]. Edge federated Learning (EdgeFed) comes with variety of advantages such as offloading training tasks from the mobile or client devices that has comparatively lesser computational power to a more capable edge layer. Moreover, the cost of global communication will be drastically reduced in this paradigm.

2.3.1 Components in Edge Federated Learning

[26]

EdgeFed which is a combination of both edge computing and federated learning. It has 3 layers similar to that of a edge computing architecture. Cloud layer consists of global server. Edge layer consists of a series of edge servers at each edge network. The edge servers play a major role in managing edge clients and communicate more with the global server since it has a higher bandwidth communication. Terminal layer consists of edge clients that participate in the EdgeFed architecture. Following are the steps followed in the edge federated paradigm.

- Central server - A cluster of high capability servers provided by the internet service providers that are capable of doing much intense tasks that cannot be handled by an

edge network. These servers are located at the top of data link layer which witnesses instability in data transmission from the edge devices. Additionally huge amount of data transmission to the central server can lead congestion in the network.

- Edge Server - Edge servers are next in line that are deployed in the edge of the network. It has better computational capability compared to that of the edge client. It is considered to have stable power sources and the connectivity is sufficient. These are devices like routers, gateways, servers on edge, switches or any base stations

- Edge clients - Edge clients are devices like IoT devices, smartphones, etc that has limited computational power but are the closest to the user and hence is responsible for data gathering. These devices has limited access to the power and connectivity and hence only preliminary tasks are carried out on the edge client networks. Heavier tasks are sent over to the edge servers, in addition data is passed over so that better training results can be achieved due to larger collection of data reducing the non-iid behaviour.

2.3.2 Steps in Edge Federated Learning

For training of a model in this paradigm it involves two steps to it. The initial step is a training between the edge client and the edge server. The next step is aggregation of updated models in the edge server to the global server. The underlying working of both the stages are that of the federated learning, using the federated averaging algorithm. Following are the generic steps [26] followed in EdgeFed architecture.

- **Step 1** - Initial step is to randomly initialize the model in the global server and these model are shared with the edge server.

- **Step 2** - The shared models from the global servers are then shared to the edge clients from the edge servers. Edge clients are randomly selected based on different conditions such as the availability of the edge client, its power capacity, user experience etc. These randomly initialized model from global server is then passed on the clients.

- **Step 3** - The clients does a preliminary training of the model using the locally generated data. Once the model is trained locally the data generated from the edge device and the updated model itself is shared to the edge server. Since the edge server has access to more computational resource, tasks that are comparatively heavy such as further training of models are carried out on larger base of data.

- **Step 4** - The updated models alone are then send to the global server for aggregation. In both the cases where the updated models are shared federated averaging algorithm 2.1.2
- **Step 5** - The updated model is then shared with the edge server
- **continue to Step 2 until a desirable results are achieved**

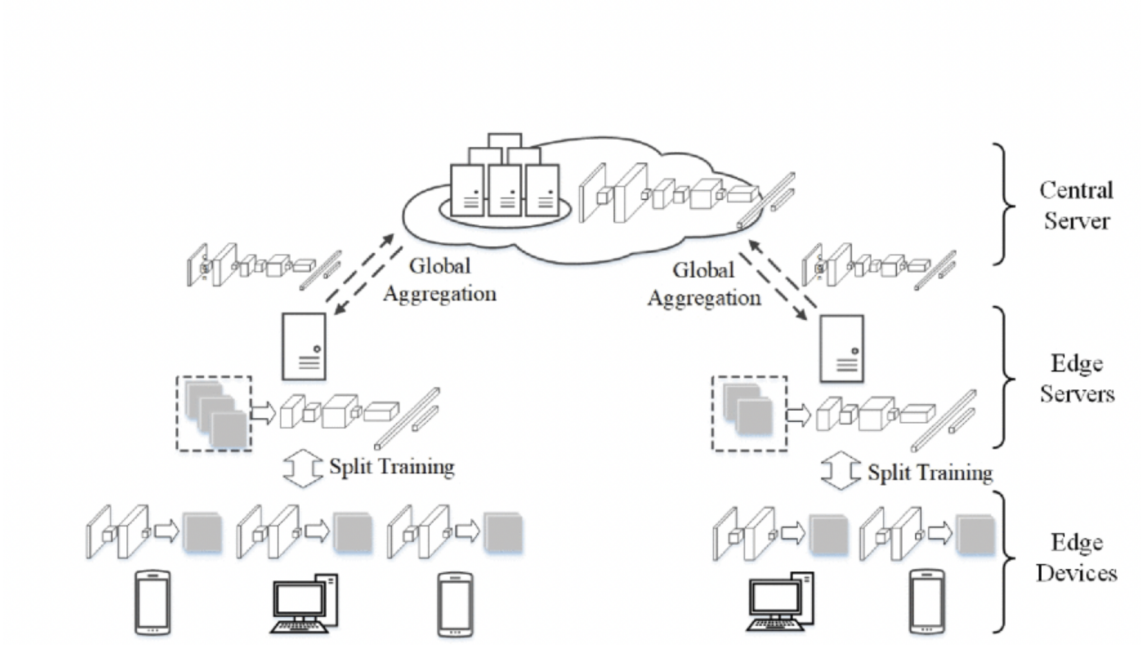


Figure 2.3: This figure illustrates the function of different layers in edge computing[26]

All the processes have been illustrated in the image 2.3

2.4 Federated Learning Frameworks

With the introduction of federated learning different frameworks have been introduced to aid abstraction of the complex computation and orchestration involved in the background, and are available as open-source projects for researchers to build on top of it. The notable ones are TensorFlow Federated and Flower Framework.

2.4.1 TensorFlow Federated Framework

TensorFlow federated framework[1][6]is an abstraction for the federated learning tasks for the models that are built using TensorFlow. It aids in evaluating federated learning

especially on existing model and data. TensorFlow Federated (TFF) consists 3 key parts. Models - helps in wrapping up existing models into a federated learning setup. Federated Computation Builders - that are responsible for training, evaluating the existing models and Data-sets - a collection of data-set that could be loaded in python that simulates the federated learning scenarios. All of these functionalities except for the simulation capabilities have been grouped into a module tff.learning and simulation as tff.simulation.

There are few architectural assumptions made by the framework, when implemented. Initial assumption is that due to the variety of devices taking part in the process such as high end servers or computationally restricted edge devices, the only safe requirement would be, the device should be capable of supporting TensorFlow on the run time and the existing graph build using TensorFlow should be serializable as a graph such that could be run on even lower end devices. The second one being, there are two aggregations that are taking place internally. A local aggregation and Federated Aggregation. Local Aggregation starts first starts with initialization of variables. It then invokes forward_pass function to update the initialized variables and report_local_unfinalized_metrics are invoked to report the results in each stages. The Federated aggregation consists of distributing the initial model, which is then independently or parallel run on client devices. Then the aggregation from multiple clients are invoked.

There are basically two components to TFF. A higher layer of an API could be employed to conduct researchers and simulation of the federated learning setup and the second lower layer is a federated core which is responsible for internal computation and evaluations. To customize the whole flow changes has to made to the Federated Core.As a framework TFF provides different modules and are mentioned in the table 2.1

Simulation using the TFF framework is the most important function that is aimed to be utilized in this dissertation. To better under TFF simulation module consists of mainly three parts:

- **tf.function** - is used to encapsulate the function without referencing tff but could be reused inside a client. These are normal tf functions that runs of client to train the model and could be used outside the framework.
- **tff.tf_computations** - is used to encapsulate the tf.functions and forms a federated wrapper for training and evaluations.it is also responsible orchestrating of functions inside tff.tf_computations using functions tff.federated_broadcasting and tff.federated_mean inside the given module
- **Outer Driver** - An outer script that is responsible for determining the control flow, such as selecting the clients etc determines the whole flow of the system.

aggregators	that are responsible for the aggregation tasks in the federated learning in a setup
analytics	for analysis of the FL using TFF
async_utils	a python utils file for enabling the asynchronous orchestration of the training
backends	This used to for all the backend activities such as compiling,constructing and triggering computations
frameworks	It is a module that could be used to extend the existing framework to a new one
learning	there are various FL algorithms and to choose between different algorithms this module is used
profiler	This module is used to as a utility function to time and data tracing
program	It is a set of libraries used to create different federated program
simulation	this used to create simulation with the intended setup
structure	instrument to create name or unamed fields
template	these are used to create templates for commonly used calculations
test	used to create tests in TFF setup
types	library types used to interface different functions and computations within the framework

Table 2.1: Different modules in TFF framework

TFF framework however could not be considered as an apt choice for this dissertation since it is restricted to limited use cases and are really inconvenient to create a custom architecture such a edge federated learning which require multiple layers of federated learning incorporating edge computing.TFF provides means to work on researching different federated learning algorithms and simulating its effect. Additionally TFF only provide simulations to TensorFlow models alone whereas the dissertation aims on exploring effects on variety of models using EdgeFed. Again although simulation satisfies the feasibility study it never gives a provision for deployment of real architecture, and would then require a need to search for another framework for the same.

2.4.2 Flower Framework

Flower framework [7] is an open-source end-to-end framework that enables deployment of federated learning in real edge devices rather than just simulations using experimental setup. Flower Framework is considered a stable setup that can easily integrate existing machine learning models and frameworks, and flower framework can be easily extended. Novel and emerging algorithms that are need to be implemented could be easily incorpo-

rated, as well as flexibility to choose between different training strategies is a plus. Finally different communication protocols could be employed to secure communication between clients and servers. Flower framework could be used for both research and simulations as well as production ready setup in real edge devices.

Flower framework helps in closing gap in various use case that are not supported by current FL frameworks. Following are the use cases that could be enabled by Flower framework

- **Scaling** - Flower framework is capable of scaling the experiment with large number of client, especially in a federated learning setup algorithms have to be tested out on large pool of clients concurrently to test out it's performance on generalizing using an algorithm.It can also be used to test out a federated learning design based on different set of compute.

- **Heterogeneous Devices** - Flower framework is supported on variety of devices and the same can be used to test out heterogeneity. Researchers make can make use of flower framework to simulate federated learning on heterogeneous devices as well as on real edge devices. Measurements could be easily evaluated using the framework especially the performance of the design.

- **Transition** - Flower framework can be used to initially study the design and algorithms. Once the design can be studied and established, it can be easily transitioned to real world devices.

- **Support of multi frameworks** - The framework naturally motivates usage various ML frameworks. Various edge devices might have models using various machine learning frameworks. For example, some of the edge devices might have models usage implemented in PyTorch and some of them might have models implemented on TensorFlow. Flower framework can be easily used in situations like enabling aggregation of models from different client centrally.

The frameworks that exist in federated learning has ecosystem gaps and mainly doesn't reflect the real world scenarios and hence when developing flower framework has independent goals that addresses these issues. Following are the goals

- scalability - Federated learning when deployed in real world would mean that there would be large number of edge client devices. The flower framework has been developed keeping scalability in mind.

- Heterogeneity - Different edge client devices work on various OS and flower framework has to support heterogeneity.

