Abstract

This paper examines the role of the Pythia federated network oracle. At that time, the interaction between applications and network is very limited. This problem might be resolved with the Pythia oracle. Users can ask Pythia, a distributed oracle, queries about network issues including packet delays and bandwidth.

This study aims to achieve queries regarding future performance. To accomplish this goal, Pythia should absorb passive monitored data that is shared in a private manner. Federated learning is employed to analyse and learn from this passive data and forecast crucial network information. Finding a mechanism to link the Pythia with other internet nodes is the final step in achieving the desired outcome. Socket programming was applied to achieve this.

The Pythias federated network oracle model's construction process and strategy are also illustrated in this study. Furthermore, it provides additional information about the difficulties that emerged and potential solutions.

Keywords: Federated Learning (FL), Machine Learning (ML), Multi-layer perceptron (MLP), FedAvg, Server, Client, Global model, Socket programming.

Table of Contents

List of Figures

List of Appendices

1. Introduction and Problem Statement

Nowadays, the socket interface is the only way to interact with applications and the network. Since it just allows for reading and writing, this interface is limited. Applications are unable to query network conditions or communicate with the network about what they are transmitting. Pythia is a modern application/network collaboration paradigm that improves the socket interface and adds a new feature to IP packets that allows network querying and information communicated to the network about what will be conveyed.

Pythia is a distributed oracle that as described above helps in the collaboration between application and network. To accomplish this, it uses Federated learning. Federated learning is a type of Machine learning, which allows for training on a vast corpus of decentralized data stored on devices such as mobile phones. Before explaining how federated learning works, a brief explanation of what machine learning is needed.

With the use of machine learning (ML), which is a form of artificial intelligence (AI), software programmes can predict outcomes more accurately without having to be explicitly instructed to do so. [1] In order to forecast new output values, machine learning algorithms use historical data as input. [1] There are several types of ML, most known are Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. The type of machine learning algorithm that will be utilized in a project is defined by the type of data that will be forecasted by the algorithm. Machine Learning helped to transfer human intelligence to computers. However, it faces a lot of challenges and one of them is privacy. One of the novel ways being utilized to tackle this difficulty is federated learning.

Federated learning is a branch of machine learning in which multiple entities (clients) collaborate to solve a machine learning problem while being monitored by a central server or service provider. Each customer's raw data is saved locally and never shared or sent. Instead, updates targeted for immediate aggregation are employed to accomplish the learning requirement. There are three main phases in FL (Federated Learning): Selection, Configuration and Reporting. Each of these phases describes the connection between the client and the server. To connect with devices that adhere to particular FL requirements, the server has a particular architecture. The server consists of coordinators, selectors and aggregators. The selectors that the FL employ to pick the devices are controlled by the coordinators. The aggregator, which is the most crucial component, combines all the models that were trained using the devices to create a new, updated model.

Depending on whether the models are data-centric or model-centric, different forms of FL are employed. The model-centric model is further defined by the following terms: Cross-device FL systems and Cross-Silo FL models. Different FL, such as Vertical or Horizontal FL, will be employed depending on the FL system selected. This report involves additional research into the algorithms that FL employs and the security

aspects that FL has to implement. Moreover, it demonstrates several real-world examples as well as FL's benefits and limitations.

Pythia will include a server that will aggregate the clients' trained models. The clients will be neural networks that will train models by using a specific dataset. This local dataset will contain information about each packet, such as Source IP addresses, Destination IP addresses, Date/Time, delay and bandwidth. Two software packet capture technologies will be used to gather this packet information. These are Tcpdump and Wireshark.

A section of this study is devoted to the methodology and the steps taken to complete the project.

A section on the project's results is also included in this report. The work that went into making the Pythia oracle is described in this section. This part also includes an analytical breakdown of the code architecture and the dataset's creation procedure.

The following figure illustrates an inference box of Pythia's oracle that communicates with other nodes/users.

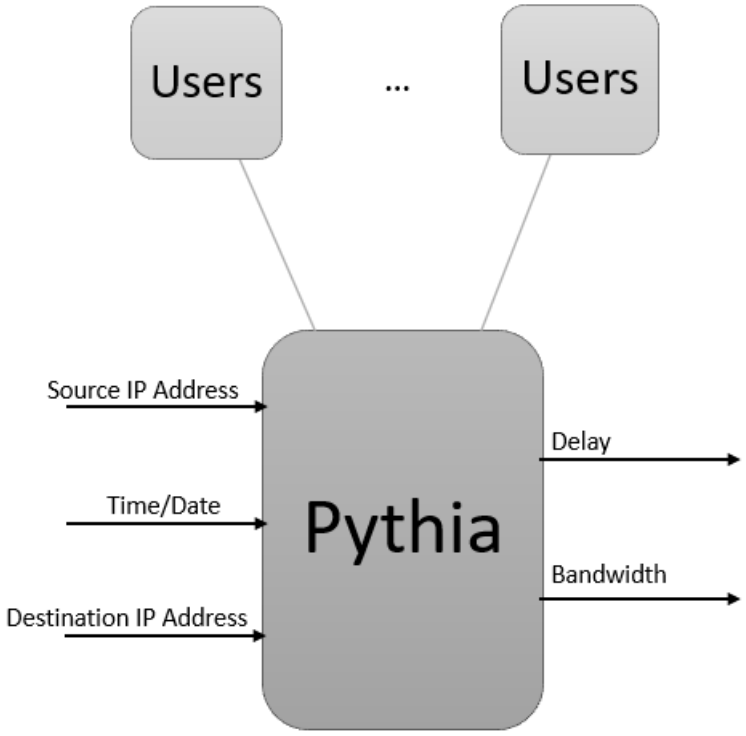

Fig. 1: Pythia's inference box

2. Context, Background and Literature Review

Important terms such as Federated Learning and Machine Learning will be defined in this section.

2.1 Machine Learning (ML)

Machine learning is a division of artificial intelligence (AI) and computer science that focuses on using data and algorithms to mimic how humans learn and improve accuracy over time. [2] Machine learning, as a discipline, investigates the analysis and development of algorithms that can learn from and predict data.

The machine learning process is broken down into three steps:

1. A Decision Process: ML algorithms are typically used to make a prediction or classification. In Machine Learning, classification is a concept that splits a set of data into classes. In this process, the algorithm will generate an estimate of a pattern in the data based on the input data. These data can be unlabelled or labelled.
2. An Error Function: An error function is implemented to evaluate the model's prediction. If there are other known examples in ML, this function can be implemented to compare the model's accuracy.
3. A Model Optimization Process: If the model fits better to the data points in the training set, then weights are modified to lessen the gap between the known example and the model prediction [2]. The algorithm will try to improve the accuracy of the procedure by updating the weights on its own. It will repeat this process until the accuracy reaches a certain level.

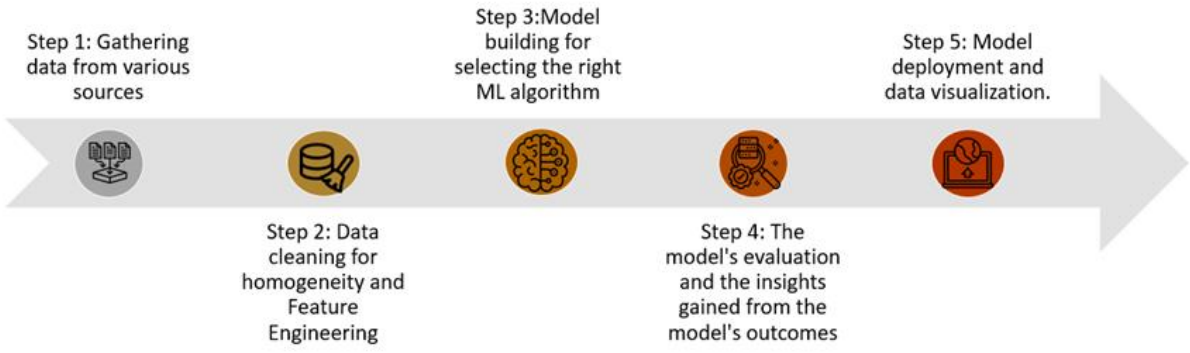

Fig. 2: Machine learning process

Machine Learning is classified according to how an algorithm comprehends and learns how to improve its prediction accuracy. Among several ML methodologies, the most common machine learning approaches are Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. The type of ML

algorithm that will be used in a project is determined by the type of data that the algorithm wants to forecast.

- Supervised Learning (SL): Supervised Learning is a type of ML in which the algorithm learns a model based on a labelled dataset. As a result, the training set or dataset implemented to train the model consists of input and intended output (labels) pairings. Linear and Logistic Regression, Random Forests, Support Vector Machines (SVM) and Neural Networks are some examples of SL techniques.
- Unsupervised Learning: This type of Machine Learning, gives unlabelled data that the algorithm attempts to understand on its own by extracting features and patterns. Clustering and anomaly detection are some methods used in unsupervised learning.
- Semi-supervised learning: This method of machine learning combines the two previous approaches. It guides classification and feature extraction from a larger, unlabelled dataset using a smaller labelled dataset during training. When there isn't enough labelled data to train a supervised learning algorithm, semi-supervised learning could be used to solve the issue.
- Reinforcement learning is a branch of machine learning that is similar to supervised learning but does not rely on sample data to train the algorithm. [2] The most important elements of reinforcement learning are trial and error search and delayed reward. To determine the optimal idea or policy for a given context, a series of successful outcomes will be reinforced.

The following are examples of real-world machine learning applications:

- Speech recognition - Speech recognition is a skill that implements natural language processing (NLP) to translate a person's voice into text. Automatic speech recognition, computer speech recognition and speech-to-text are all terms used to describe this technology.
- Customer service - Online chatbots are increasingly replacing human workers across the customer journey. They give users customized advice, cross-sell products and recommend sizing.
- Computer Vision - Computer vision is an artificial intelligence (AI) approach that enables computers and systems to extract usable information from digital pictures, movies and other visual inputs and act on that knowledge. [2]
- Self-driving cars - Machine learning techniques could let a semi-autonomous vehicle detect a partially visible object and alert the driver.

Machine learning has aided in the transmission of human intelligence to machines. Nevertheless, it is vital to understand what machine learning can and cannot do. Privacy is one of the most significant issues that machine learning faces. Federated learning is one of the innovative strategies being used to overcome this challenge.

2.2 Federated Learning (FL)

Federated learning is a type of machine learning in which numerous entities (clients) work together to solve a machine learning issue under the supervision of a central server or service provider. The raw data of each customer is stored locally and is not shared or transmitted. Instead, to meet the learning task, specific updates designed for immediate aggregation are used. [3]

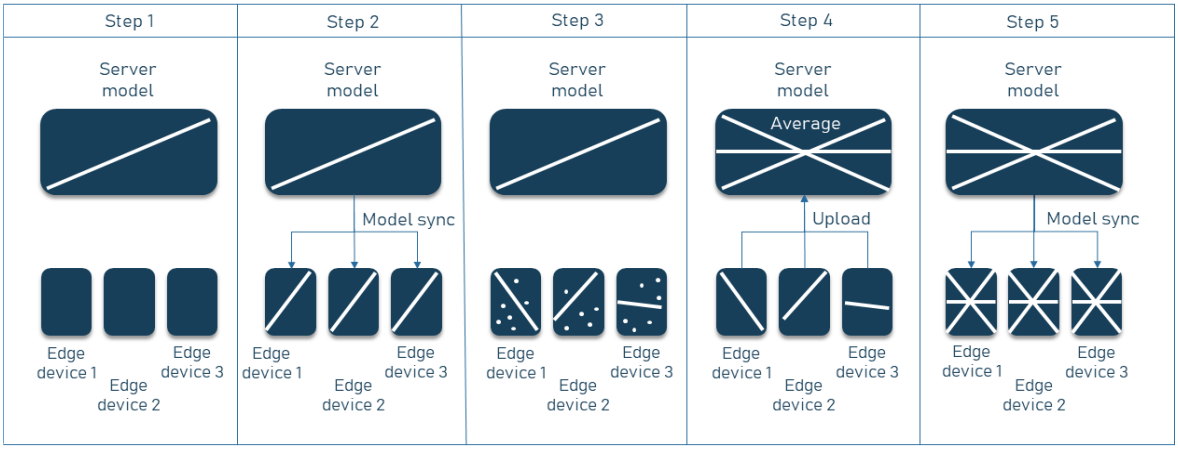

Fig. 3: Federated learning process [4]

Federated learning operates like this: the clients' devices download the current model from the server. These devices improve the model by using their data and then condensing the modifications into a minor update. Then, using encrypted protocols that will be detailed later in this section, this update is sent to the cloud (server). This update is combined with other user updates/changes on the server to improve the shared model. Individual updates are not saved to the cloud, and all training data is stored locally on the user's device.