- Communication heterogeneity - Different devices in real world has to communicate through different protocols and the framework has to support the same.
- Privacy friendly - Different privacy and updated privacy schemes has incorporated on the framework to support real world scenarios
- Flexible - the framework has to be flexible incorporate different designs of federated learning.

The core architecture of flower framework consists of mainly two components - the server side and the client side. The server orchestrates the whole process in the flower framework. The server consists of mainly three components. Client Manager, Federated learning loop and a user customizable strategy. The server selects the clients and samples using the ClientManager. It manages the clients as objects as ClientProxy and it represents a client connected to the server. The FL loop is the key element in the whole FL process. Strategy is a user customizable layer where it can used to determine the whole process. The client side consists of two main components Virtual Client Engine(VCE) and Edge client engine. The virtual client engine enables virtualization that has utilized hardware that is available. VCE initializes the clients in a hardware aware scenario. Edge client manages various Configurations of the devices that participate in the FL and it manages the connection with the server. This is illustrated in the figure 2.4

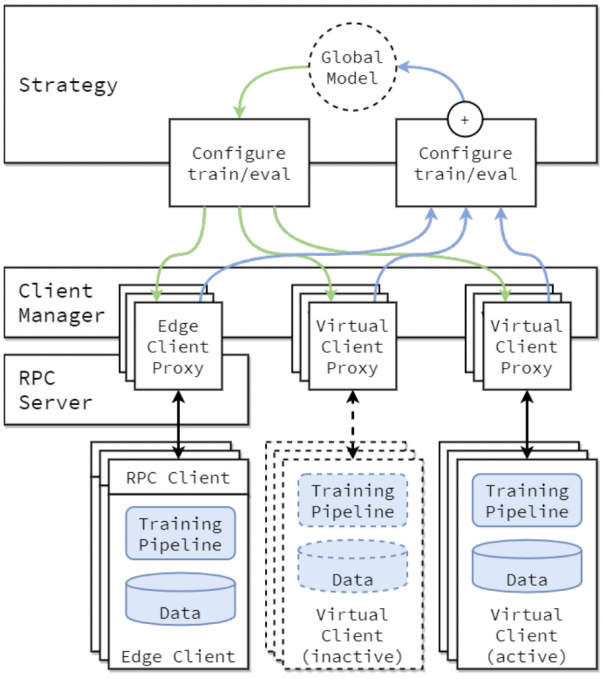


Figure 2.4: This image illustrates the core architecture of the flower framework.[7]

2.5 Machine Learning models

Machine learning is a field in computer science that learn from the given data to be able to predict outcome of an event without being told it's outcome. There are 3 types of machine learning. Supervised, unsupervised, and reinforcement learning. In this dissertation considering the medical data that has been selected the primary focus is on supervised learning where the label are mentioned unlike unsupervised and reinforcement learning. Classification is sub division of the supervised learning where the models are trained with categorical values as the label or the entities that needs to be predicted. In this study the aim to train different classification model using different methods(centrally trained model, federated learning and edge federated learning). The models under consideration are logistic regression which is a linear model, neural network capable of learning complex non-linear behaviour of the data as well as a tree based model that has better accuracy in classification problems. This sections discussed in detail about these models.

2.5.1 Logistic Regression

Logistic regression[2] is a simple yet powerful tool used most in for binary classification problems (the output expected is a categorical value). The logistic regression model is a modification of linear regression model that predicts continuous values ,whereas logistic regression predicts either 0 or 1. The output is mapped to range between 0 and 1 using using a non linear function called sigmoid function. Since the predicting function is sigmoid the basic assumption in the case of linear regression where the features have be distributed linear doesn't hold anymore. Logistic regression model can be represented as

$$f(z) = \frac{1}{1 + e^{-z}}$$

where z can be represented as the linear function in linear regression $z = \beta_0 + \beta_1 x$ where is the feature. The function visualization is represented in the figure 2.5 .

When the value of f(z)¿0.5 the classification output is considered 1 and when it's is less than or equal to zero the classification output is considered 0. The training of the logistic regression is started with random initialization of the β that are the coefficients of the linear functions as discussed above. This is then given to the sigmoid function and a prediction is made. The loss function of the problem is a cross-entropy loss function which is represented as $-ylog(\hat{y}) - (1-y)log(1-\hat{y})$ where y is actual value and $\hat{y}$ is the predicted value. i.e when the value of the actual label is 1 $log(\hat{})$ becomes zero and loss is 0 and when the actual label is 0 $log(1-\hat{y})$ becomes 0. hence the loss is at minima. Otherwise the loss tends to go a large value. The cost function is a convex function and

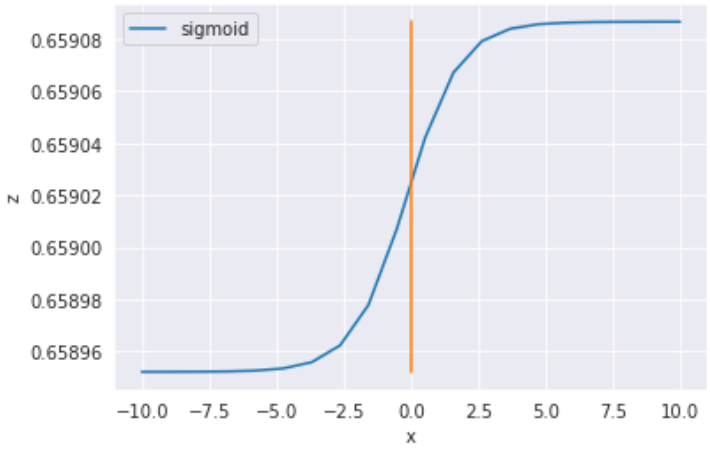


Figure 2.5: This image illustrates the sigmoid function graph

has a single global minima and it could be converged based on a gradient descent problem. Apart from batch gradient descent, advanced optimized could be used for converging the loss to the global minima.

For a multinomial classification[23], logistic regression cannot be directly used since it is only used for binary classification. For a multi class classification, logistic regression has to be built in a one vs rest methodology . For example if there are 3 classes to be predicted A,B,C there classifiers has to be made in such a way that one model has to classify class A and classes B and C and the next class and so on. Each class probability is calculated separately in the same like that in a binary classification. The class with the highest probability is selected as the classified class.

2.5.2 Neural Network

The idea of neural networks [19] were established previously, but only gained attention lately with the increase in computational capability and the availability of the data with the advancement in electronics and technologies associated with it. Neural networks were formed as an imitation of human brain cells and are attempted to design and function the same way. Just like brain cells neural networks are made up of nodes which has weights and biases and the activation functions.Neural network essentially consists of three major layers, input layer, hidden layer and output layer. Each layer is made up of nodes. Each node calculates the or weighs in the feature importance in a given set of inputs using the weights and biases are selected based on the activation functions. There are different type of activation functions used in neural network. Sigmoid activation function returns 0 or 1, are usually used in the final layer to predict binary classification problems.However sigmoid functions are not optimized activation functions since the range doesn't give

space for wider range of values including negative values and hence tanh was adopted. tanh values varies from -1 to +1 and has been considered as better activation function in the hidden layers, but still the range of values converted are limited and hence the ReLu(Rectified linear unit) activation functions were adopted that covers wider range of values and as an optimization leaky-ReLu values were introduced to cover negative values.The input layer varies based on the use case, depending on the neural network design, such as CNN (Conventional Neural Network) however in a general case the input layer only passes the input attributes to the hidden layer. The hidden layer and the output layers are responsible for computations.

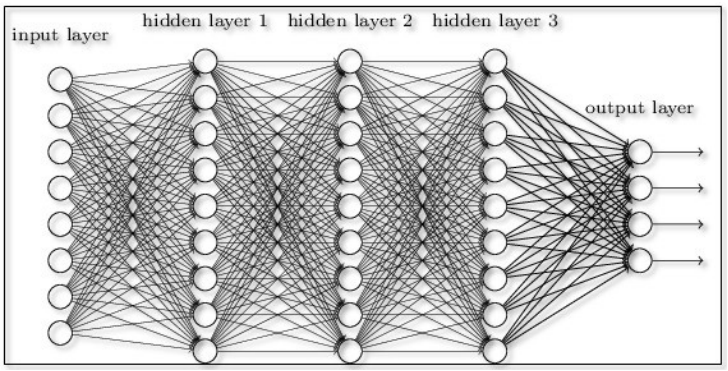


Figure 2.6: This image illustrates the architecture feed forward network

Initially the weights are assigned randomly and the information are passed over from layer to layer in a forward fashion and it is termed as feed forward network 2.6. Here no changes are weights or biases are made or no learning is made. The weights and bias are learned or changed based using the backpropogation where errors are calculated and this is propagated to make minor changes in the weights and bias until the error from the label and the predicted values are made the least. The training process is a minimization function and it is minimized using gradient descent optimization. Neural networks are known for it's ability to learn complex non linearity in the data creating better decision boundary however requires large amount of data to converge into global minima. Neural Network is best suited architecture and is integratable in flower framework and hence it is one of the models that is considered for the feasibility study in this dissertation.

2.5.3 Random Forest

Random Forest [11] is normally used as a supervised classification model and it is made up of ensemble of decision trees. However it could be also used for regression problem. In random forest the results from multiple trees to arrive at a single result. It is known for

its easiness with which it can implemented and the accuracy that could be achieved in a classification. A illustration of randomforest can be seen in the figure 2.7.

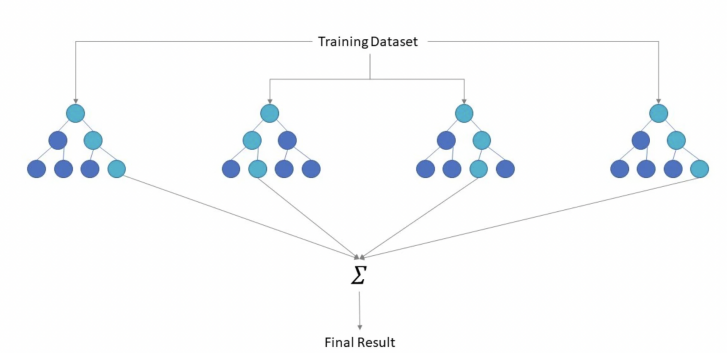


Figure 2.7: This image illustrates the random forest classifier

Decision trees are made out of a series of 'yes' or 'no' question that could be used to separate and classify the dataset. it training process of a decision tree is formulating the right split when a data comes in, based on its label. Each classification is represented by a leaf node in a decision tree. One of the common measure that is used to calculate the split is Gini index. Other measures like information gain and mean squared error (MSE) can be used to determine the split. The main of these measures in to minimize the impurity in the data based on the split. Regression mainly uses MSE to determine the impurity. Gini impurity mentions the probability of a misclassification of the data. When the gini impurity value is less the split is better. Although decision trees can be used for classification problem it can lead to problems such as bias towards the prediction as well as over-fitting based on the training data. However, when the a cluster of decision trees are used it can create better decision boundaries on a condition that separate set of decisions trees should has less correlation or no correlation.

Ensemble method is a learning technique where a set of classifiers are involved and the most popular result is selected based on certain conditions which is an aggregate. A popular method of ensemble is called bagging method[9]. In bagging method the data points are sampled multiple time. Each data point could be sampled more than once so that it appears in multiple sample. The samples are used to train, in this case multiple number of decision trees. The results are averaged and this method is known for reducing the variance in the split of the data.

The random forest classifier makes use bagging method and decision tree. Decision trees uses all attributes to train a single model. However in random forest bagging is used as well as feature randomisation [10] method is used where the features with least correlation is used to select random set of attributes. There by the random forest uses a bunch of decision trees that are least correlated on subsets of attributes or features rather

than all the data features like in decision tree.

However this model implementation and training using edge federated network is beyond the scope of this dissertation. Federated learning implemented random forest [25] has higher level of the trees in central servers whereas the lower depth layers reside in the clients. Implementing the same using flower framework is not directly supported and hence would require additional efforts.

2.6 Summary

Federated learning and edge computing as two separate technologies have been adapted by researches and engineers in various solutions. One of the notable and introductory solution for federated learning is Gboard from google. In Gboard the next word prediction model trained is trained using the federated learning technique. Federated learning as a technology has been adopted by various cloud solution providers especially Amazon Web Services. federated learning gained it's popularity especially with the IoT devices. The idea of federated learning is mostly promoted by Google. Edge computing is promising novel technique which would gain significant attention in the coming days especially with the introduction of fifth generation of networks. IBM is aiming to develop technology immensely in the field of edge computing. Most of the studies were conducted separately and very few studies especially [26] have only combined federated learning and edge computing where the multiple layer of federated learning is discussed.

This chapter discusses the federated learning and edge computation is detail with the aim of helping the reader to better understand the novel paradigm of edge federated learning. Combing the ideas edge federated learning have been closely correlated and elaborated. Popular frameworks such as TensorFlow federated learning and Flower Frameworks are discussed thoroughly. Its components are discussed closely so that it gives a better idea about the most aligned choice for the design and implementations. The chapter further explains the overview of machine learning models Logistic Regression, Neural Networks and Random Forest that needs be trained using the paradigm which is 3 different category of models available to better understand the performance of the technique in its diversity. The chapter enables the reader better familiarize with the necessary background required for the design and implementation of, proof of concept, that is to be discussed in the following chapter.

Chapter 3

Design and Implementation

Design and Implementation chapter of the dissertation explains in detail the development of a system responsible for training of machine learning models using data in healthcare domain using Edge Federated Learning (EdgeFed) with references to the ideas set and discussed in the previous chapter, Literature Review 2. This chapter begins with detailed explanation of Edge Federated Learning (EdgeFed) architecture independently on a implementation perspective. Consequently the chapter discusses about a generalized components, in Flower Framework development. Moving ahead in the mainstream direction of a machine learning problem, the dissertation explains about the data, the architecture and how it is implemented using the selected framework. Finally the chapter concludes with the challenges encountered during the implementation in consideration with proposed design.

3.1 Design Overview

The core idea of the whole implementation is the ability to preserve the data privacy of the users by never having to share the data with a central entity but at the same time still being able to make use this data to create better machine learning models in the edge computing paradigm. Following are the components in the system proposed.

- Edge Federated learning simulation - The above discussed core idea about data privacy in machine learning training in edge computing paradigm, as discussed and chosen is Edge federated leaning. The crucial idea in design is the flow of machine learning models and data between the clients and servers and has to be discussed in detail to better understand the design of the architecture. Here to simulate data generated in real edge client devices, diabetes dataset has been chosen and imputed accordingly.

- Flower Framework - Flower Framework has modules that support the development of federated learning where only a central server and clients are considered. The implementation for edge federated learning is not directly available and hence with the available set of modules in the framework it has to be tweaked to create the proposed architecture. It has to be further used to obtain the evaluations of the model.

- Machine Learning models - Each model is implemented in a different methodology using the flower framework. The models used in the study are Multinomial Logestic Regression and Neural Networks. These models has to be trained and evaluated using three setups, conventional machine learning, Federeated learning and Edge Federated learning for comparison.

3.2 Edge Federated Learning

The basic idea of Edge Federated learning has been discussed in the previous chapter in the section 2.3. It discusses that the edge federated learning consists of basically 3 layers, Global server, Edge server and Edge client. It also explains the steps involved in the architecture where the global model selects a random weights and biases from a commonly agreed ML architecture. This model is then shared to edge server which then sends the same to edge clients. Edge clients updates the model weights and shares the updated model and the generated data with the edge server. Edge server then updates the aggregated model with the collected data from the clients. These resultant updated models are then shared with global server for aggregation.This could be mathematically designed and modelled as:

- The randomly initialized model shared with the edge server and the edge server in shares the same model with edge client. Random initialization happens initialized by the flower framework in the background.

$$w_t < -random\ initialized\ (global\ server) \tag{1}$$

- The edge client trains the model using the data that is generated on the edge client. Here g is the gradient, ∇F is the loss function, w_t is the weight shared by the global server through edge clients and η is the learning rate.

$$g = \nabla F\left(\omega_t\right) \tag{2}$$

$$\omega_{t+1}^{i} \leftarrow \omega_t^i + \eta g \tag{3}$$

- The updated weights of the model from the edge client is aggregated in the edge server

$$\omega_{t+1}^{i} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \omega_t^i \tag{4}$$

- Now the randomly initialized model weights from the global server is replaced with the aggregated weight from the client using federated averaging. i.e from equation 4 and 1 now we have

$$\omega_t = \omega_{t+1}^{i} \tag{5}$$

$$w_{pool}^{m} \leftarrow \left[w_{t+1}^{1},\ w_{t+1}^{2}, \ldots, w_{t+1}^{k}, \ldots, w_{t+1}^{K}\right]$$

.

- From the edge server using the collection of data received from the edge clients and the new model weight ω_t the server trains the model.

$$g = \nabla F\left(\omega_t\right) \tag{6}$$

$$\omega_{t+1}^{i} \leftarrow \omega_t^i + \eta g \tag{7}$$

.

- This model is then again shared with the global model and it is aggregated in the global server using federated averaging algorithm.

$$\omega_{t+1}^{i} \leftarrow \sum_{m=1}^{M} \frac{n_m}{n} \omega_t^i \tag{8}$$

- The new weights are obtained $\omega_t \leftarrow \omega_{t+1}$ and these weights are again reshared with edge server and edge clients through edge server until desired results are obtained.

This mathematical representation can represented as an algorithm for the implementation of the same programmatically. There are two sets of processes identified from the previous mathematical representation. There is a edge update and and there is an update between the edge network and the cloud layer. Hence separate algorithms has to be defined between the edge server-edge client communication and edge server-global server communication.

The local update in the edge network can be referred to as Algorithm 1. Assuming that there are m edge servers and each edge server has k clients. E is the number of rounds, the edge server will receive the aggregate from the edge devices and train the model within itself. B is the batch size used in the training.1

Algorithm 1 Local Update on Edge - Algorithm 1

Require: data from k edge clients and updated model from the client
Ensure: The model is updated

run the code on edge server
procedure EDGEUPDATE(:)
    for e in Epochs E from 1 to E **do**:
        for b in batch B **do**:
            for each client k in parallel **do**:
                $X_{client}, Y_{client}, W_t \leftarrow ClientUpdate()$
            end for// aggregate the model updated from the edge client based on the parameters.

$$\omega_{t+1}^{i} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \omega_t^i \tag{4}$$

            $W_{t+1} \leftarrow W_t$ - $\eta \nabla loss(X_{client}, Y_{client}, W_t)$
        end for
    end for
    return W_{t+1}
    send W to cloud servers
end procedure
//training of models in clients
procedure CLIENTUPDATE(:)
    for e in Epochs E from 1 to E **do**:
        for b in batch B **do**:
            $W_{t+1} \leftarrow W_t$ - $\eta \nabla loss(X_{client_i}, Y_{client_i}, W_t)$
        end for
    end for
    return W_{t+1}
    send W to edge servers
end procedure

The second algorithm is the global aggregation of the model from the edge server to the global server. The updated model from the edge client which is then trained again in the edge server will be sent to the global server and the aggregation of the models in cloud based global server is aggregated as the weighted average of model (Federated Averaging algorithm). Number of edge clients are indicated as m, E is the number of rounds run by the server, b is the local number of epochs. η is the learning rate. 2

Both of these algorithms work with the basic assumptions that all the models mentioned throughout has the same architecture. Hence the weights are denoted directly in the algorithm. Following the same model architecture is required since the parameters has to be the same in all the servers and clients. If there is a miss match, the whole

Algorithm 2 Global Update on the cloud - Algorithm 2

Require: updated model from the edge servers
Ensure: an updated global model in the cloud server // the code is run on the cloud (global server)

procedure GLOBALUPDATE(:)
  for (**do**each i in epochs E):
    for each m in M edge servers **do**:
      $W_{t+1} \leftarrow EdgeUpdate(k, w_t)$
    end for
    Global aggregation happens based on the parameters selected

$$\omega_{t+1}^{i} \leftarrow \sum_{m=1}^{M} \frac{n_m}{n} \omega_t^i \tag{8}$$

  end for
end procedure

communication and aggregation will fail. One of the major difference in the algorithm proposed in this dissertation is aggregation of models based on the number of weighted average of edge servers rather than communicating the number of clients involved in the training in comparison to taking weighted average of all the clients like mentioned [26]. This paradigm design of edge federated learning aims on aggregating the models as a step by step by approach.Models trained on the clients when aggregated on the edge server now acts as a starting point for the edge server and global server aggregation. This whole process is illustrated in the previous chapter 2.3

3.3 Flower Framework Development

As Previously discussed there are various frameworks available in the arena of federated learning that helps in either simulation or both simulation and deploying in various real edge devices. Tensorflow Federated Learning, LEAF, Flower Framework, Clara from NVIDIA all supports the implementation. However most notable ones are TFF (Tensorflow federated learning and Flower Framework. Flower Framework has been chosen for design and implementation due to factors discussed in 2. Flower Framework is basically developed for the implementation of the conventional federated learning. This section aims in putting forth different development modules so that it explains how the design of an edge federated learning is possible with this framework.