In FL, there are three main phases of communication between a server and its clients:

1. Selection: Periodically, devices that match the eligibility criteria (e.g., charging and connected to an unmetered network or being idle) establish a bidirectional stream to check in with the server. The stream is used to track whether or not a device is awake, as well as to orchestrate multi-step communication. Based on specific criteria, the server selects a subset of connected devices. The optimal number of participating devices could be one of these factors (usually a few hundred every round). Furthermore, if a device is not chosen for participation, the server responds with instructions to rejoin at a later time. [5]
2. Configuration: The server is customized for the selected devices based on the aggregation mechanism (e.g., basic or Secure Aggregation). The FL plan and an FL checkpoint containing the global model are sent to each device by the server.
3. Reporting: The server waits for updates from the participating devices. As updates arrive, the server uses an optimization technique to aggregate them and tells the reporting devices when to reconnect. If enough devices report in time, the round will be finished successfully and the server's global model will be updated. Otherwise, the round will be abandoned. [5]

As previously said, one of the most important parts of FL is the server. The need to function over many orders of magnitude in population sizes and other dimensions led to the development of the FL server. The server must be able to operate with FL clients ranging in size from tens of devices to hundreds of millions during the selection step and process rounds with tens of devices to tens of thousands of participants. [5] Furthermore, the modifications gathered and provided throughout each round can range in size from a few kilobytes to tens of megabytes. [5] Finally, the amount of traffic entering or leaving a given geographic region may fluctuate dramatically during the day based on when devices are idle or charging.

The FL server's architecture is made up of the following components:

- Coordinators - Coordinators are the top-level entity of the server hierarchy who enable global synchronization and round progression in lockstep. There are several Coordinators, each of whom is in charge of an FL population of devices. A Coordinator registers its address and the FL population it maintains in a shared locking service, ensuring that every FL population has a single owner who can be reached by other elements of the system, particularly the Selectors. The coordinator receives information about how many devices are linked to each Selector and directs them on how many devices to accept for participation based on which FL tasks are planned. Master Aggregators are spawned by Coordinators to oversee the rounds of each FL task. [5]
- Selectors - Accepting and routing device connections are the responsibility of selectors. They receive regular updates from the coordinator about how many devices are needed for each FL population, which they use to make local judgments about whether or not to accept each device. The Coordinator orders the Selectors to send a subset of its connected devices to the Aggregators. When the Master Aggregator and set of Aggregators are generated, the Coordinator can allocate devices to FL tasks efficiently, regardless of the number of devices available. The Selectors can also be spread globally (near devices) to reduce the communication between the remote Coordinator and the devices. [5]
- Master Aggregators - Master Aggregators oversee the rounds of each FL task. They make dynamic choices to spawn one or more Aggregators to whom work is outsourced in order to scale with the number of devices and the number of updates. [5]

The server's architecture is depicted in the figure below.

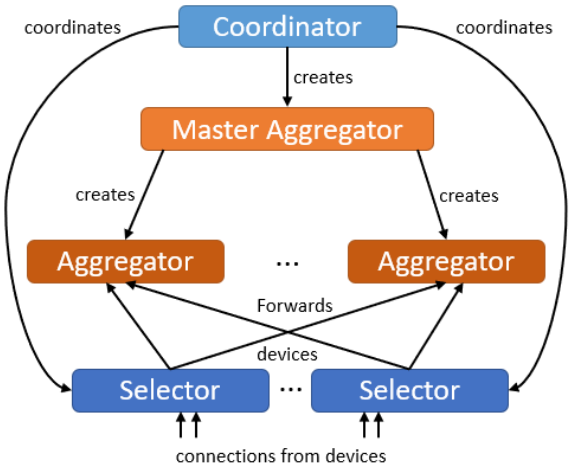

Fig. 4: Federated learning server architecture

While the round's Selection, Configuration, and Reporting stages are all consecutive, the Selection phase does not rely on any prior round's input. Latency can be minimized by running the Selection phase of the next cycle of the protocol in parallel with the Configuration/Reporting stages of the previous round. This system architecture enables such pipelining without adding additional complexity because parallelism is provided simply by the Selector conducting the selection process continuously.

In all failure circumstances, the system will continue to work, either by completing the current round or by beginning from the prior committed round's findings. The loss of a subsystem of the server will not always prevent the round from succeeding. If an Aggregator or Selector crashes, only the devices attached to that system will be lost [5]. If the Master Aggregator fails, the current round of the FL task it oversees will also fail, but the Coordinator will restart it. Finally, the Selector layer will detect and respawn the Coordinator in case the coordinator fails. This will only happen once because the Coordinators are registered in a shared locking service. [5]

The client is another crucial aspect of FL. This system must be capable of client-side training operations as well as coordinating model parameters with the central service. To update the local model, the client system will need to get new parameters from other clients in the network.

To participate in FL, clients must overcome several challenges. The datasets implemented in FL are frequently heterogeneous, ranging in size from a few terabytes to several gigabytes, which can have a significant impact on the performance of the training process. Since federated learning clients often rely on less powerful communication mediums like Wi-Fi and battery-powered systems such as smartphones and IoT devices, they may be unreliable, as they are more prone to make errors or drop out.

2.2.1 Types of FL

Federated Machine Learning can be classified into two main types, Model-Centric and Data-Centric.

In a Model-Centric type of FL, the data is generated on a local level and is not centralized. Each client keeps track of its data and is unable to access the data of other clients. This type deals with non-IID data (non-independent and identical distribution). Furthermore, this type of FL can further qualify with the following terms:

- Cross-Device Federated Learning Systems – These FL systems can have an extensible number of clients. They can be enabled and deactivated at any time. Typically, these are mobile and IoT gadgets. In addition, Cross-device FL systems use Horizontal FL for their datasets. Horizontal FL is when the decentralized datasets have the same feature space but differ in sample space. [6]
- Cross-Silo Federated Learning Systems – These FL systems have a low scalable federation. They contain data centres or organizations and the clients connected to them are modest and fluctuate infrequently. Cross-Silo FL systems use Vertical FL. Vertical FL presents a scenario in which two decentralized datasets share the same sample space but differ in feature space. [6]

The second type of FL is Data-Centric. This is a newer, emergent version of Federated Learning that could be outgrowing the Federated definition, as it has a more peer-to-peer feel to it. The challenge of data-centric FL is the same as model-centric FL, but it is approached from the opposite direction. The most likely scenario for this type of FL is where a person or organization has data they want to protect. This would allow a data scientist who is not the data owner, to make requests for training or inference against that data. [6]

FTL (Federated Transfer Learning) is another type of FL that could develop. This type was introduced to improve statistical models in a data federation by allowing knowledge to be shared without jeopardising user privacy and allowing complementary knowledge to be transported across the network. However, there is not a concrete production implementation or a case study on this type of FL yet. [6]

2.2.2 Enabling Technologies: Algorithms

It is vital to delve deeper into the algorithms employed in federated learning to acquire a better knowledge of how it works. There have been a lot of algorithms developed for FL. These technologies have been implemented to improve FL even more.

FedAvg is one of these FL algorithms. The FedAvg method works by initializing training with the help of the main server, where the clients share a global model. The Stochastic Gradient Descent (SGD) algorithm is used for optimization. The iterative approach of SGD is used to optimize an objective function with sufficient smoothness criteria. Furthermore, the FedAvg algorithm has five parameters that must be taken into consideration: the number of customers, batch sizes, epochs, learning rate, and decay. [7]

The FedAvg algorithm launches by starting up the global model. The server selects a group of clients and sends the most recent model to all of them. Then Clients divide their data into different batch sizes and perform a set number of SGD epochs after changing their local models to the shared model. The clients then provide the server with their freshly changed local models. The server creates new global models by calculating a weighted average of all the local models it has collected. [7]

Another algorithm that FL use is the Federated Stochastic Variance Reduced Gradient (FSVRG). FSVRG's concept is to conduct one full calculation in the server, followed by multiple distributed stochastic updates on each client. The updates are made by performing a single update on a random permutation of data. The FSVRG method is primarily concerned with sparse data. [7]

FedProx is another algorithm for FL that has been proposed. FedProx and FedAvg are similar in that they both need the selection of groups of devices at each iteration. Local updates are carried out and then pooled together to create a global update. FedProx is intended to be a tweak to the original FedAvg algorithm. FedProx adds a few tweaks that improve performance and heterogeneity. The argument behind this is that a variety of devices used for FL will often have their device limits, so expecting all devices to accomplish the same amount of work would not be optimal or reasonable. [7]

CO-OP suggests an asynchronous approach, whereas FedAvg and FSVRG rely on synchronized model updates. CO-OP, unlike FedAvg, quickly incorporates every received client model into the global model. Rather than directly averaging models, the merger of a local and a global model is accomplished by a weighting mechanism depending on the age difference between the two models. This is due to the fact that in an asynchronous framework, some clients will train on old models while others will train on newer models.

2.2.3 Security aspects of FL

However, FL must operate within a secure environment in order to start up. Each FL system must use one of the secure methods listed below:

- Secure Multi-party Computation (SMC) - Secure multi-party computation (also known as multi-party computation, SMPC, or MPC) is a cryptographic approach that allows two or more parties to do a computation using their private data without revealing their private data to each other. [8] MPC can be used to sign transactions in the context of digital assets instead of using individual private keys. MPC divides up the signature procedure among several computers. Each computer has a bit of private data that represents a portion of the key, and they work together to sign transactions in a distributed fashion. [8]
- Differential Privacy - Differential privacy allows companies to acquire and share aggregate information about user habits while protecting individual users' privacy. Differential privacy attempts to solve security issues by introducing "noise" or unpredictability to the data, making it impossible for users to identify individual data points. [9] At the very least, such a system allows for plausible denial. As a result, individual privacy is protected while data accuracy is

maintained to a minimum. While FL allows for machine learning without collecting raw data, differential privacy (DP) is a quantitative measure of data anonymization that, when applied to machine learning, can alleviate worries about models retaining sensitive user data. [9]

- Homomorphic Encryption - Homomorphic encryption aims to make it possible to do computations on encrypted data [10]. As a result, data can be kept private while being analysed, allowing useful operations to be completed with data stored in untrustworthy locations. A homomorphic cryptosystem works similarly to other types of public encryption in that it encrypts data with a public key and only provides access to the decrypted material to those who have the matching private key. However, what distinguishes Homomorphic encryption from other types of encryption is that it employs an algebraic system that allows users to do a wide range of computations (or operations) on the encrypted data.

2.2.4 Real-world FL Applications

The real-world applications of FL will be explored in this chapter.