A Basic Flower framework structure is divided into two components. The client and the server. Each component is responsible for its own computation.Infact a simple setup

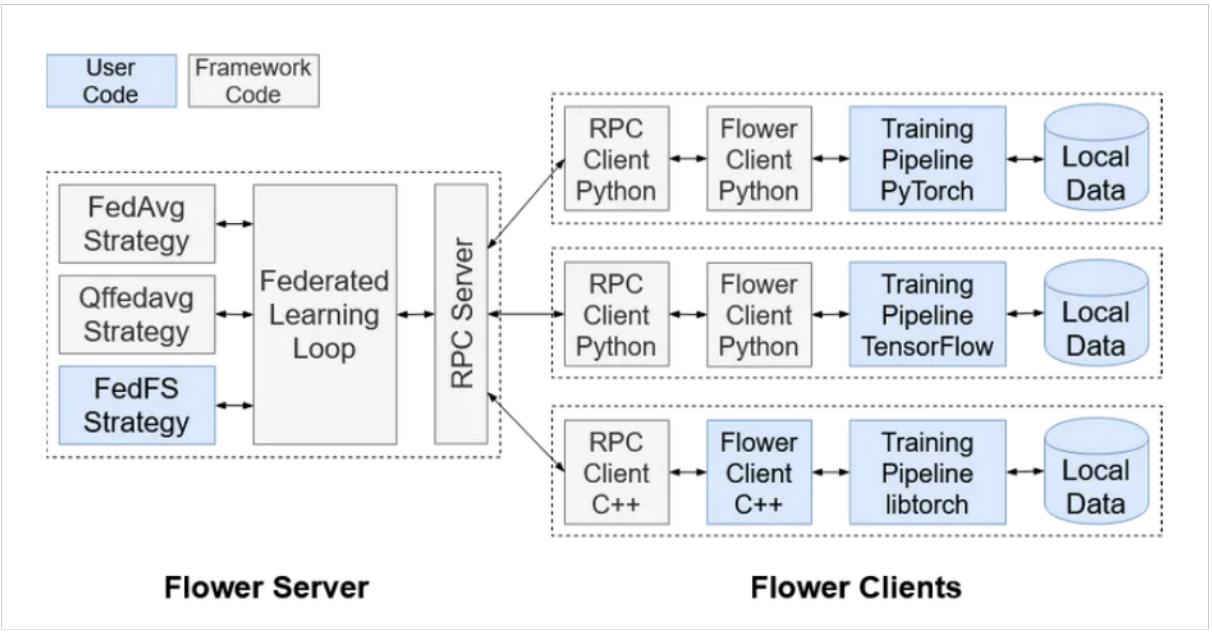


Figure 3.1: Architecture of multiple nature client of Flower Framework[7]

of a client and a server using this framework can be implemented using less than 20 lines of code. In essence there should be at least two different modules that has to be spun for the basic framework structure. Following are the implementation details of the client and the server.

- Client Code: The client script in the flower framework in designed in such a way that training of the model using the available data on the client and evaluation of the trained model can be set according to the use case or in other words is customizable. Flower Framework comes with a great abstraction and these modules in the framework is responsible for managing the search for the server, establishing a connection, serializing the model parameters and sending over to the server by trying to avoid communication failure. The client module that is generally used is a class named NumpyClient and is communicated with the server using numpy array or either dictionaries. This class can be used to as the base class in case a custom client module has to be created enabling to inherit the same properties on top of the custom properties. However the structure for implementation remains the same. The NumpyClient consists to basically four member functions and those are
  get_parameters - The function returns the model parameters such as weights and other necessary parameters
  set_parameters - The function is used to obtain the parameters from the server and initialize the model.
  fit - This function is used to train the model specified in the architecture.set_parameter

is called inside this function before a model is fit is carried out. This function returns serialized model parameters, the number of training points that took part in the training. The return gives an additional provision to return any custom data,but are mainly intended for metrics, however any data in a dictionary format could be sent over the server and has been used a tweak to enable data sending to implement EdgeFed

evaluate - This function when called return with the evaluation metrics that is specified within the function.

The whole process is initiated with by calling the core function start_client to start the client which searches for a server at the specified IP address. With the client flower class mentioned the core modules orchestrate the process.

- Server Code: Server plays a crucial role in the whole framework. This is the module that orchestrates the learning between client and then aggregates the updated parameter. The code involved in the implementation of a server is very abstract. It sets up a server on the mentioned port and selects the clients based on the strategy mentioned. Flower Framework module strategy can be used while spinning up the module to built custom strategies of training by inheriting the properties on the base classes implemented for standard strategies. Flower framework in the background sets up remote procedural call (RPC) server that is a standard network interface used to establish connection between remote servers and client there by inherently aligning with the most of the properties of the federated learning required. The notable strategy that is used commonly and is adopted is fl.server.strategy.FedAvg which is the library implementation of the selected aggregation, Federated Averaging that is in interest of this dissertation. Flower Frameworks gives the flexibility to choose different aggregation algorithm as well. The core implementation is this section is the flower server, since it sends and receives aggregates, and most important it enables the saving the model on a local file system in the event of the server going offline, as well as managing the whole training,communication and orchestration. Additional logic used to implemented is specified as the following. **start_server** is the function that is used to spin up the gRPC server where the parameters mentioned are the number of rounds a server to aggregate the updated models from. It can specify the address on which the server has to be setup. Both client and server on flower framework support certificate authentication and could be enabled if necessary. **fl.server.strategy.FedAvg** like mentioned above orchestrates the aggregation from the client. Additional parameters such as the minimum available of clients which is the minimum number of clients that are required to start

a training, minimum fit client which indicates the number of clients in minimum has to participate in the training and fraction of the available clients that need to be considered can be mentioned while setting up the strategy. There are additional parameters that could be used to create a custom design based on the use case such as on_fit_config_fn which can be used to modify the strategy when the training occurs and on_evaluate_config_fn to determine what should happen on the server side when evaluate function of the client takes place and resultant aggregation happens.

3.4 Data Collection and Simulation

The objective of this dissertation is to study the feasibility of edge federated learning in the healthcare domain. The ideal scenario for data collection would be to obtain data for certain use case from a edge client. i.e. the data from IoT devices or monitoring devices that are used for either research or are deployed in real world. However, such a segregation of data due to its sensitive behavior is not directly available and would require extensive effort to obtain those data. Moreover, the data available usually comes with the similar behavior, mostly on the basis of patients with a single data points representation are mostly obtained rather than a continuous stream of monitoring system data that reports the attributes over the time with a label with each instance of edge client. Another possibility for the data collection would be in the scenarios where monitoring devices collecting the data on a edge client with labels from experts that could be trained on the device and then updated to an edge server that is responsible for aggregation in a area, for example in a county as an edge network, where the models are trained and from there updated models can be aggregated on to a country level which would be the global server. However none of the data provides an identifier to enable such a split in data to be able to simulate such a real world example. Hence a data that is directly available has to be processed to form a real world simulation for this study.

The initial data selected is the diabetes[3] are directly available from Kaggle. This dataset is directly available in a structured format and hence only minor preprocessing are required for the data. However simulation of the data to fit into the study is the real challenge. The data consists of 101,766 data points that consists of diabetes data from different patients that is obtained over a time period of 10 years. With preliminary Explanatory data analysis (EDA) it was found that the data consists of information regarding 71,518 patients. The dataset consists of 51 features out of which, a feature can denoted as the label for the data. The attributes are found in the figure 3.2

From the initial EDA it can be observed that there are no missing data present in the obtained data by visualizing as a matrix as observed from the figure 3.3.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 51 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   id                        101766 non-null  int64
 1   encounter_id              101766 non-null  int64
 2   patient_nbr               101766 non-null  int64
 3   race                      101766 non-null  object
 4   gender                    101766 non-null  object
 5   age                       101766 non-null  object
 6   weight                    101766 non-null  object
 7   admission_type_id         101766 non-null  int64
 8   discharge_disposition_id  101766 non-null  int64
 9   admission_source_id       101766 non-null  int64
 10  time_in_hospital          101766 non-null  int64
 11  payer_code                101766 non-null  object
 12  medical_specialty         101766 non-null  object
 13  num_lab_procedures        101766 non-null  int64
 14  num_procedures            101766 non-null  int64
 15  num_medications           101766 non-null  int64
 16  number_outpatient         101766 non-null  int64
 17  number_emergency          101766 non-null  int64
 18  number_inpatient          101766 non-null  int64
 19  diag_1                    101766 non-null  object
 20  diag_2                    101766 non-null  object
 21  diag_3                    101766 non-null  object
 22  number_diagnoses          101766 non-null  int64
 23  max_glu_serum             101766 non-null  object
 24  A1Cresult                 101766 non-null  object
 25  metformin                 101766 non-null  object
 26  repaglinide               101766 non-null  object
 27  nateglinide               101766 non-null  object
 28  chlorpropamide            101766 non-null  object
 29  glimepiride               101766 non-null  object
 30  acetohexamide             101766 non-null  object
 30  acetohexamide             101766 non-null  object
 31  glipizide                 101766 non-null  object
 32  glyburide                 101766 non-null  object
 33  tolbutamide               101766 non-null  object
 34  pioglitazone              101766 non-null  object
 35  rosiglitazone             101766 non-null  object
 36  acarbose                  101766 non-null  object
 37  miglitol                  101766 non-null  object
 38  troglitazone              101766 non-null  object
 39  tolazamide                101766 non-null  object
 40  examide                   101766 non-null  object
 41  citoglipton               101766 non-null  object
 42  insulin                   101766 non-null  object
 43  glyburide.metformin       101766 non-null  object
 44  glipizide.metformin       101766 non-null  object
 45  glimepiride.pioglitazone  101766 non-null  object
 46  metformin.rosiglitazone   101766 non-null  object
 47  metformin.pioglitazone    101766 non-null  object
 48  change                    101766 non-null  object
 49  diabetesMed               101766 non-null  object
 50  readmitted                101766 non-null  object
dtypes: int64(14), object(37)
memory usage: 39.6+ MB
```

Figure 3.2: The attributes in the diabetes dataset

However this is not a real world representation of the data and hence some of the data has to be deliberately deleted to simulate the real world scenario. A few of the attributes are deleted from the dataset such as id,encounter_id,admission_type_id,discharge_disposition_id, admission_source_id,payer_code. Since these are unique identifiers and does not contribute the prediction labels it has been adopted as a initial cleaning step for the data. Consequently the age of a patient is represented as a range in the dataset and hence it changed to the mean value of the range of data and is replaced as an integer type data. Then using the label encoder from sklearn[20] categorical values in the given attributes are converted into integer format between the range of 0 and the total number of classes $num\_classes - 1$ so that the it can be used as a numerical measure in training the classifier.

Before deleting the value to simulate for the study initial the data is split into train and test sets of the data. Since the data has to be spread across with the nature of the study, the conventional 80-20 split of the data wouldn't make sense since if we happen to split the data in that fashion the number of testing data-points in a client or an edge server would exceed the number of training data-points which is not ideal. Hence the split done on the data in this dissertation is a 95-5 split. Once the split is done the test data is saved separately.The training data is then further processed to add missing values to imitate the real world data. This is done so, by first determining the number of iterations which is calculated as length of total number of columns multiplied by a random float value between the range 0 and 0.5 which is then typecasted to the nearest integer value.

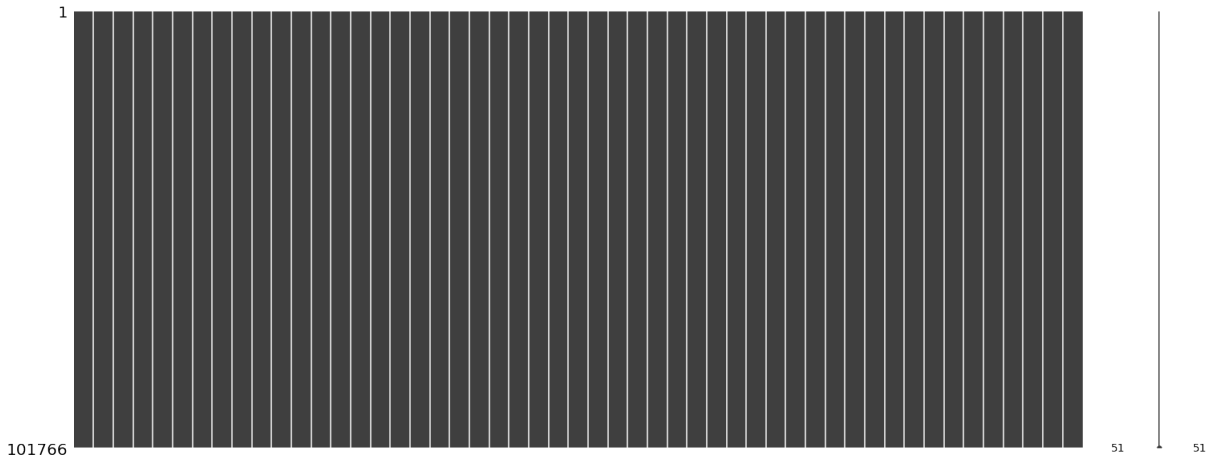


Figure 3.3: Missing value representation of the initial data obtained

The function selects a random column name and deletes 10% of the data from the selected column and this process is repeated until the total number of iterations. The result can be visualized as matrix in the figure 3.5. The code for the same can be found in the figure 3.4

```
def data_deletion(df):
    iterations = int(len(df.columns)*random.uniform(0,0.5))
    columns_df = df.columns.tolist()
    columns_df.remove('patient_nbr')
    for _ in range(iterations+1):
        i = random.choice(columns_df)
        sampled_index = df.sample(frac=0.1).index
        df.loc[sampled_index,i] = None
    return df
```

Figure 3.4: Python function to randomly add missing values to a pandas dataframe

In addition the data that has missing values are now randomly sliced into a multiple chunks of data.i.e. train dataset the training set has to be divided between multiple number of clients that participate in the training. In this dissertation the aim to study is to use 4 edge clients in total, 2 edge servers and 1 global server where, 2 edge servers will have 2 edge clients each. Edge Federated Learning/ Federated learning studies are to be conducted with a non-iid data feed as discussed in the chapter 2.Hence unequal number of data points have to be split between different clients. As an initial step the train dataset is then split to larger number of chunks which are combined into unequal number of datasets which are then used by each clients. This process is repeated whenever a round of clients are to be chosen by the edge servers to replicate the random nature,

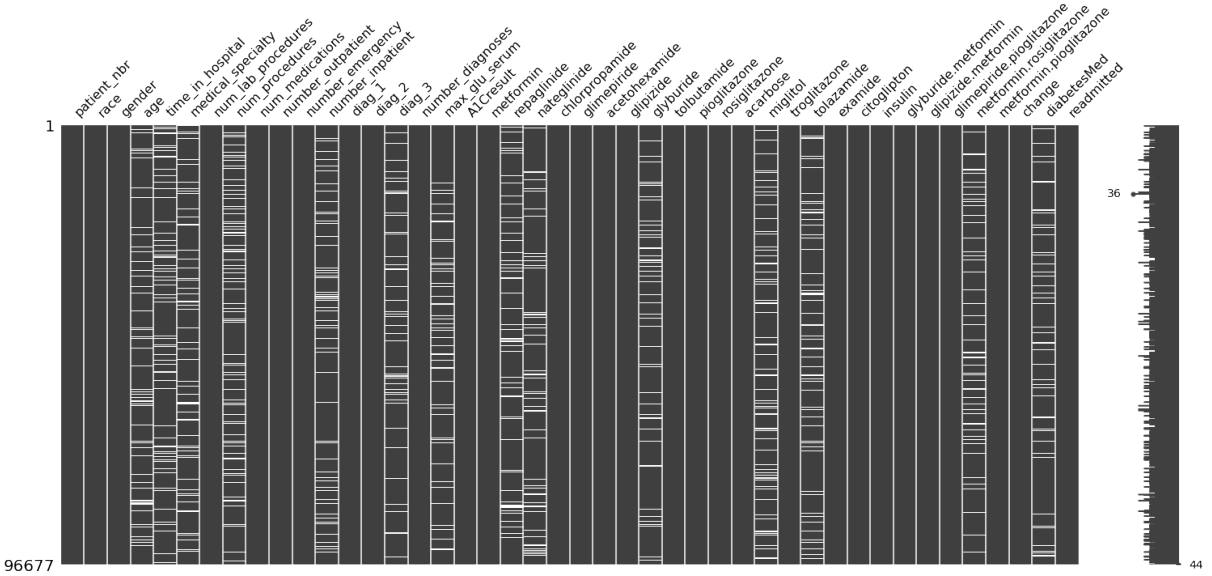


Figure 3.5: Data Visualization after randomly deleting from the dataframe

different client will receive different sized with different set of data. The missing values introduced with client is a closer representation of generation of real world behaviour of the data.

3.4.1 Multivariate Imputation by chained equations

There are three types of missing data[8], Missing completely at Random (MCAR) when the data is missing , Missing at random (MAR), Missing not at random (MNAR). The data has missing data completely at random when the probability of the occurrence of missing data has no influence on the observed data.When the data is missing at random when the data is missing, based on the systematic nature of the data collection. The data is labelled missing not completely at random when the missing data can influence the observed results especially when the study is a targeted for a specific purpose and when the target value as such is found to be missing it called MNAR .

Multivariate Imputation by chained equation is a method to fill in the missing data. There are basic assumptions considered in the MICE imputer. The basic assumption is that the missing data should be either MCAR or MAR. Here in the dataset since the data is deleted completed at random it is safe to assume that the dataset for each client has data missing completely at random. This data imputation has to be implemented in the client. The number of data points used for training are very small and hence the data points with missing values cannot be deleted. Filling the missing values with mean values or so could not be beneficial either. Since MICE learns from the trend of the whole data

taking in consideration rest of the attributes, and since the data generated has almost the same behaviour when collected from the same source, MICE works well in this scenario. MICE works in steps where initially the points where data is missing it replaces it with a place holder, such as mean, median or mode of the data depending on the nature of the data. Then considering one attribute of interest, assuming that other features does not have any missing data, it considered as label to predict. A regression model, usually linear regression model is used learn from the data present that doesn't have missing values and then predict the missing value. The missing value and the place holder is subtracted and the absolute value is calculated as the loss.Now in the subsequent step the missing value is replaced with imputed value and the same process is repeated. This is then again repeated for all the attributes until the loss calculated is at a desirable level. Then in the final round the imputed value is placed instead of the initial placeholder to fill in the missing data. MICE imputer is introduced into all client while loading the data.

From the dataset,out of 43 features that is left in the dataframe, the label can be chosen as the column insulin which indicates level of insulin that has to be injected to a patient. There are four classes in the column- up,down, steady or no if the drug was not prescribed. There are 24 features mentions about the drugs prescribed and other features talks about the patient characteristics and tests conducted on the patient Missing values are not introduced on the labels and if rest of the attributes has miising values MICE imputer is used in client to fill in values.

3.5 Machine Learning Models Development

There are two models used in this dissertation for the feasibility study, Logistic Regression and Neural Network.

Logistic Regression is implemented using the sklearn[20] library in python. In the implementation the parameter $warm\_start$ has to be set to value $True$ so that in each round of the training it is not set to its initial parameters but will start from the point where model was last updated.

Neural Network model is implemented using the library keras[15] implemented inside the open source library TensorFlow[6]. Keras can be used for easy implementation of the neural network models are widely adopted. The neural network model selected for the study in this dissertation has 4 layers with input layer of shape 42 followed by a dense layer of 84 units with activation function Relu 2, followed by another dense layer of same configuration, followed by a dense layer of units 32 units with same activation function , finally

a 4 output dense layer with softmax as the activation function. softnax is represented as

$$\sigma(\vec{Z}_i) = \frac{e^{Z_i}}{\sum_{j=1}^{K} e^{Z_i}}$$

where in the function Z_i represents the input for the multiclass classifier and Z_j represents the output which returns the probability of each class. The loss calculated is a categorical crossentrophy discussed in the section 2 and the optimizer used is an adam optimizer. Adam optimizer is an updated version of SGD discussed in the 2. Adam optimizer is a combination of RMSProp optimizer as well as heavyball optimizer. Heavyball optimizer uses momentum or weighted average for convergence. When there are like higher oscillation in the optimizer due to the random behaviour in the SGD adding momentum reduces the oscillation. RMSProp which is a different version of Adagrad updates the learning rate. Adagrad updates by adding a denominator to the learning rate by adding the gradient. Hence as the epochs progresses the convergence speed reduces. RMSProp introduces a factor β and is added along with the summation of gradients as 1 - β that results in reduced summation of gradient at each step increasing the speed of convergence when compared to Adagrad optimizer. When combined Adam has 3 paramters β_1 for the RMSProp β_2 for the heavyball and α as the intial learning rate. Keras has set the default values and is not tweaked in this dissertation.

3.6 Implementation with Flower Framework

Experiments of the proposed study was conducted on a cloud instance with a hardware access to 30 cores on Intel(R) Xeon(R) Platinum 8170 CPU with 200 GB of RAM. Programming language used was Python 3.9. The environment used to do the experiment was a Jupyter Lab hosted on the above said instance. With the discussion of the development modules in flower framework and the algorithm explained in the edge federated learning section in this chapter the architecture can be designed as two separate federated learning setup. In the first layer global server acting as a flower server is the server side and the edge server requires a flower client to finish the top layer of federated learning using flower framework. The bottom layer federated learning is implemented using flower framework that consists of a server component which is basically another flower server instance in the edge server and the edge client as the flower client. The following architecture is illustrated in the figure 3.1

The models implemented in the server and clients for **neural network** is :

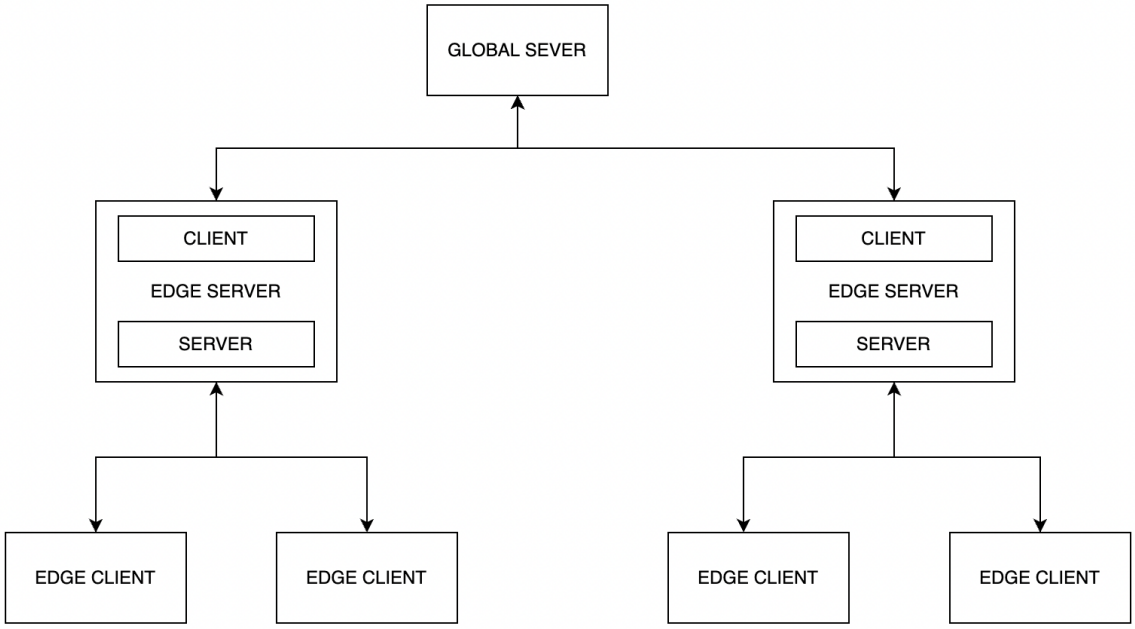


Figure 3.6: Basic Architecture Design