Since protected health information cannot be shared freely due to specific laws, healthcare is one of the industries that can gain the most from federated learning. In this approach, a large amount of data from various healthcare databases and devices may be used to construct AI models while remaining compliant with rules. The use of federated learning for predicting clinical outcomes in COVID-19 patients is one of the real-life instances. [4]

Personalization is heavily reliant on the data of each unique user. However, as more people become concerned about how much data they would prefer not to disclose to others, sites like social media, eCommerce platforms and other online locations. Advertising can take advantage of federated learning to stay afloat and alleviate people's anxieties while relying on personal data from consumers. Facebook is now re-engineering its ad system to prioritise user privacy. The company is experimenting with on-device learning by running algorithms locally on phones to see what ads users are interested in. The results are then consolidated, encrypted and transmitted back to the cloud server for marketing teams to analyse. [4]

Because federated learning can make real-time predictions, it is being used to construct self-driving cars. Real-time updates on road and traffic conditions may be included in the data, allowing for continuous learning and faster decision-making. This may result in a more enjoyable and secure self-driving automobile experience. The automotive industry is a promising area for federated machine learning implementation. [4]

Users can benefit from on-device keyboard implementations by recommending relevant material, such as search queries connected to the input text. Federated Learning can be used to train machine learning models for both triggering and ranking the objects that can be offered in the present environment. Google's Gboard mobile keyboard team use this specific FL system. [11]

2.2.5 Benefits and Challenges

FL has drawbacks despite offering all these benefits. Federated learning is a new concept in the machine learning world, but it already has a lot of advantages over traditional, centralised systems. The benefits of federated learning are numerous.

Data security is one of the advantages that FL offers. The necessity for a data pool for the model is eliminated by keeping the training dataset on the devices. Furthermore, FL allows for continuous learning in real time. Since models are regularly modified utilising client input, there is no need to collect data for continuous learning. Furthermore, FL improves hardware efficiency. This solution requires less complex hardware because federated learning models do not require a single complex central server to analyse data. Federated learning also makes it easier to access a wide range of data, even if data sources can only communicate at specific times. [12]

Security and performance are two of FL's challenges. It is feasible that federated learning models will need to communicate frequently between nodes. This means that system requirements include high bandwidth and storage capacity. In federated learning, models from various devices are combined to create a superior model. Device-specific factors may limit the generalisation of models from some devices, lowering the accuracy of the model's next generation. [12]

In federated learning, data is not collected on a single entity or server. Instead, it is collected and analysed by numerous devices. This has the potential to increase the attack surface. Even though only models, not raw data, are sent to the central server, models can be reverse-engineered to identify client data. [12]

2.3 FL tools and Frameworks

The clients must be fed with their local datasets before the FL system can operate. Two packet capture technologies could be used to create the datasets that will be supplied to the clients for this project. Tcpdump and Wireshark are these tools. These tools are similar in terms of functionality. Tcpdump is a packet analyzer that may be run from the command prompt [13]. It is typically used to examine network traffic by intercepting and showing packets created or received by the computer on which it is installed [13]. Wireshark is a data packet capture tool with a graphical user interface. [13]

Furthermore, numerous open-source frameworks might be used to build the project's software. The following are some of the frameworks that could be used in this project:

- Google's TensorFlow Federated (TFF) is an open-source federated learning system written in Python 3. Google's requirement to deploy mobile keyboard predictions and on-device search was the driving force behind TFF. TFF is a tool that Google uses to help customers. [14]
- Intel® Open Federated Learning is an open-source Python 3 project that Intel created to apply FL to sensitive data. OpenFL contains bash deployment scripts and uses certificates to secure communication, but the framework's user must manage the majority of this on his own. [14]

- PySyft is an open-source Python 3 package that utilizes FL, differential privacy, and encrypted computations to enable federated learning for research. It was created by the OpenMined community and is designed to operate with deep learning frameworks like PyTorch and TensorFlow. [14]

2.4 Network Programming

Since the beginning of the Internet, applications have used network capabilities via the socket interface. The majority of languages are supported by this interface, which makes it simple to send and receive data from numerous network hosts. This section entails the principles of developing distributed programmes utilising the socket interface.

The TCP/IP protocol suite, which is the foundation of internet protocols, implements two data delivery services:

- TCP: A reliable and controlled data transmission protocol. TCP makes sure that all bits get to where they're supposed to and that transmission speeds change according to how congested the network is.
- UDP: An unreliable and ungoverned transmission protocol. The delivery of every bit is not guaranteed by UDP. The application also manages the transmission rate.

The code for a server and a client that uses TCP sockets is shown in the figure below. The socket structure is the central component of the code. A socket structure is created by the server, which then binds it to a host and port and begins to listen for connections. Once established, a connection makes use of the conn variable to deliver and receive data. It uses the recv and send techniques for this.

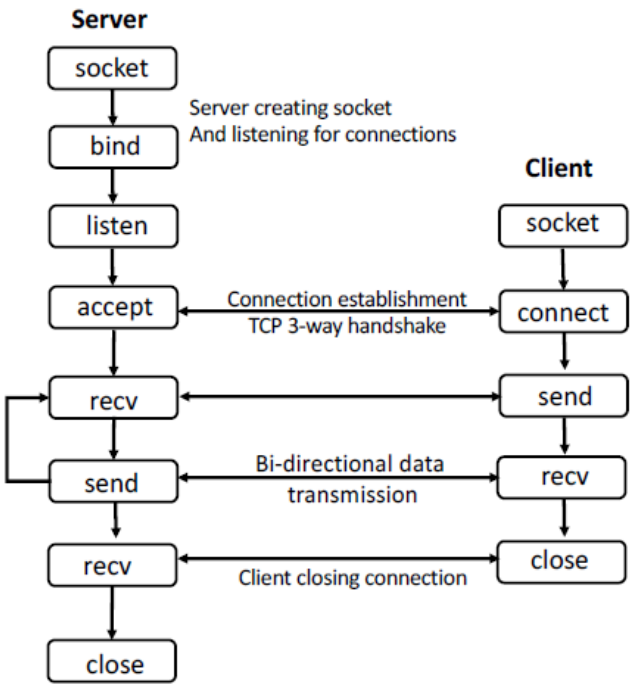

Fig. 5: TCP Socket structure [15]

A different communication approach is UDP sockets. There is no link construction. Every UDP packet has its own existence. The UDP sockets use a protocol based on datagrams which is SOCK DGRAM. One datagram is sent, one is received, and then the connection is terminated. There is no connection-listening feature on the server. Additionally, it gets one UDP datagram at a time.

The transmission pace of the application is managed by clients and servers. The operating system does not pace the data as TCP does. As a result, the network can be quickly overloaded using packets.

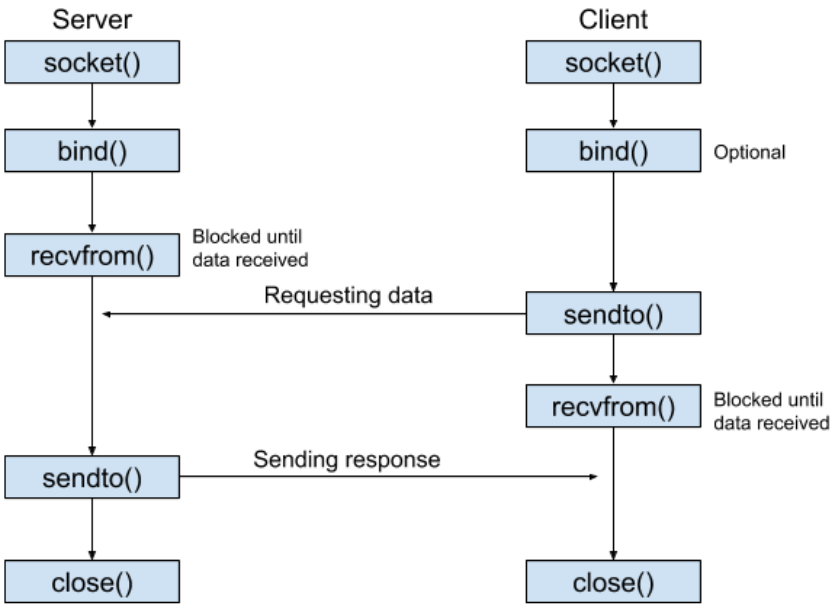

Fig. 6: UDP Socket structure [16]

The main weakness in the above examples is that they can only handle one client at once. An application can use threads to enable the same server to handle multiple clients at once. [17]

The fundamental concept behind almost all current programming languages is multithreading, particularly Python due to its simple thread implementation. [17]

A thread is a portion of code within a programme that can run separately from other code. A thread executes inside the same context while sharing memory and other executable resources. Multithreading is the simultaneous execution of several threads within a single process. [17]

3. Theory/Methodology

The purpose of this section is to illustrate the project's methodology. In addition, it will explain further the tools that were implemented to create the Pythia oracle. This section will be divided into three different chapters: Pythia architecture, Data collection and Pythia creation process.

3.1 Pythia Architecture

Pythia will be a Cross-Device Federated Learning System and it will consist of a server and ten clients. Updates from clients will be gathered and summarised by an aggregator on the server. The FedAvg algorithm will be implemented by the server to carry out its process. The clients will be neural networks that will train their local datasets and send their updates to the server. A neural network is a group of algorithms that seeks to find underlying links in a set of data by employing a method that is similar to how the human brain functions. The results section will go into further detail about how the datasets and neural networks were constructed. Moreover, the results section will explain further the codes that were built to implement both the server and the client system. In addition, other nodes will communicate with Pythia by requesting network information and the oracle will distribute the conditions to each of them by using a simple socket interface.

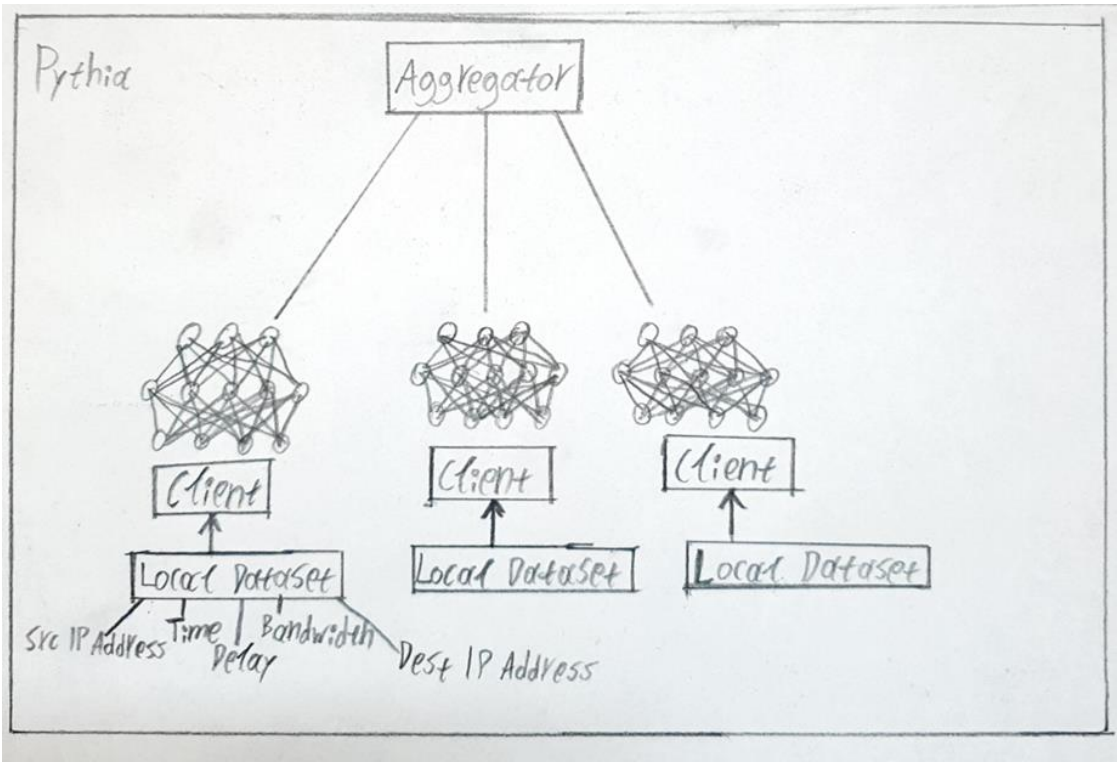

Fig. 7: Pythia oracle architecture

The code that will construct Pythia oracle will be based on a specific programming language which is python. The Anaconda navigator will also be used to execute the python codes and manage federated learning packages. The anaconda navigator will help to avoid the use of command-line commands.