```
model = Sequential()
model.add(Dense(84, input_shape=(42,), activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(4, activation='softmax'))

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

The implementation using flower framework has been added into the appendix section. For edge client A.1, edge server 3.7 and global server A.2

The cloud infrastructure in the background is a Linux based system and hence the orchestration is managed using a bash script. Initially the bash script spins up the global server and pushes the process into the background. To avoid any issues such the client side of the edge server unable to establish a connection with global server, the global server is given around additional 10s to spin up. The after the said time the edge server scripts are initiated and are separately and are again initiated as a set of background processes. The process ID of the edge servers are saved in an array and in a while loop the client servers are spun up. Each client are spun up and are sent to the background, in the meanwhile the the process ID of the client is saved in a separate array. The client scripts wait till

the whole round for clients are complete. The clients are attempted to spun till the edge servers are online and are available for training. This is the case until the global servers are connected to the edge servers. The global server keep the edge server up until all the rounds are completed. Hence the only the check has to be done on to the client side.

3.7 Challenges

Various challenges faced during the designing and implementation are the following

- The selection of the dataset was one of the main challenges. Bigger and better datasets such as MIMIC-III dataset are not directly available. The approval process of the dataset is time consuming. Large number of data points can bring in better results in case more number of clients needs to be included in the study.
- Implementation of the whole setup in a local system was not feasible since the entire architecture can be computationally expensive especially when the model sizes and the trainable parameter size are large. 16 GB RAM in a typical local system might not be enough for the study during the given timeline for the dissertation.
- The base paradigm of federated learning was introduced in 2016 and the edge computing paradigm are all comparatively new. The framework that supports the implementation of federated learning and edge computing are available the documentation and support is not extensive, and there are multiple bugs within the libraries. A framework to implement edge federated learning is not directly available. The library versions are still improving along with issues raised. The flower framework doesn't support the saving of models directly.
- Since most the frameworks are not evolved properly it was time consuming to choose from the available frameworks. Edge federated learning needs additional tweaking of the framework and choosing the framework trying out different modules was time consuming.
- The chosen flower framework does not support the implementation of all the models. Randomforset was one of the models that was intended for the study. However it remained as a challenge due to the inability to implement using the framework.
- Implementing the TensorFlow on Apple devices using M1 or 32 bit chip set is a real challenge. This could be a real problem when these could be implemented in real world devices.

3.8 Summary

This chapter discussed about design and implementation of the edge federated learning using the flower framework. Initially the chapter discussed in detail the algorithm design of edge federated learning. Consequently , it discusses about the flower framework modules that is required for the implementation of the architecture. It further discussed about the models used to train in edge federated learning. Finally the chapter lays out in detail the implementation using flower framework and the bash script to orchestrate the entire architecture. The following chapter will discuss more of the evaluation methods and the results obtained and its comparison with the existing paradigms.

```python
class SaveModelStrategy(fl.server.strategy.FedAvg):
    def aggregate_fit(
        self,
        rnd: int,
        results: List[Tuple[fl.server.client_proxy.ClientProxy, fl.common.FitRes]],
        failures: List[BaseException],
    ) -> Optional[fl.common.Weights]:
        global x_train,y_train,res
        aggregated_weights = super().aggregate_fit(rnd, results, failures)
        if aggregated_weights is not None:
            # Save aggregated_weights
            res = results
            x_train = np.concatenate([np.frombuffer(i[1].metrics['train_features'],dtype=np.float).reshape(i[1].num_ex
            y_train = np.concatenate([np.frombuffer(i[1].metrics['train_label'],dtype=np.float32).reshape(i[1].num_exa
            print(f"Saving round {rnd} aggregated_weights...")
            parameter,_ = aggregated_weights
            weights: List[np.ndarray] = fl.common.parameters_to_weights(parameter)
            model.set_weights(weights)
            model.save('edge_server_model')
#             np.savez(f"round-{rnd}-weights.npz", *aggregated_weights)
        return aggregated_weights
```

```python
'''
strategy initialization
'''
# Create strategy and run server
strategy = SaveModelStrategy(
    # (same arguments as FedAvg here)
)
```

```python
'''
client of the flower network
'''
# Define Flower client
class CifarClient(fl.client.NumPyClient):
  def get_parameters(self):
    return model.get_weights()

  def fit(self, parameters, config):
    global x_train,y_train
    fl.server.start_server('localhost:8901',config={"num_rounds": 1},strategy = strategy)
    model.set_weights(parameters)
    model.fit(x_train, y_train, epochs=1, batch_size=8)
    return model.get_weights(), len(x_train), {}

  def evaluate(self, parameters, config):
    model.set_weights(parameters)
    loss, accuracy = model.evaluate(x_test, y_test)
    return loss, len(x_test), {"accuracy": accuracy}
```

```python
'''
starting the client network
'''
# Start Flower client
fl.client.start_numpy_client("localhost:8800", client=CifarClient())
```

Figure 3.7: Python Implementation of edge server on flower framework]

Chapter 4

Evaluation and Result

This chapter discusses about the evaluation methodology and the results of the edge federated learning architecture implemented in the previous design and implementation chapter. Initially the chapter discusses about the experimental setup used in the dissertation, consequently the evaluation the results obtained and the critical analysis.

4.1 Experimental setup

The data manipulation and the imputation discussed in the previous design and implementation chapter remains the same for all the servers and clients. The neural network architecture and the logistic regression parameter has to be the same for the federated learning and its corresponding communication and aggregation to work. The changes however are brought in with the unequal partition and the data deletion with the aim to retain the random behaviour of the data nature that could be observed in a real world scenario. To explain things in the perspective of the control sequence of a machine learning project, the data obtained are partitioned as train and test data. In total the data points are 101,766 data points out of which 96677 data points are considered as the training set and the rest are considered as the test dataset . There are no data manipulation and imputation introduced to the test data.
The experimental setup consists of three parts.The same test data has been used with the edge clients, edge servers and central server for evaluation.There are 7 components in the experiment setup. 4 edge clients, 2 edge servers and one central server script that is implemented. The central server is initialised first, and then the edge server are initialized establishing the connection between the central server and the edge server. Now the central server expects the edge server to return a updated model as that is the top layer of the edge federated layer.The edge server searches for the edge clients which is the

next layer of edge federated learning. The edge server passes the same model send by the global server on to the edge client now expecting the edge client to update the model. The edge client updates the model as a serialized numpy array and the data as a separate numpy array. The model numpy array is aggregated using the federated average in the edge server,and the data numpy array is concatenated and the aggregated model is again trained using the collected data and the same is aggregated to the central server.The flower framework evaluates using the test set that was separate from the initial dataset. The central server and edge servers saves the model which is implemented using a custom strategy for further evaluation.

The second part of experiment setup is a federated learning setup where there a central server and it waits for four clients to connect for the training. The central server is a normal flower server and the clients are normal flower servers. By default flower server looks for at least 2 clients to connect for the training to take place in a federated setup. Like discussed in the previous chapter fed.server.strategy.FedAvg strategy has to be modified to obtain connection and to wait till 4 clients are connected for the training to start. The model is further saved by the global server for further studies.Even here the data is split is unequally and the missing data are introduced in the same as that in the case of edge federated learning as discussed above.

The final approach would be to use a normal machine learning setup where models are trained centrally. All the data are directly available and there are no unequal partition required however missing data are being introduced and the missing data is filled using the same data imputer as that in edge federated learning and federated learning setup, MICE.

4.2 Evaluation of the Architecture

Two models were studied as a part of evaluation of performance of the training using a edge federated learning, Logistic Regression and Neural Network Model. Edge Federated learning was simulated using bash scripts in a Linux based cloud instance. The data like mentioned in the above section are partitioned unequally to portray the non-iid behaviour behaviour the data in real world scenario. The neural network model consists of 4 layer with input layer size of 42,and the rest as dense layers with size 84,84,32 and 4. The last layer is the output layer and has activation functions Relu for the first three and softmax for the output layer to output the probability of the multiple classes in this case 4 classes. Logistic regression implemented using sklearn has parameters mentioned warm_start = True to avoid re-initialization of the model with each round of train. The maximum iteration count of the logistic regression compared to the default has been increased to

1000 in all the simulated device due to incomplete training warning received from the library.

Although the number of data points changes with each run in a client, a for a typical run in the current setup, an unequal set for partition can be found in the table 4.1

Client	Number of data points
Edge Client 1	1520
Edge Client 2	6444
Edge Client 3	4871
Edge Client 4	5815

Table 4.1: Number of data points distributed among clients

4.3 Performance of the evaluation

The study in total consists of 3 techniques with two models of studying and the time taken for the execution hugely varies. The time in seconds can be found in the table 4.2. The time recorded was for 10 epochs in each architecture in each model. It can be seen that from the simulation the edge federated learning is time consuming compared to other two architecture. Centrally trained model in both cases takes the least amount of time as expected. With federated learning time drastically increased since there is overhead for the communication between two entities now and the aggregation turns out to be an extra step. Further it also depends on the data transfer latency which in this case could be eligible since it was essentially trained on the same system. The edge federated learning although is good at offloading task it takes considerable amount of time to train the model in the same setting due to the presence of large number of clients and servers involved and it will be interesting to study the performance of edge federated learning as a separate topic given right resources. As expected the neural network training takes significantly more time compared to that of the linear model. In the edge federated learning this increased by a very large proportion. One of the work around would be to reduce the number of training rounds in each step. i.e. training on the edge client device can avoided but it will miss out of the further advantages such as personalising of the model in the device.

Another factor to discuss in relation with the performance of the architecture is global communication of the model. The point to note is the number of global communications are very less compared to other architectures.In a centrally trained model the global communication happens between the client and the server directly for the transfer of data. In federated learning the clients directly upload the model to the central server. In

Model	Training Method	Time taken for training
Logistic Regression	Centrally trained	128 s
	Federated Learning	1268.3s
	Edge Fed	3600.3s
Neural Network	Centrally trained	993 s
	Federated Learning	4492.11s
	Edge Fed	58683.7s

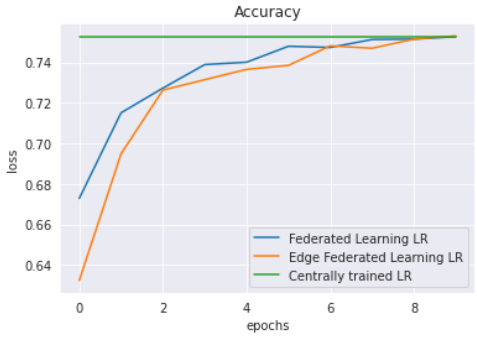
Table 4.2: Execution time taken to train models in different frameworks

edge federated learning, only the edge server communicates with the global server. It can derived that centrally trained architecture will have large number of global communication happening compared to federated learning since it uploads the data itself on to the cloud whereas FL only uploads the updated weights. Additionally the edge federated learning has less number of global communication since only updated weights from edge servers are sent, which will drastically reduce the global communication overhead. The evaluation of data transfer however,is beyond the scope of this dissertation.