The Pythia oracle will divide into three different codes:

- Server code - The FedAvg algorithm and other crucial features will be included in the first code, which will function as a server system. Additionally, this server system will have other vital elements like client creation and data batching. The second code, which is needed to implement the global model, will import and use each of these functions.
- Global model code - The second code as referred before will be the global model implementation code. This code will take all the functions that are made in the server's code and use them to compile Pythia's FL system.
- Socket programming – The third code will be supplementary and it will be used to connect other nodes with Pythia. This connection will be made with Multi-threaded socket programming.

As mentioned above, the results section will explain further the work that was made to create Pythia's FL model. In this section, it will be discussed the steps that were made to complete the target of this paper. These steps are:

- Data collection – The collection of private data by using specific network packet capturing tools.
- Model Framework selection – The selection of Pythia's model framework.
- Construction of the centralised service (Server) – The creation of the server system and other important functions that will be implemented in the global system.
- Design of the client system – The construction of the client system.
- Construction of the global model – The design of the global model.
- Set up training and testing processes – The separation of the data collected and putting them into training and testing processes.
- Addressing privacy and security - Adding privacy and security into Pythia's oracle.
- Communication with other nodes – The connection between the server with other nodes and communication about the network.

All these steps are crucial to the construction of the Pythia oracle.

3.2 Data Collection

The Pythia oracle will consist of a server and clients/devices, as mentioned in earlier sections of this study. A local dataset created using Wireshark will be fed to those devices. This dataset will include information about passive monitored data captured from Wireshark such as source IP addresses, destination IP addresses, date/time, delay and bandwidth. The following figure shows the dataset's architecture.

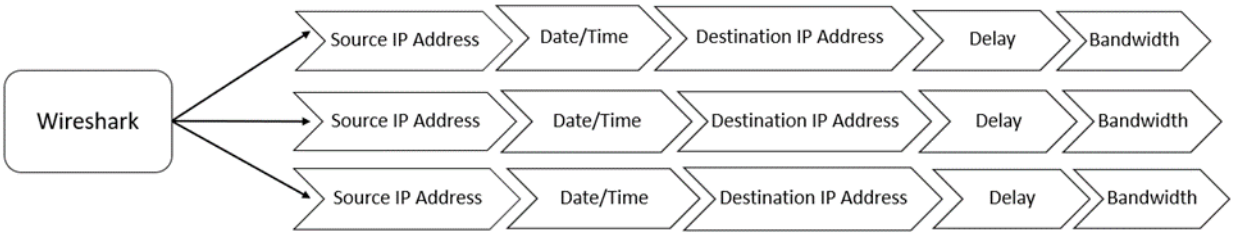

Fig. 8: Dataset contents

Wireshark already provides the source, destination IP addresses and the date/time of each packet. However, the delay of the packets should be calculated. To add the delay of the packets in the dataset a new function was made in Wireshark. This function has as a base this calculation:

Pythia_Packets_Delay = Timestamp_ACK_sent – Timestamp_Packet_sent

A computer transmits an acknowledgement code (ACK), a particular form of a distinctive signal, to indicate that data has been successfully transmitted. Therefore, the packet delay is the time between the sender's communication of the packet and the receiver's delivery of the associated ACK.

The bandwidth can be calculated by subtracting the time between the first and last packets that were received and dividing the result by the total number of bytes in all the packets. This process will be added to a function in Wireshark. The Date/Time column was separated into two different columns. Since this dataset contains a huge amount of data a small portion of it will be included in the appendices (Appendix 1).

A part of the dataset that has been created is depicted in the figure below:

	A	B	C	D	E	F
1	Date	Time	Source IP address	Destination IP address	Bandwidth	Delay
2	8/9/2022	2:15:16 PM	192.168.1.3	52.85.221.134	55	0
3	8/9/2022	2:15:16 PM	52.85.221.134	192.168.1.3	66	0.014674
4	8/9/2022	2:15:16 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2a00:1450:4001:80f::2004	75	0
5	8/9/2022	2:15:16 PM	2a00:1450:4001:80f::2004	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.04809
6	8/9/2022	2:15:18 PM	192.168.1.3	116.202.225.117	55	0
7	8/9/2022	2:15:18 PM	116.202.225.117	192.168.1.3	66	0.053148
8	8/9/2022	2:15:21 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2a00:1450:4001:800::200e	75	0
9	8/9/2022	2:15:21 PM	2a00:1450:4001:800::200e	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.051756
10	8/9/2022	2:15:22 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2a00:1450:4001:802::2001	75	0
11	8/9/2022	2:15:22 PM	2a00:1450:4001:802::2001	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.050298
12	8/9/2022	2:15:25 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2620:1ec:46::45	75	0
13	8/9/2022	2:15:25 PM	2620:1ec:46::45	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.015461
14	8/9/2022	2:15:26 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2606:4700::6812:ad2	75	0
15	8/9/2022	2:15:26 PM	2606:4700::6812:ad2	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.018712
16	8/9/2022	2:15:26 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2606:4700::6812:1c26	75	0
17	8/9/2022	2:15:26 PM	2606:4700::6812:1c26	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.012052
18	8/9/2022	2:15:27 PM	192.168.1.3	44.241.46.189	55	0
19	8/9/2022	2:15:27 PM	44.241.46.189	192.168.1.3	66	0.217609
20	8/9/2022	2:15:28 PM	192.168.1.3	116.202.225.117	55	10.002802
21	8/9/2022	2:15:28 PM	116.202.225.117	192.168.1.3	66	0.056459
22	8/9/2022	2:15:38 PM	192.168.1.3	116.202.225.117	55	10.012552
23	8/9/2022	2:15:39 PM	116.202.225.117	192.168.1.3	66	0.058699
24	8/9/2022	2:15:39 PM	192.168.1.3	20.199.120.85	153	0
25	8/9/2022	2:15:39 PM	20.199.120.85	192.168.1.3	223	0.058817
26	8/9/2022	2:15:39 PM	192.168.1.3	20.199.120.85	54	0.046047
27	8/9/2022	2:15:39 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2a00:1450:4001:831::2003	75	0

Fig. 9: Dataset creation

3.3 Pythia Creation Process

The steps taken to construct the Pythia Oracle will be further explained in this chapter.

3.3.1 Model Framework Selection

Selecting the model implementation is the first step. In order to select the optimal model, the model framework should offer specific federated learning capabilities in the project's application style. Selection criteria include things like domains like imaging, natural language processing, or tabular data as well as framework compatibility with current infrastructure. Popular choices include TensorFlow or PyTorch.

TensorFlow was selected as the library for this project. TensorFlow is an end-to-end open-source platform for machine learning. [18] Researchers can advance the state-of-the-art in ML thanks to its extensive, adaptable ecosystem of tools, libraries, and community resources, while developers can simply create and deploy ML-powered applications. [18] TensorFlow contains TensorFlow Federated (TFF) which is an open-source framework for federated learning and other computations on decentralized data. Due to the operating system of the infrastructure being used in this project, the TFF was not implemented in the final code. More specifically, the Windows operating system does not support TFF. This problem is explained in further detail, along with potential solutions, in the discussion chapter of the results section.

3.3.2 Construction of the centralised service (Server)

The server of Pythia's architecture will contain certain functions to manage the clients that will be imported after in the global model code. One of them is to create the clients and put specific prefixes for each of them like client_1, client_2 etc.

One of the key responsibilities of the server is acting as an aggregator. For the server to function as an aggregator, it must have selectors or features that will randomly select the data models it will receive. The server should then divide this random data into more manageable portions (shards). Afterwards, the data needs to be processed in batches. To do this, a method was developed that creates a tfds object from a client's data shard. The client's data shard contains random data. The term "TensorFlow datasets" is an acronym for a collection of ready-to-use datasets.

The other important part of the server is the aggregation algorithm. For this project, a Federated Averaging algorithm (FedAvg) was created to aggregate the client's data. The following equation describes the FedAvg algorithm:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

Fig. 10: FedAvg algorithm [19]

On the right-hand side of this equation, the model estimates the weight parameters of each client based on the loss values across all of the data points implemented for training. On the left-hand side, each of these parameters is scaled and then summed all of them together.

The Pythias server should also control the MLP that its clients utilise. A multilayer perceptron is a feedforward artificial neural network that creates a collection of outputs from a set of inputs (MLP). [20] An input layer, an output layer and a hidden layer are the three different kinds of layers that compose the MLP. The input layer is where the signal is received for processing. The necessary tasks, such as classification, regression and prediction are completed by the output layer. The real computational engine of the MLP consists of an arbitrary number of hidden layers that are placed between the input and output layers. [20] Similar to a feed-forward network, data flows from the input to the output layer of an MLP in the forward direction. The MLP's neurons are trained with the use of the back propagation learning technique. Because MLPs are designed to mimic any continuous function, they can tackle problems that cannot be separated linearly.

In the Pythia server code, a specific class for creating MLPs was developed. This class will be imported into the global model code and used in the client's local training process.

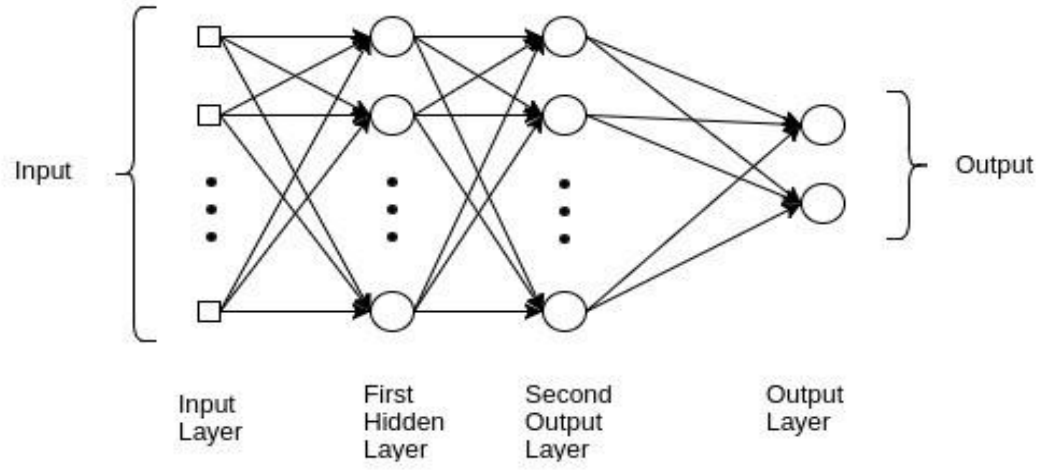

Fig. 11: Multi-layer Perceptron (MLP) Example [21]

The server will also provide a test mechanism. This function will evaluate the data and demonstrate the expected delay of packets sent across a network. Additionally, it will print the global model loss.

3.3.3 Design of the client system

This system must be able to carry out client-side training activities and coordinate the model parameters with the central service. The client system will also need updated parameters from other clients on the network to update the local model. The client

system will be incorporated into the global model code for this project. The client system will provide a load function that will be utilized to load the dataset.