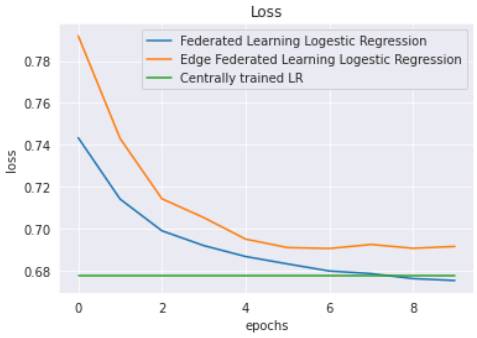
4.4 Evaluation of the Models trained

The model trained are for a classification problem based on a health care dataset. The initial parameters to check would be to quality of the training process from the accuracy gained over multiple epochs and the reduction in loss. The centrally trained logistic regression gets trained in a single step and the implementation library does not extend direct results regarding the training of the model in each step. Hence the loss and accuracy cannot be accounted for over the epochs like in other cases of the training strategy. The loss and the accuracy of the models trained are plotted in the figure 4.1

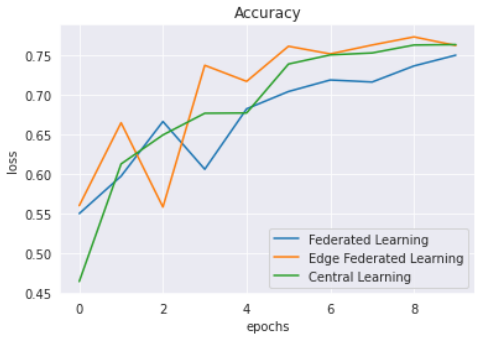
On critically analysing the loss and accuracy of the models trained the linear model did not observe higher fluctuations , it was comparatively smoother when compared to the loss function. this could be because of the less number of trainable parameters when compared to neural network. Neural Network with large number of parameters when trained using mini batch SGD there tends to be a large oscillations in the learning of the model. This could be reduced by the usage of Adam optimizer. The reason for increase in oscillation could be with the aggregation using federated averaging, where an optimizer with momentum is missing to reduce oscillation, as a result could be the probable cause for oscillation when large of training parameters are involved . This behaviour requires further studying with variable number of epochs to understand the behaviour especially if the oscillation would die out with further number of epochs.



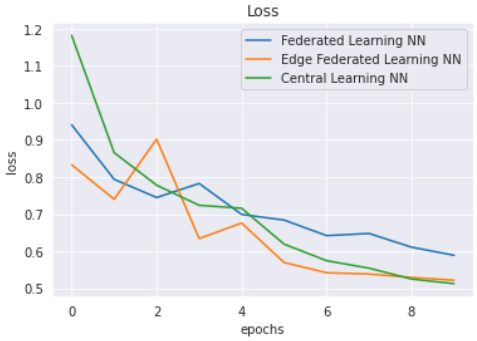
(a) Accuracy of Logistic Regression



(b) Loss of Logistic Regression



(c) Accuracy of Neural Network



(d) Loss of Neural Network

Figure 4.1: Loss and Accuracy of the models trained over multiple architectures

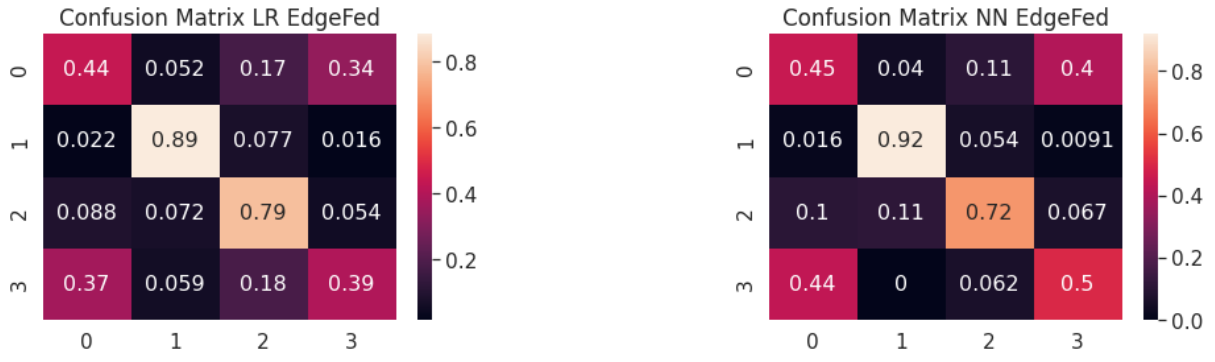
The model accuracy for both cases are around the value 75% with logistic regression less than 75% and neural network accuracy with more than 75% which is an expected behaviour. Neural Network is expected to have better accuracy with further number of epochs and it was observed with the basic experiments during the dissertation. Logistic Regression on the other hand with further progress would not have better accuracy with larger number of rounds.

Another parameter in machine learning especially classification are other evaluations such as Precision, recall and F1 score. There could be large imbalance in the data with label distributed in a different quantities. These parameters are calculated from the confusion matrix which is a representation of the actual observed values and the predicted values from the model. It indicates the the overlap between the actual value and the predicted value by the parameters such as True Positives, True Negatives, False Negatives and False Positives. Precision is a fraction of true positives by the total all the predicted true values. Recall is the fraction of true positives by the total number of actual positive values. F1 score is the harmonic mean of both precision and recall. The reason why there parameters are extremely important in the healthcare domain is that, the models should

be detect the necessary target. For example, a model predicting cancer should be able to detect all the cases of cancer. This is explained by the parameter, recall. Consequently , in the same example the model detecting cancer should be able to rightfully say if the patient has cancer or not, and if it falsely predicted it can lead to wrong/unwanted treatment for the patients who does not have cancer. Accuracy cannot be considered as real metric since the number of patients who has cancer will be less and even if the model predicts one label the model would still have higher accuracy. The above said results of the models can be found in the table 4.3. In each column and row the values are represented for the label which in indicates whether the dosage has to be decreased, or no prescription is required, or if kept steady or increased in its corresponding order. It can be seen that the reliable predictions are for the classes 'no' and 'steady' rest of the prediction using this model is not useful. However the results are comparable with the centrally trained model and federated learning trained model , and hence the edge federated learning can be used as feasible architecture in real world scenario as an architecture instead of centralised trained machine learning model. The results can be improved with better hyper parameter optimization of the model for better results. The normalised confusion matrix 4.2 A.3

Model	Training Method	Precision	Recall	F1
LR	Centrally trained	0.43,0.90,0.73,0.38	0.45,0.92,0.80,0.21	0.44,0.91,0.76,0.27
	Federated Learning	0.44,0.90,0.74,0,0.36	0.49,0.91,0.77,0.22	0.46,0.91,0.77.0.25
	Edge Fed	0.42,0.90,0.73,0.39	0.43,0.92,0.79,0.27	0.42,0.91,0.72,0.32
NN	Centrally trained	0.30,0.92,0.69,0.42	0.06,0.87,0.86,0.63	0.09.0.89,0.77,0.51
	Federated Learning	0.45,0.92,0.71,0.46	0.61,0.87,0.85,0.11	0.52,0.89,0.77,0.17
	Edge Fed	0.45,0.92,0.72,0.40	0.50,0.90,0.85,0.02	0.53,0.91,0.78,0.03

Table 4.3: Results of Models trained

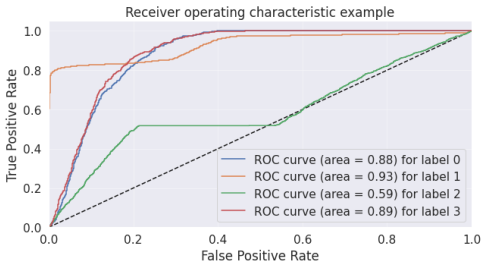


(a) Confusion Matrix of Logistic Regression EdgeFed

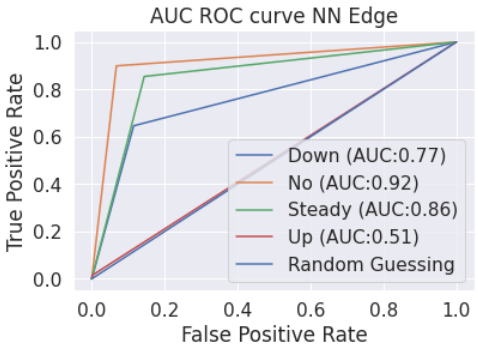
(b) Confusion Matrix of Neural Network EdgeFed

Figure 4.2: Confusion matrix of models

Another factor that is considered to understand the classification label is a AUC-ROC curve which indicates the quality of prediction of the class. Larger the area under the graph better the prediction of the class and again it is an indicator of classification quality. The plots are illustrated in figures. 4.3 A.4. It can seen that in comparison with teh confusion matrix matrices 'no' and 'steady' are the only reliable prediction from the current trained model.



(a) AUC ROC curve of Logistic Regression EdgeFed



(b) AUC ROC curve of Neural Network EdgeFed

Figure 4.3: Confusion matrix of models

4.5 Summary

This chapter discusses about the evaluation and results of the models trained using edge federated learning, federated learning, and centrally trained model and it can be found that the edge federated learning has comparable results with other two techniques with an accuracy of nearly 75%. However, the only two classes are the only reliably predicted classes but that can be improved with better hyper-parameter optimization and better cleaning of data. This study is an different variation of the paper [26] but cannot be compared directly but the dissertation explains using the edge federated learning in healthcare domain and it can seen that it is a feasible architecture since it has comparable results with the existing methods of training.

Chapter 5

Conclusions & Future Work

5.1 Conclusion

The dissertation discussed about the feasibility study of design and implementation of the edge federated learning. There were mainly three goals set with set with the beginning of study. Initially the diabetes dataset obtained from Kaggle with no missing points where introduced with random missing values by selecting random columns and random rows from the dataframe. The train part of the dataset with missing values were partitioned into unequal number of chunks of data and then unequally distributed with the client to portray the real world data distribution just the data distributed on the edge clients(non-iid behaviour). Using the data, a logistic regression model and neural network classifier was trained using the designed edge federated learning, federated learning , and the centrally trained where models are trained on a cloud. The whole setup was implemented on a Linux based cloud instance with 30 cores of Intel Xeon Platinum CPU with 200 GB of RAM . The whole process was simulated using a bash script in Linux and was able to train the model and record its accuracy, loss, performance in each architecture. It was found that there was considerable time taken in extra for the training of the neural network models using the edge federated setup. Logistic Regression has an accuracy over 74% and the neural network models has an accuracy of more than 76%. Finally, the metrics were compared in three architectures, and it was found that the results where comparable between the edge federated learning, federated learning and central training of models. Hence it could be concluded that Edge Federated Learning is a feasible approach in the healthcare domain and certainly is one of the key elements in the future of digital healthcare.