Separating the data into training and testing sets is a critical step in analysing data mining methods. When you separate a data set into a training set and a testing set, the majority of the data is frequently utilised for training while a smaller amount is used for testing. The client system code will also comprise this step.

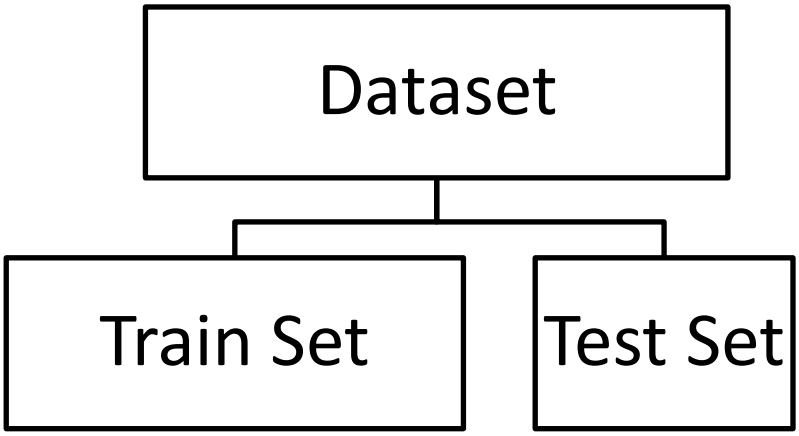

Fig. 12: Dataset splitting

3.3.4 Construction of the global model

The global model will include a number of functions. The client system will be incorporated into the global model, as was already described.

The process of constructing the clients comes next, after the processes of adding the dataset and dividing it into training and test sets. The method added to the server code to create clients will be imported into the global model. Following that, each client's training data will be batch processed. The next step is to develop a function for batching the test set for each client once the previous process is complete. The global communication rounds were also defined.

An optimizer function will be created once the batching procedures have been completed. This function will include an optimizer, a loss function and other metrics that later in the code will compile into the client models.

Following the completion of these procedures, the global model begins to be initialised. The global training loop is started by importing the MLP model that was constructed in the server's code. The data from each local training will then be gathered using the FedAvg algorithm and fed into the aggregator-server. Additionally, a new shuffle function was created to randomise the data.

The development of local models happens once the global model employs the FedAvg algorithm. The Multi-Layer Preceptor was added along with the optimizer, loss

function, and metrics that were previously added while the local models were being built. Next, the weights of the local models are set. The local models that are built will then need to fit in the client's data. After that, the global model will be updated after taking the average of all the local models. In order to free up memory and avoid a slowdown, a function was also developed to remove the session after each communication round was completed.

The last process is to test the global model and print out metrics after each communications round. The global model's operation is shown in the following figure:

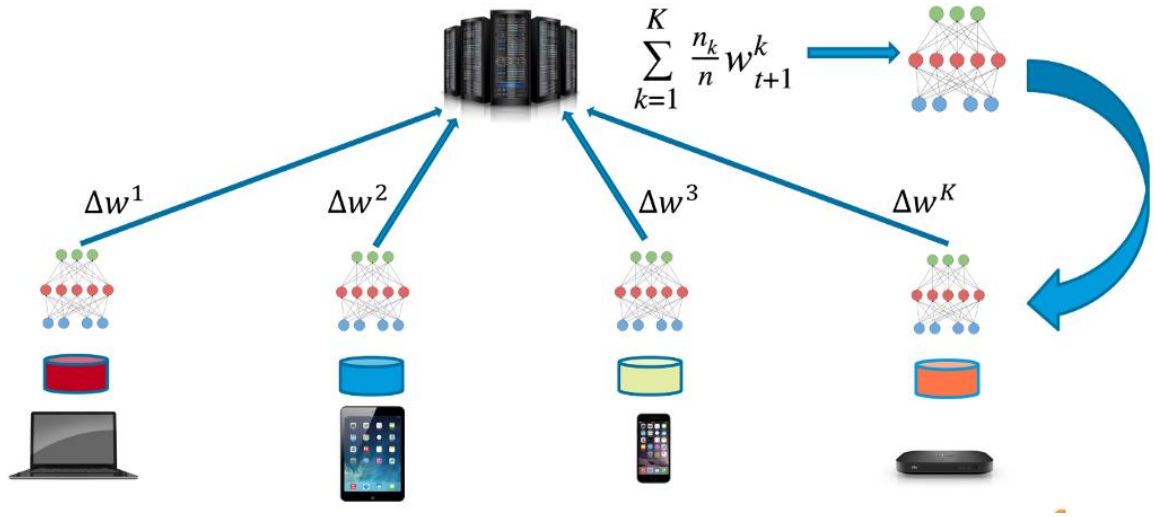

Fig. 13: Federated Learning Model Example [22]

3.3.5 Set up training and test processes

To train the local models for a certain session, the federated learning system has to know which private data should be used from each client. [23] The central service (server) or another user have to provide this data. As a result, it is necessary to handle the metadata about the available data in some way. The central service usually does this. The training logic of Pythia's oracle consists of two primary loops: the outer loop handles global iteration, while the inner loop handles iterating over local training for each client. However, there is an implied third one that accounts for local epochs and is handled by the epochs argument in the Pythias model.fit method.

Various prerequisites must be met before the training can begin. The MLP created from the server's code must be imported first. Obtaining the global model's initialised weights is the second step. Following that, the client training iteration can proceed.

The parameters of the current global model were applied to the initialization weights of a new model object that was built specifically for local training. After that, the local model (client) performed one training session. The new weights were scaled and placed into a separate function after training. Moving to the outer loop, the global model was updated using the sum of all the scaled local training weights. A complete global training epoch ends with this process. After each communications round, the global model was evaluated for the testing procedure, and metrics were printed out.

3.3.6 Addressing privacy and security

One or more participants have access to the final model locally. This model can be inverted to get information about the underlying training data because it was trained by distributing weights among various parties.

So it's crucial to decide what risks are acceptable for the model itself. These dangers could include the possibility of being able to re-identify a participant in a particular training set or to regenerate the training set itself, which might contain sensitive IP.

In Pythia Oracle, all data were randomised using various processes to lower the risks associated with addressing privacy and security. Before the training processes, two functions randomise the data, making it more difficult for other nodes to attack the system.

3.3.7 Communication with other nodes

In addition to its role in providing information about the state of the network (delay, bandwidth), the Pythia Oracle will be programmed to do the same for other nodes. These nodes may represent various businesses or organisations that desire to use the network with the lowest latency or for other purposes.

Programming using multiple threads will be implemented to facilitate communication between the Pythia server and other nodes. This part of socket programming is used to improve the performance of Pythia's oracle with the other nodes. For instance, the server will use several threads to handle requests for data concurrently.

Minimization and more effective utilisation of computing resources is also one of the benefits of multithreading. As requests from one thread do not block requests from other threads, application responsiveness is increased. [17] The following figure illustrates this application.

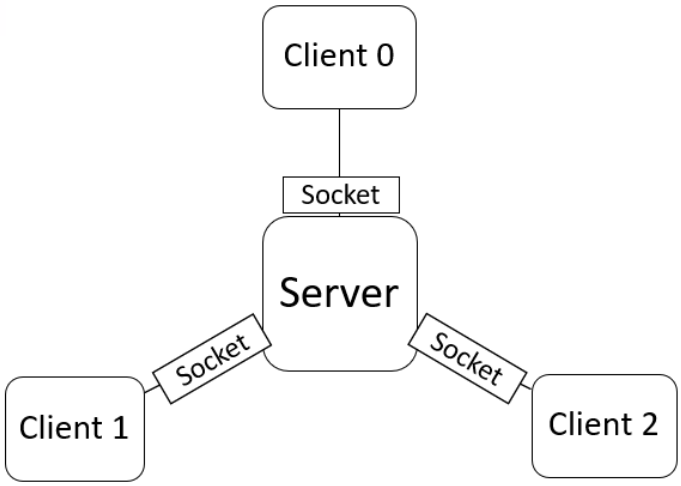

Fig. 14: Multithreaded Sockets

4. Results

This section will detail the outcomes of the project's progress. It will consist of three parts: Data preprocessing, Pythia FL system and socket programming. All the codes utilised in the following chapters will be inserted in the appendices (Appendix 2,3,4).

4.1 Data Preprocessing

This chapter will describe the dataset design process that followed data collection. The Pythia system will then be fed with this dataset.

The first step is to import libraries. The main two libraries used for this part are pandas and NumPy. The pandas library offers data structures and operations for manipulating numerical tables and time series. [24] The NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. [25] After importing the libraries the next step is to load the dataset, the dataset was loaded with the help of pandas named pd.

```
df = pd.read_csv('Path where the CSV file is stored\File name.csv')
df
```

	Date	Time	Source IP address	Destination IP address	Bandwidth	Delay
0	8/9/2022	2:15:16 PM	192.168.1.3	52.85.221.134	55	0.000000
1	8/9/2022	2:15:16 PM	52.85.221.134	192.168.1.3	66	0.014674
2	8/9/2022	2:15:16 PM	2a02:587:7813:500:f0f7:f781:b084:7d42	2a00:1450:4001:80f::2004	75	0.000000
3	8/9/2022	2:15:16 PM	2a00:1450:4001:80f::2004	2a02:587:7813:500:f0f7:f781:b084:7d42	86	0.048090
4	8/9/2022	2:15:18 PM	192.168.1.3	116.202.225.117	55	0.000000
...	...	...	...	...	...	...
14944	8/9/2022	2:21:13 PM	2606:4700:4400::ac40:9aa3	2a02:587:7813:500:b1f1:bd6e:afbb:bcbb	1434	0.000063
14945	8/9/2022	2:21:13 PM	2a02:587:7813:500:b1f1:bd6e:afbb:bcbb	2606:4700:4400::ac40:9aa3	74	0.000017
14946	8/9/2022	2:21:14 PM	2606:4700:4400::ac40:9aa3	2a02:587:7813:500:b1f1:bd6e:afbb:bcbb	1434	0.004248
14947	8/9/2022	2:21:14 PM	2606:4700:4400::ac40:9aa3	2a02:587:7813:500:b1f1:bd6e:afbb:bcbb	1434	0.024639
14948	8/9/2022	2:21:14 PM	2a02:587:7813:500:b1f1:bd6e:afbb:bcbb	2606:4700:4400::ac40:9aa3	74	0.000123

14949 rows × 6 columns

Fig. 15: Dataset Loading

From the figure is understood that an issue occurs in terms of the type of data. Most of the data are containing strings, so any FL system cant process them because they contain characters. As a result, the data need to be converted either into integers or floating-point numbers. The first two columns (Date, Time) will be explained later in this section on how they are converted into integers. For the next two columns, the processing was difficult because even if specific punctuation was removed, the data could not be placed with all these numbers together. A new function was made for this issue that translates this information into numerical data. This function was completed with the help of sklearn.preprocessing.LabelEncoder library. This library can be used to normalize labels. As long as they are hashable and comparable, they can also be used to convert non-numerical labels into numerical ones. [26] The following figure shows this function:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Source IP address']=le.fit_transform(df['Source IP address'])
df['Destination IP address']=le.fit_transform(df['Destination IP address'])
df
```

	Date	Time	Source IP address	Destination IP address	Bandwidth	Delay
0	8/9/2022	2:15:16 PM	18	117	55	0.000000
1	8/9/2022	2:15:16 PM	116	18	66	0.014674
2	8/9/2022	2:15:16 PM	82	69	75	0.000000
3	8/9/2022	2:15:16 PM	65	83	86	0.048090
4	8/9/2022	2:15:18 PM	18	1	55	0.000000
...	...	...	...	...	...	...
14944	8/9/2022	2:21:13 PM	47	82	1434	0.000063
14945	8/9/2022	2:21:13 PM	81	51	74	0.000017
14946	8/9/2022	2:21:14 PM	47	82	1434	0.004248
14947	8/9/2022	2:21:14 PM	47	82	1434	0.024639
14948	8/9/2022	2:21:14 PM	81	51	74	0.000123

14949 rows × 6 columns

Fig. 16: IP Translation

After converting the Source IP and Destination IP address columns into a numerical number then the next are the first two columns. For this step another function was imported to help with the data organisation, this library is called datetime. The datetime module supplies classes for manipulating dates and times. [27] When the datetime module was utilized in the Date column, it changed its format of it and make it like this format 'yyyy-mm-dd'. From the date column, three new columns were generated, one for the year, one for the month and another one for the day. From the time column also three new columns were produced, one for the hour, one for minutes and another one for seconds. Then, the date and time columns that contained the strings were dropped. Furthermore, the delay column was moved to the first column for other processing purposes which will be discussed later in this chapter. The following figures display all this process:

```
df['Date'] = pd.to_datetime(df['Date'])
df['date_year'] = df['Date'].dt.year
df['date_month_no'] = df['Date'].dt.month
df['date_day'] = df['Date'].dt.day
```

```
df['Time'] = pd.to_datetime(df['Time'])
df['hour'] = df['Time'].dt.hour
df['min'] = df['Time'].dt.minute
df['sec'] = df['Time'].dt.second
```

Fig. 17: Data separation

```
df.drop(['Date','Time'], axis=1)
first_column = df.pop('Delay')
df.insert(0, 'Delay', first_column)
```

Fig. 18: Data reorganization

After that, the new dataset was saved and stored with the help of the pandas DataFrame to_csv() function that converts DataFrame into CSV data.