5.2 Future Work

The research in dissertation can be extended to multiple arenas. For the future, a few suggestions to better cover the shortcomings of this study are to simulate the study using better dataset with larger number of data points. Datasets such as MIMIC-III requires approval in prior to access the data. Better understanding and a presence of an identifier in the data should aid in better partitioning of the data to better simulate a non-iid behaviour. Consequently, one of the interesting study would be to learn the behaviour of tree based models like RandomForest, decision trees in edge federated learning. Another essential factor to further progress in the study of edge federated learning is its performance. There is a considerable amount of time required for the training of the model especially with neural network. When large of clients take part and large amount of data will be involved in the training this could exponentially increase the amount of time taken for a round of training although number of global communication is really less. There could be different optimizations that could be introduced to edge federated learning such as enabling training only in the edge server rather than edge client in addition, in an edge network. This study makes use of both edge server and edge client to take part in the training process which could be one of the reasons for increased time. One of the scopes for future study is evaluating the number of global communication involved in the architecture.

Bibliography

[1] Federated learning - tensorflow federated. URL https://www.tensorflow.org/federated/federated_learning.

[2] Copyright. In R. Buyya, R. N. Calheiros, and A. V. Dastjerdi, editors, *Big Data*, page iv. Morgan Kaufmann, 2016. ISBN 978-0-12-805394-2. doi: https://doi.org/10.1016/B978-0-12-805394-2.09993-1. URL https://www.sciencedirect.com/science/article/pii/B9780128053942099931.

[3] 2022. URL https://www.kaggle.com/datasets/jimschacko/10-years-diabetes-dataset.

[4] 5g and edge computing, 2022. URL https://www.ibm.com/cloud/smartpapers/5g-edge-computing/.

[5] 5g and edge computing: Why does 5g need edge?, Feb 2022. URL https://stlpartners.com/articles/edge-computing/5g-edge-computing/.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[7] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane. Flower: A friendly federated learning research framework, 2020. URL https://arxiv.org/abs/2007.14390.

[8] M. Bland. *An introduction to medical statistics*. Oxford university press, 2015.

[9] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[10] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[11] L. Breiman and A. Cutler. Random forests for scientific discovery. *línea]. Available: https://www. stat. berkeley. edu/˜ breiman/RandomForests/berkeley_files/frame. htm*, 2016.

[12] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[13] K. Cao, Y. Liu, G. Meng, and Q. Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020. doi: 10.1109/ACCESS.2020.2991734.

[14] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd, 2016. URL `https://arxiv.org/abs/1604.00981`.

[15] F. Chollet et al. Keras. `https://keras.io`, 2015.

[16] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020. doi: 10.1109/MSP.2020.2975749.

[17] B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data, 2022. URL `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`.

[18] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-efficient learning of deep networks from decentralized data. 2016. doi: 10.48550/ARXIV.1602.05629. URL `https://arxiv.org/abs/1602.05629`.

[19] B. Mehlig. *Machine Learning with Neural Networks*. Cambridge University Press, oct 2021. doi: 10.1017/9781108860604. URL `https://doi.org/10.1017%2F9781108860604`.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] S. Published by Statista Research Department and M. 23. Total data volume worldwide 2010-2025, May 2022. URL https://www.statista.com/statistics/871513/worldwide-data-created/.

[22] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[23] Wikipedia contributors. Multinomial logistic regression — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Multinomial_logistic_regression&oldid=1098653133. [Online; accessed 12-August-2022].

[24] Wikipedia contributors. Stochastic gradient descent — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=1098148439. [Online; accessed 8-August-2022].

[25] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi. Privacy preserving vertical federated learning for tree-based models. *arXiv preprint arXiv:2008.06170*, 2020.

[26] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu. Edgefed: Optimized federated learning based on edge computing. *IEEE Access*, 8:209191–209198, 2020. doi: 10.1109/ACCESS.2020.3038287.

Appendix A

Figures

```
'''
edge client
'''
# Define Flower client
class CifarClient(fl.client.NumPyClient):
  def get_parameters(self):
    return model.get_weights()

  def fit(self, parameters, config):
    model.set_weights(parameters)
    model.fit(x_train, y_train, epochs=1, batch_size=8)
    return model.get_weights(), len(x_train), {'train_features':x_train.tobytes(),'train_label':y_train.tobytes(),'tr

  def evaluate(self, parameters, config):
    model.set_weights(parameters)
    loss, accuracy = model.evaluate(x_test, y_test)
    return loss, len(x_test), {"accuracy": accuracy}
```

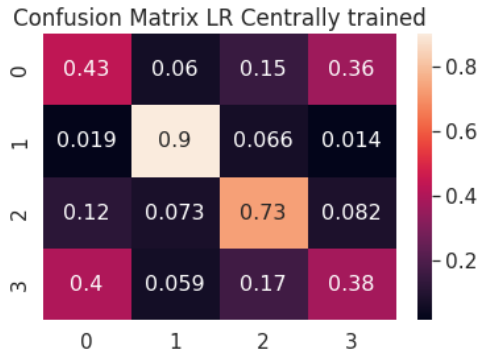
Figure A.1: Python Implementation of edge client on flower framework]

```python
class SaveModelStrategy(fl.server.strategy.FedAvg):
    def aggregate_fit(
        self,
        rnd: int,
        results: List[Tuple[fl.server.client_proxy.ClientProxy, fl.common.FitRes]],
        failures: List[BaseException],
    ) -> Optional[fl.common.Weights]:
        aggregated_weights = super().aggregate_fit(rnd, results, failures)
        if aggregated_weights is not None:
            # Save aggregated_weights
            print(f"Saving round {rnd} aggregated_weights...")
            parameter,_ = aggregated_weights
            weights: List[np.ndarray] = fl.common.parameters_to_weights(parameter)
            model.set_weights(weights)
            loss, accuracy = model.evaluate(x_test, y_test)
            with open('./loss.txt','a+') as f:
                f.write(str(loss)+'\n')
            with open('./accuracy.txt','a+') as f:
                f.write(str(accuracy)+'\n')
            model.save('global_server_model')
            #np.savez(f"round-{rnd}-weights.npz", *aggregated_weights)
        return aggregated_weights
```

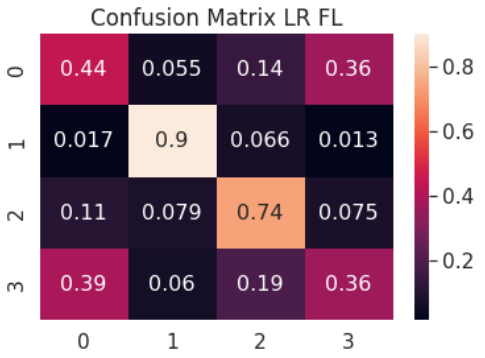
```python
'''
strategy initialization
'''
# Create strategy and run server
strategy = SaveModelStrategy(
    # (same arguments as FedAvg here)
)
```

```python
# Start Flower server
fl.server.start_server(
  "localhost:8800",
  config={"num_rounds": 10},
  strategy = strategy
)
```

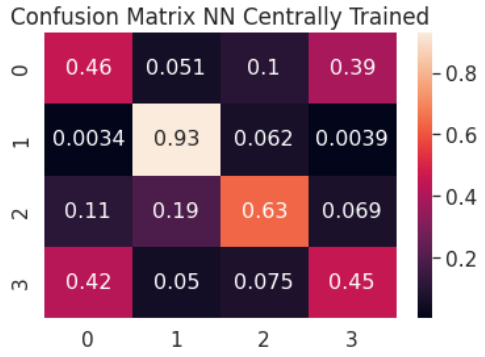
Figure A.2: Python Implementation of global server on flower framework]



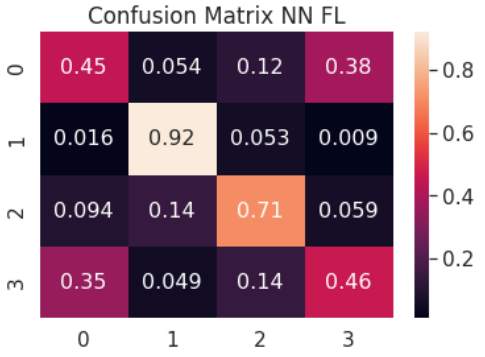
(a) Confusion Matrix of Logistic Regression centrally trained



(b) Confusion Matrix of Logistic Regression FL

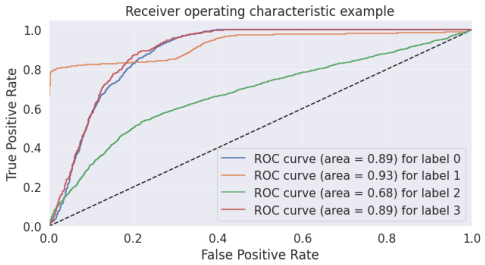


(c) Confusion Matrix of Neural Network centrally trained

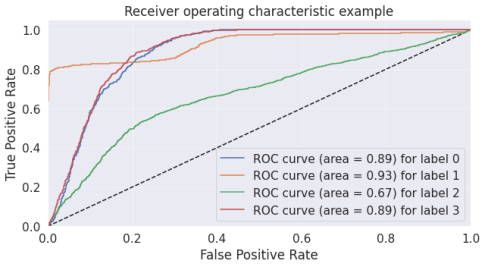


(d) Confusion Matrix of Neural Network FL

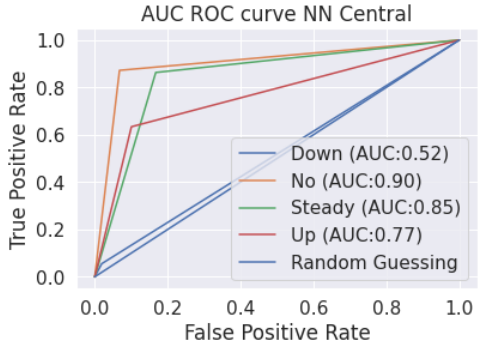
Figure A.3: Confusion matrix of models



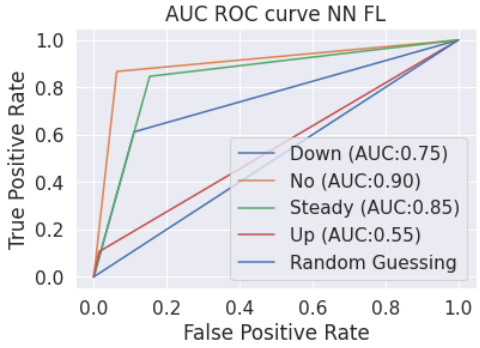
(a) AUC ROC Curve of Logistic Regression centrally trained



(b) AUC ROC Curve of Logistic Regression FL



(c) AUC ROC Curve of Neural Network centrally trained



(d) AUC ROC of Neural Network FL

Figure A.4: Confusion matrix of models