```
df.to_csv(r'Path where the CSV file is stored\File name.csv')
df
```

	Delay	Bandwidth	Source IP address	Destination IP address	date_year	date_month_no	date_day	hour	min	sec
0	0.000000	55	18	117	2022	8	9	14	15	16
1	0.014674	66	116	18	2022	8	9	14	15	16
2	0.000000	75	82	69	2022	8	9	14	15	16
3	0.048090	86	65	83	2022	8	9	14	15	16
4	0.000000	55	18	1	2022	8	9	14	15	18
...	...	...	...	...	...	...	...	...	...	...
14944	0.000063	1434	47	82	2022	8	9	14	21	13
14945	0.000017	74	81	51	2022	8	9	14	21	13
14946	0.004248	1434	47	82	2022	8	9	14	21	14
14947	0.024639	1434	47	82	2022	8	9	14	21	14
14948	0.000123	74	81	51	2022	8	9	14	21	14

14949 rows × 10 columns

Fig. 19: Dataset storing

Moreover, another two functions were added to test the new dataset. The first one concerns whether the dataset is missing any values and the second one includes the data types for each of the dataset's columns.

```
df.isnull().sum()
```

```
Delay                    0
Bandwidth                0
Source IP address        0
Destination IP address   0
date_year                0
date_month_no            0
date_day                 0
hour                     0
min                      0
sec                      0
dtype: int64
```

```
df.dtypes
```

```
Delay                    float64
Bandwidth                  int64
Source IP address          int64
Destination IP address     int64
date_year                  int64
date_month_no              int64
date_day                   int64
hour                       int64
min                        int64
sec                        int64
dtype: object
```

Fig. 20: Dataset evaluation

The figures mentioned above illustrate two things. The first is that there are not any missing values from the dataset and the second is that all the columns except the delay one are containing integers. The delay column consists of floating point numbers.

4.2 Pythia FL System

This chapter will include all the work that was made to achieve the construction of Pythia's Federated Learning system. This chapter will be separated into two parts: the server's code and the global model code.

4.2.1 Server code

This code will contain important functions that will be utilized in the global model code. Before starting to explain these functions, the libraries that are imported should be mentioned. These modules are the following:

- NumPy – As was already mentioned, NumPy provides a wide range of mathematical functions, as well as Fourier, transforms, random number generators and linear algebra functions.
- Random - A Python built-in module called Python Random is used to create random integers.
- OS - In Python, the OS module has functions for creating and deleting directories (folders), retrieving their contents, and altering and locating the current directory.
- Sklearn.model_selection.train_test_split - The train_test_split function of the sklearn. model_selection package in Python splits arrays or matrices into random subsets for train and test data, respectively. [28]
- Sklearn.utils.shuffle - Consistently shuffle sparse matrices or arrays.
- TensorFlow - As mentioned before, TensorFlow is an open-source library for machine learning that offers a range of tasks but also emphasizes the training and inference of deep neural networks.
- Tensorflow.keras.models.Sequential - For a simple stack of layers where each layer has precisely one input tensor and one output tensor, a sequential model is sufficient. [29]
- Tensorflow.keras.layers.Activation – This library applies an activation function to output.
- Tensorflow.keras.layers.Dropout – Dropout is applied to the input. To avoid overfitting, the Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training. [30]
- Tensorflow.keras.layers.Dense – The dense layer is the typical layer of a neural network with several connections.
- Tensorflow.keras.optimizers.SGD – An optimizer for gradient descent is imported by this module. An iterative technique for maximising an objective function with sufficient smoothness qualities is stochastic gradient descent or SGD for short.
- Tensorflow.keras.backend – This library imports the Keras backend API.

```
import numpy as np
import random
import cv2
import os
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras import backend as K
```

Fig. 21: Libraries importation

The next stage is to begin developing the Pythias model's clients when all the necessary libraries have been incorporated into the code. An entirely new function was created to achieve that. A list of customer names is created by this function. The function then takes a data shard and instals it at each client, keeping in mind that there must be an equal number of clients and shards. The process that was used to develop this function is depicted in the following figure:

```
def create_clients(Features_list, Targets_list, num_clients=10, initial='clients'):
    '''   args:
            Features_list: a list of numpy arrays of the training columns
            Targets_list: a list of numpy arrays of the testing columns
            num_client: number of fedrated members (clients)
            initials: the clients'name prefix, e.g, clients_1

    '''

    #create a list of client names
    client_names = ['{}_{}'.format(initial, i+1) for i in range(num_clients)]


    #shard data and place at each client
    size = len(df)//num_clients
    shards = [df[i:i + size] for i in range(0, size*num_clients, size)]

    #number of clients must equal number of shards
    assert(len(shards) == len(client_names))

    return {client_names[i] : shards[i] for i in range(len(client_names))}
```

Fig. 22: Create clients function

In this case, the number of federated clients will be ten and their names will be like this clients_1, clients_2, etc.

The next step is to batch-process each client's data into a TensorFlow dataset. This process was compressed into a small function called batch data in order to simplify this process and prevent repetition.

```
def batch_data(data_shard, bs=32):
    '''Takes in a clients data shard and create a tfds object off it
    args:
        shard: a data, label constituting a client's data shard
        bs:batch size
    return:
        tfds object'''
    #seperate shard into two lists
    data = zip(*data_shard)
    dataset = tf.data.Dataset.from_tensor_slices((list(data)))
    return dataset.shuffle(len(Features_list)).batch(bs)
```

Fig. 23: Batching data method

The tuple split into two lists, as depicted in the figure. These lists were then combined to produce a TensorFlow dataset object that was shuffled and batched.

The most important stage of this code is the design of the MLP which is the next step. A class called SimpleMLP were built that contains the three layers of Pythia's neural network. These layers are the input layer, the hidden (middle) one and the output one.

The following function defines the model layer per layer: model = sequential() (sequentially). The input layer of this model will be a dense layer composed of 64 neurons. Every neuron in a dense layer must be coupled to every neuron in the layer above or below it. In the input layer's case, the input shape was used to denote that this layer would be the first. The number 9 was inserted to represent the training data's number of columns since this layer must also have the same form as the training set. The decision to activate a neuron or not was made using the ReLU activation function. The ReLU activation function was implemented in all three layers. For the middle layer, a dense layer of sixty-four neurons was chosen. However, the output layer was selected to have a dense layer of only one neuron.

A dropout layer was then used to randomly drop 20% of the nodes. This dropout layer was added to the model to improve accuracy and prevent overfitting. For the middle/hidden layer, the procedure is repeated. Furthermore, in the MLP class, a static method was used. A static method is a technique utilized to link the class instead of the class's object.

```
class SimpleMLP:
    @staticmethod
    def build(shape, classes):
        model = Sequential()
        model.add(Dense(64, activation= "relu",input_shape=(9,)))
        model.add(Dropout(0.2))
        model.add(Dense(64, activation= "relu"))
        model.add(Dropout(0.2))
        model.add(Dense(1, activation= "relu"))
        return model
```

Fig. 24: MLP Creation

The second most important step in this code is the Federated Averaging algorithm. This algorithm was encapsulated into three functions. These are:

1. Weight_scalling_factor - The weight scaling factor function determines the proportion of a client's local training data to the entire training data held by all customers. The client's batch size must be established to calculate the client's number of data points. The total amount of the global training data was then calculated. Next, a fractional calculation of the scaling factor was performed. It is obvious that this cannot be the approach employed in a real-world application. Due to the disconnection of the training data, no one client can precisely predict the size of the merged set. In such a case, each client will be needed to transmit the number of data points used for training after each local training step while updating the server with new parameters.
2. Scale_model_weights - Each weight in the local model is scaled by the scale model weights function according to the value of the scaling factor measured in the previous function.
3. Sum_scaled_weights – This function sums up the scaled weights of all clients.

```
def weight_scalling_factor(clients_trn_data, client_name):
    client_names = list(clients_trn_data.keys())
    #get the bs
    bs = list(clients_trn_data[client_name])[0][0].shape[0]
    #first calculate the total training data points across clinets
    global_count = sum([tf.data.experimental.cardinality(clients_trn_data[client_name]).numpy()
                        for client_name in client_names])*bs
    # get the total number of data points held by a client
    local_count = tf.data.experimental.cardinality(clients_trn_data[client_name]).numpy()*bs
    return local_count/global_count
```

Fig. 25: Weight scalling factor

```
def scale_model_weights(weight, scalar):
    '''function for scaling a models weights'''
    weight_final = []
    steps = len(weight)
    for i in range(steps):
        weight_final.append(scalar * weight[i])
    return weight_final
```

Fig. 26: Scale model weights function

```
def sum_scaled_weights(scaled_weight_list):
    '''Return the sum of the listed scaled weights. The is equivalent to scaled avg of the weights'''
    avg_grad = list()
    #get the average grad accross all client gradients
    for grad_list_tuple in zip(*scaled_weight_list):
        layer_mean = tf.math.reduce_sum(grad_list_tuple, axis=0)
        avg_grad.append(layer_mean)

    return avg_grad
```

Fig. 27: Scaled weights summation

The last step to complete this chapter is to make a function that will test Pythia's model. This function will test the model and print its loss of it. The Mean Squared Error will be used to calculate the loss of the global model (MSE). The MSE is the regression loss function that is most frequently employed. The difference in squared values between the true and predicted values is the loss.

```
def test_model(X_test, Y_test,  model, comm_round):
    mse = tf.keras.losses.MeanAbsoluteError()
    y_pred = model.predict(X_test)
    loss = mse(y_test,y_pred)
    print('global_loss:',loss.numpy())
    return loss
```

Fig. 28: Model testing function

4.2.2 Global model code

The Pythia oracle's operating code is described in this chapter. Some libraries should be imported before continuing with the code. This code will include all of the libraries that were implemented in the server code. However, an additional library called "import ipynb" will be imported. The notebook loader defined by this library enables the import of different ipynb files into the currently open ipynb file. This module will make it easier for the global model code to use the functions created in the server. Additionally, the character (*) will be inserted to make the server code accessible to the global model.

```
import numpy as np
import random
import cv2
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import SGD
from tensorflow.keras import backend as K

import import_ipynb
from Server_code import *
```

Fig. 29: Modules importation

The next step is to load the fixed dataset with the help of the pandas library.

df = pd.read_csv(r'Path where the CSV file is stored\File name.csv')

After that, the data must be divided into inputs X, which will be used to make predictions, and the target value Y, which will be forecasted based on the features. This was accomplished using the pandas library's .iloc function. The network's delay and bandwidth will serve as the project's target variables. Only the delay column will be the target in this case due to a few issues that will be discussed later in this chapter. All of the columns—aside from the delay column—are assigned to a variable called X. The delay column is represented by the y variable. The code to designate the columns as X and y is as follows:

```
X = df.iloc[:,1:10]
y = df.iloc[:,0]
```

In this code, each value before the comma represents the array's rows, and each value following the comma denotes the array's columns. Nothing was changed to the code before the comma and the X variable will take all of the rows from the dataset, therefore the rows were unaffected. The columns that are used are identified by the values that come after the comma. In this case, columns 1 through 10 were added to represent the features for the variable X (9 columns). The y variable's delay column was added as column 0. To begin training the model, the data must be divided into two sets: training and test. The train test split library was used in this stage.

```
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=1)
```

This code demonstrates how the data were divided into training and test sets, with the test set being 20% of the total data. Finally, these four variables are present in this phase:

- x_train (9 input features, 80% of the dataset)
- x_test (9 input features, 20% of the dataset)
- y_train (1 output, 80% of the dataset)
- y_test (1 output, 20% of the dataset)

Additionally, the train test split() function imports a random state hyperparameter that regulates the shuffling step prior to implementing the split. Since the random state in this instance is 1, the train test split will produce the same results for each execution. The X and y variables were then given new names, Features and Target lists, respectively, as a conclusion of the previous procedure.

```
Features_list = df.iloc[:,1:10]

Targets_list = df.iloc[:,0]
```

After this process, a new code line was added to create the clients based on the function made into the server code. This code line is the following:

```
clients = create_clients(X_train, y_train, num_clients=10, initial='client')
```

This function will be applied in the training set and it will create ten clients with specific names (client_1,client_2, etc.).

Then, the process of batching data created in the server code will be applied in this code in both the training and test set.

```
#process and batch the training data for each client
clients_batched = dict()
for (client_name, df) in clients.items():
    clients_batched[client_name] = batch_data(df)

#process and batch the test set
test_batched = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(len(y_test))
```

Fig. 30: Batching data process

The next step, before running the global model is to define an optimizer, a loss function and metrics that will be implemented in the global model initialisation.

```
comms_round = 100
#create optimizer
lr = 0.1
loss='mse'
metrics = ['accuracy']
optimizer = SGD(lr=lr,
                decay=lr / comms_round,
                momentum=0.9
               )
```

Fig. 31: Optimizer construction

Stochastic Gradient Descent (SGD) is the optimizer that will be used in this case. The loss function is the Mean Squared Error (MSE). In addition, the metric that will be used is accuracy. But there is also a new function that consists of the decay argument and the comms_round function. This function, it's simply the number of global epochs (aggregations) that will be running during training. So rather than decaying the learning rate with respect to the number of local epochs, in this case, will be decayed with respect to the number of global aggregations. With the addition of momentum, the gradient descent optimization process can overcome the oscillations of noisy gradients and ride across areas of the search space that are flat. This allows the search to develop inertia in a particular direction in the search space. [31]

The following step is to begin initialising the global model upon the completion of the previous process. Starting out, the global model was created with the input shape of (9), which is the number of feature columns that would be used and the number of classes as 1. In addition, the class that was created for the MLP in the server's code was imported. The outer loop's design came next, by collecting the global model's initialised weights first and then making a list to gather them after scaling. After that, a function that shuffle the client's dictionary in order to ensure randomness was created. The iteration through client training then began. A new model object was constructed and compiled for each client, and its initialization weights were set to the current global model parameters. This procedure is depicted in the next figure:

```
#initialize global model
smlp_global = SimpleMLP()
global_model = smlp_global.build(9, 1)

#commence global training loop
for comm_round in range(comms_round):

    # get the global model's weights - will serve as the initial weights for all local models
    global_weights = global_model.get_weights()

    #initial list to collect local model weights after scalling
    scaled_local_weight_list = list()

    #randomize client data - using keys
    client_names= list(clients_batched.keys())
    random.shuffle(client_names)

    #loop through each client and create new local model
    for client in client_names:
        smlp_local = SimpleMLP()
        local_model = smlp_local.build(9, 1)
        local_model.compile(loss=loss,
                    optimizer=optimizer,
                    metrics=metrics)

        #set local model weight to the weight of the global model
        local_model.set_weights(global_weights)
```

Fig. 32: Global model initialization

After that, the local model (client) completed one period of training.

```
#fit local model with client's data
local_model.fit(clients_batched[client], epochs=1, verbose=0)
```

Fig. 33: Data fitting

The new weights were scaled following training and added to the scaled local weight list. The local training is now complete.

```
#scale the model weights and add to list
scaling_factor = weight_scalling_factor(clients_batched, client)
scaled_weights = scale_model_weights(local_model.get_weights(), scaling_factor)
scaled_local_weight_list.append(scaled_weights)
```

Fig. 34: Scale model weights procedure

Furthermore, a small function was inserted after the local training operation to clear sessions and free memory after each communication round.

```
#clear session to free memory after each communication round
K.clear_session()
```

Fig. 35: Session clearing

Two more functions must be performed to complete a full global training epoch. The first function is to calculate the average weights across all the local models, which is done by summing the scaled weights. The global model must then be updated to include this new aggregate.

```
#get the average over all the local model
average_weights = sum_scaled_weights(scaled_local_weight_list)

#update global model
global_model.set_weights(average_weights)
```

Fig. 36: Model update

Testing and evaluating the model for accuracy and loss is the last step of the Pythia global model. Following the execution of the global model, the evaluation function generated the following:

```
local_model.evaluate(X_test, y_test)
```

```
94/94 [==============================] - 0s 1ms/step - loss: 27.1772 - accuracy: 0.3535
```

Fig. 37: Model evaluation

This picture shows that this model has a high loss and a low level of accuracy. To be more precise, the accuracy is a dismal 35%. In the discussion section, many approaches to enhancing accuracy and, perhaps, resolving this issue will be considered. Additionally, the server's test model function was employed to test the model and printed the same result with the evaluation function. The end conclusion is that each communication round's global model loss is close to 65% (64.64874%), on average.

```
#test global model and print out metrics after each communications round
for(X_test, Y_test) in test_batched:
    global_loss = test_model(X_test, y_test, global_model, comm_round)
```

```
94/94 [==============================] - 0s 1ms/step
global_loss: 64.64874
```

Fig. 38: Global Model testing

The predict function was also implemented. This function will predict the label of a new set of data, given a trained model. In contrast to the previous poor accuracy measurements, the network delay was correctly predicted by the model.

```
new_sample = [[55,18,1,2022,8,9,14,15,38]]
output = local_model.predict(new_sample)
print(output)
```

```
1/1 [==============================] - 0s 26ms/step
[[0.]]
```

Fig. 39: Global model prediction

4.3 Socket programming

This section of the project will describe a supplementary code that is made for the communication between Pythia oracle and other network nodes. Some nodes such as companies and enterprises will use this interface to learn about the network delay and bandwidth. This information will give guidelines on which network these nodes can use depending on the network latency and network traffic. The code was divided into two parts: the server and the client. The server will be the Pythia oracle that will bind a port and listen for any connections. The client part will be other nodes that will connect in the same port and they will ask for the information described above.

This code will consist of threads that as mentioned before, can give the solution of communication with more than one node at the same time. To insert threads into this system specific libraries are needed. These libraries are _thread module & threading module, these two are the most commonly used modules for multi-threading. The other library needed to create this interface is the socket module. Then, a lock object is created by the following function:

print_lock = threading.Lock()

Locks can be in one of two states: locked or unlocked. There are two fundamental methods: acquire() and release (). Print lock.acquire() and print lock.release() are used to convert the state from unlocked to locked and back. A new thread can be started and its identification is returned using the thread.start new thread() function. The function to call is given as the first argument, and the positional list of arguments is given as the second parameter in the form of a tuple.

After explaining the main functions implemented into a multi-threading socket programming, the next step is to make two functions one for the threads and one for the socket.

The socket function starts by inserting the host IP address and then reserves a port from the current computer that is implemented. To construct the socket two modules were used socket.AF_INET and socket.SOCK_STREAM. The type of addresses that the socket can connect with is designated by the address family AF_INET. [32] SOCK_STREAM offers two-way, sequential byte streams along with a stream data transfer method. [33] This socket type has out-of-band capabilities and reliably transfers data in the correct order. After that, the socket binded to the port and starts listening for any connections. A forever loop until the client exits were also created to establish connections with the clients and lock their state. Furthermore, the start_new_thread function mentioned before was utilized to start a new thread and return its identifier.

The thread function consists of a while loop that receives that data from the client, if it does not collect any data it will print ‘Bye.’ Then the lock is released and this function sends back the given string from the client but is reversed. Then the connection closes. The following figures illustrate the server code:

```python
# import socket programming library
import socket

# import thread module
from _thread import *
import threading

print_lock = threading.Lock()
```

Fig. 40: Socket and Threads libraries importation

```python
def Main():
    host = "127.0.0.1"

    # reserve a port on your computer
    # in our case it is 12345 but it
    # can be anything
    port = 2006
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("socket binded to port", port)

    # put the socket into listening mode
    s.listen(5)
    print("socket is listening")

    # a forever loop until client wants to exit
    while True:

        # establish connection with client
        c, addr = s.accept()

        # lock acquired by client
        print_lock.acquire()
        print('Connected to :', addr[0], ':', addr[1])

        # Start a new thread and return its identifier
        start_new_thread(threaded, (c,))
    s.close()


if __name__ == '__main__':
    Main()
```

Fig. 41: Multi-threaded server

```
# thread function
def threaded(c):
    while True:

        # data received from client
        data = c.recv(1024)
        if not data:
            print('Bye')

            # lock released on exit
            print_lock.release()
            break

        # reverse the given string from client
        data = data[::-1]

        # send back reversed string to client
        c.send(data)

    # connection closed
    c.close()
```

Fig. 42: Multi-threaded server (continued)

The next stage is the construction of client code. This code will contain the main function that will control the connection with the server. First, the socket module is imported. Then, the main function begins by defining the local host IP and the port that will be the same as the server one to achieve the connection between them. The same socket modules used before AF_INET and SOCK_STREAM will also be implemented into this code. After that, with the help of .connect() function, the client connects with the server by inserting the local host address and the port. The client then will its message to the server. In this case, the message is "Hello Pythia, I would like to know the network delay and bandwidth. "

Next, a while loop will be made to send the previous message and receive the reply from the server. The reply will be in the format “Received from the server : Hello, The network delay and bandwidth are ….” . In addition, the server will ask the client if he wants to continue after the previous communication. If the client wants to continue it can ask again about the network latency or traffic if it does not the connection closes. The next pictures show the client code:

```python
# Import socket module
import socket

def Main():
    # local host IP '127.0.0.1'
    host = '127.0.0.1'

    # Define the port on which you want to connect
    port = 2006

    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

    # connect to server on local computer
    s.connect((host,port))
```

Fig. 43: Client for Multi-threaded server

```python
    # message you send to server
    message = "Hello Pythia, I would like to know the network delay and bandwidth. "
    while True:

        # message sent to server
        s.send(message.encode('ascii'))

        # message received from server
        data = s.recv(1024)

        # print the received message
        # here it would be a reverse of sent message
        print('Received from the server : Hello, The network delay and bandwidth is ')

        # ask the client whether he wants to continue
        ans = input('\nDo you want to continue(y/n) :')
        if ans == 'y':
            continue
        else:
            break
    # close the connection
    s.close()

if __name__ == '__main__':
    Main()
```

Fig. 44: Client for Multi-threaded server (continued)

The results of the server and client are the following:

```
socket binded to port 2006
socket is listening
Connected to : 127.0.0.1 : 1163
Bye
Connected to : 127.0.0.1 : 1387
Bye
```

Fig. 45: Multi-threaded server results

```
Received from the server : Hello, The network delay and bandwidth are 0ms and 5Mbps

Do you want to continue(y/n) :y
Received from the server : Hello, The network delay and bandwidth are 0ms and 5Mbps

Do you want to continue(y/n) :n
```

Fig. 46: Client connection results

5. Discussion

Overall, the project performed effectively: the desired construction of Pythia's Federated network oracle was performed.

The following components were chosen for Pythia's federated learning system's design due to their reliability and simplicity:

- Python – Python was the programming language that was picked to create the Pythia oracle. Python is a strong, flexible, and all-purpose programming language. The fact that it is concise and simple to read makes it an excellent programming language. Python can perform any task. Python provides the stability, versatility and wide range of tools needed for a machine learning, federated learning, or artificial intelligence project.
- Anaconda Navigator – The Anaconda navigator was the environment that was used to run the python scripts. Without using command-line commands, Navigator enables to run typical Python programmes and quickly manage conda/FL packages, environments and channels.
- TensorFlow – TensorFlow is the main open-source library that is implemented to build the Pythia oracle. TensorFlow is an open-source, Python-compatible toolkit for numerical computation that accelerates and simplifies the creation of neural networks, machine learning and federated learning algorithms.

There were both benefits and drawbacks to this project. The created Pythia oracle worked successfully and in accordance with the hypothesis, which was its primary benefit. The system's high loss affected the model's ability to forecast outcomes accurately, which was a drawback. Another drawback was that the system became inaccurate after adding a second column to the project's target list. As a result, the target list was switched into either one or the other column to test the delay and bandwidth independently.

Previous studies based on federated learning systems offer some suggestions for enhancing the accuracy of this model. One of the options is to employ a different model framework, such as TensorFlow Federated or the Flower FL research framework. However, the current setup lacked the Linux operating system that these frameworks require in order to function. Create the model from scratch using some elementary libraries.

Other studies advise including non-i.i.d settings in the data to increase the trained models' accuracy. In this case, the dataset that was used to create the final model had i.i.d. information.

Another paper demonstrates how various algorithms, including Linear Regression, Support Vector Machines, Decision Trees and Random Forests, should be used to determine the optimal solution in terms of accuracy.

An issue that occurred is the process of IP addresses. Some research papers suggest the ipaddress module. This library helps in the creation, manipulation and operation of IPv4 and IPv6 addresses and networks. As result, it could be implemented to translate IP addresses into numbers instead of labels.

6. Conclusion and further work

While the socket interface is the only way to communicate with the network, applications still cannot inform the network of the data they are transmitting or inquire about the network's state. The evidence is clear: in order to improve the socket interface and users' experiences in terms of latency and communication, a protocol is needed that will enable network condition queries and communication with the network.

This study proposes the Pythia protocol, a federated network Oracle paradigm that improves the socket interface and adds a new feature to IP packets that allows network querying and information communicated to the network about what will be conveyed. This oracle employs a better and safer technique of learning data. With the use of packet sniffing tools like Wireshark, this new technique, which implements federated learning, can create a resilient and adaptable system that can accept requests and provide network information. With the use of federated learning, a new innovative world opens to Pythia oracle that could overcome the issues of privacy and security of other artificial intelligence technologies such as machine learning. Furthermore, the implementation of packet sniffing tools can help assess more parameters about a network. However, a multi-threading interface is also offered to increase its efficiency and ability to support several nodes. Although this approach would initially have some accuracy issues, from a long-term perspective, it would represent a significant step towards a limitless socket interface and a better user and business experience.

Further research is needed to determine the causes of accuracy issues or to define the best possible solutions for federated learning algorithms. Pythia’s system can further be improved by adding it to other existing FL frameworks. In addition, the Pythia model can use different federated learning algorithms such as FedProx, FSVRG or CO-OP to improve its accuracy.

To better understand the implications of Pythia’s results, future studies could address the importance or the use of non-iid data. This could develop more accurate FL models by a significant percentage and help to create a more efficient model.

Based on these conclusions, practitioners should also consider the challenges of the translation of IP addresses in federated learning models. Both IPV4 and IPV6 IP addresses should convert into numerical numbers, in this case labels. As a result, an improved version of Pythia should consider a specific function for converting IP addresses into integers and vice versa.

The results suggest that the Pythia federated network oracle can effectively operate and reply to network queries and provide information about the network delay and bandwidth. All parts of the system both server and global model code contribute to the construction of a steady Federated learning model that helps and assists users. The steps taken in this project demonstrate the detailed construction method of a mechanism that can be used to assist the daily user experience and make it better.

7. References

1. Burns E. Machine learning, TechTarget. 2020 [online] Available from: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML#:~:text=Machine%20learning%20(ML)%20is%20a,to%20predict%20new%20output%20values [Accessed 20 June 2022].

2. IBM. Machine Learning. 2020 [online] Available from: https://www.ibm.com/cloud/learn/machine-learning [Accessed 20 June 2022].

3. Peter Kairouz, H. Brendan McMahan, et al. Advances and Open Problems in Federated Learning. 2019 [online] Available from: https://arxiv.org/abs/1912.04977 [Accessed 20 June 2022].

4. Altexsoft. Federated Learning: The Shift from Centralized to Distributed On-Device Model Training. 2022 [online] Available from: https://www.altexsoft.com/blog/federated-learning/#:~:text=Federated%20learning%20or%20FL%20(sometimes,vehicles%20to%20IoT%20devices%2C%20etc [Accessed 20 June 2022].

5. Keith Bonawitz, Hubert Eichner, et al. Towards Federated Learning at Scale: System Design. 2019 [online] Available from: https://arxiv.org/pdf/1902.01046v2.pdf [Accessed 20 June 2022].

6. OpenMined. Understanding The Types Of Federated Learning. 2020 [online] Available from: https://blog.openmined.org/federated-learning-types/ [Accessed 20 June 2022].

7. Aledhari, Mohammed & Razzak, et al. (2020). Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Access. 8. 1-1. 10.1109/ACCESS.2020.3013541.

8. Dilmegani C. In-Depth Guide Into Secure Multi-Party Computation in 2022. 2022 [online] Available from: https://research.aimultiple.com/secure-multi-party-computation/ [Accessed 20 June 2022].

9. Brendan McMahan, Abhradeep Thakurta. Federated Learning with Formal Differential Privacy Guarantees. 2022 [online] Available from: https://ai.googleblog.com/2022/02/federated-learning-with-formal.html [Accessed 20 June 2022].

10. Bernard Marr. What Is Homomorphic Encryption? And Why Is It So Transformative? 2019 [online] Available from: https://www.forbes.com/sites/bernardmarr/2019/11/15/what-is-homomorphic-encryption-and-why-is-it-so-transformative/?sh=436a95747e93 [Accessed 20 June 2022].

11. Andrew Hard, Kanishka Rao, et al. Federated Learning For Mobile Keyboard Prediction. 2019 [online] Available from: https://arxiv.org/pdf/1811.03604v2.pdf [Accessed 20 June 2022].

12. Dilmegani C. What is Federated Learning (FL)? Techniques & Benefits in 2022. 2022 [online] Available from: https://research.aimultiple.com/federated-learning/ [Accessed 20 June 2022].

13. LearnVern. TCP Dump. 2020 [online] Available from: An explanation of how to use tcpdump from the command line on Linux. (learnvern.com) [Accessed 20 June 2022].

14. Apheris. Top 7 Open-Source Frameworks for Federated Learning. 2020 [online] Available from: https://www.apheris.com/blog-top7-open-source-frameworks-for-federated-learning [Accessed 20 June 2022].

15.ARMKEIL. BSD Socket. 2015 [online] Available from: https://www.keil.com/pack/doc/mw6/Network/html/using_network_sockets_bsd.html [Accessed 20 July 2022].

16. Lakshan P. Fundamentals of UDP Socket Programming in Java. 2020 [online] Available from: https://medium.com/javarevisited/fundamentals-of-udp-socket-programming-in-java-4a6972370592 [Accessed 20 July 2022].

17. Burns B. What Is Multithreading: A Guide to Multithreaded Applications. TotalView. 2020 [online] Available from: https://totalview.io/blog/multithreading-multithreaded-applications [Accessed 20 July 2022].

18. Shalamanov J. Behind the Creation of TensorFlow. Udacity. 2022 [online] Available from: https://www.udacity.com/blog/2022/08/behind-the-creation-of-tensorflow.html [Accessed 20 August 2022].

19. Tijani S. Federated Learning: A Step by Step Implementation in Tensorflow. 2020 [online] Available from: https://towardsdatascience.com/federated-learning-a-step-by-step-implementation-in-tensorflow-aac568283399 [Accessed 20 June 2022].

20. S. Abirami PC. Chapter Fourteen - Energy-efficient edge based real-time healthcare support system. 2020 [online] Available from: https://www.sciencedirect.com/science/article/pii/S0065245819300506 [Accessed 20 August 2022].

21. PEIXOTO F. A Simple overview of Multilayer Perceptron(MLP). 2020 [online] Available from: https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/ [Accessed 20 August 2022].

22. Shahrokhian D. Federated Learning with TensorFlow. 2021 [online] Available from: https://www.slideshare.net/DaniyalShahrokhianNo/federated-learning-with-tensorflow [Accessed 20 August 2022].

23. Cheong N. Design a federated learning system in seven steps. 2022 [online] Available from: https://www.integrate.ai/blog/design-a-federated-learning-system-in-seven-steps-pftl [Accessed 20 August 2022].

24. McKinney W. Pandas (software). 2018 [online] Available from: https://en.wikipedia.org/wiki/Pandas_(software) [Accessed 20 August 2022].

25. Oliphant T. NumPy. 2022 [online] Available from: https://en.wikipedia.org/wiki/NumPy [Accessed 20 August 2022].

26. scikit-learn. sklearn.preprocessing.LabelEncoder. 2022 [online] Available from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html [Accessed 20 August 2022].

27. Foundation PS. Datetime — Basic date and time types. 2022 [online] Available from: https://docs.python.org/3/library/datetime.html#module-datetime [Accessed 20 August 2022].

28. scikit-learn. sklearn.model_selection.train_test_split. 2022 [online] Available from: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [Accessed 20 August 2022].

29. TensorFlow. tf.keras.Sequential. 2022 [online] Available from: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential [Accessed 20 August 2022].

30. Keras. Dropout layer. 2022 [online] Available from: https://keras.io/api/layers/regularization_layers/dropout/ [Accessed 20 August 2022].

31. Brownlee J. Gradient Descent With Momentum from Scratch. 2021 [online] Available from: https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/ [Accessed 20 August 2022].

32. Braun M. What is AF_INET, and why do I need it? STACK OVERFLOW. 2021 [online] Available from: https://stackoverflow.com/questions/1593946/what-is-af-inet-and-why-do-i-need-it#:~:text=AF_INET%20is%20an%20address%20family,that%20type%20with%20the%20socket [Accessed 20 August 2022].

33. Boyd I. What is SOCK_DGRAM and SOCK_STREAM? STACK OVERFLOW. 2022 [online] Available from: https://stackoverflow.com/questions/5815675/what-is-sock-dgram-and-sock-stream [Accessed 20 August 2022].