Development of a Federated Learning Aggregation Algorithm

Mohammadreza Salarbashishahri

Thesis in the Department of Electrical and Computer Engineering

Concordia University

Montréal, Québec, Canada

June 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohammadreza Salarbashishahri**

Entitled: **Development of a Federated Learning Aggregation Algorithm**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair
Dr. Yan Liu

______________________ Examiner
Dr. Yang Wang

______________________ Supervisor
Dr. Jun Cai

Approved by ______________________
Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

__________ 2023

____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Development of a Federated Learning Aggregation Algorithm

Mohammadreza Salarbashishahri

The internet of things is made possible by the recent developments in communication and 5G networks, which enable real-time sensory data transfer between billions of devices. Raw data has no value until it is processed to extract its features. The features can be extracted using a machine learning technique, a common way to use these data. Federated learning (FL) is a platform that enables a group of clients to train a model cooperatively without disclosing their personal information. Traditional federated learning has issues such as data and model poisoning attacks, free-riding attacks, and model divergence caused by clients' non-independent and identically distributed (non-IID) datasets. Because the conventional federated averaging (FedAvg) aggregation algorithm in FL lacks an evaluation technique, it is unable to detect dishonest users or correct the global model's divergence. In this study, we suggest Shapley averaging (ShapAvg), a Shapley-based aggregation technique, to aggregate the global model by analyzing the models of the clients more effectively. Each client's weight in the weighted average under this approach will be proportionate to how much it contributed to the overall model performance. The results demonstrate that while employing non-IID datasets and in the presence of data poisoning or free-riding attacks, our suggested technique overperforms the FedAvg.

Acknowledgments

I would like to express my sincere gratitude to Dr. Jun Cai for his guidance and supports that assisted me to overcome the hurdles of my graduate studies path.

My appreciation extends to my parents and my wife as well who unconditionally supported me and nurtured my curiosity and ambition to follow my goals.

Contribution of Authors

This dissertation is the research work of Mohammadreza Salarbashishahri and it is submitted under supervision of Dr. Jun Cai for the degree of Master of Applied Science at Concordia University. The dissertation is original and unpublished.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the requisite background information to grasp the concepts and significance of federated learning and its challenges and research fields.

1.1 Federated learning

The data collected from sensors now present in millions of mobile devices with powerful computing capabilities could be utilized in crowdsourcing projects. These gadgets are linked to a network through the internet setting up the Internet of Things (IoT). With the 5G networks, IoT nodes may communicate real-time data for crowdsourcing applications more efficiently. Nevertheless, raw data without employing data analytics techniques will not generate any value. Using powerful machine learning (ML) algorithms to examine and discover the hidden properties of the data available on these devices is a frequent strategy for using the data from these devices.

1.1.1 From centralized to distributed approaches

Different approaches can be employed regarding machine learning training with the mentioned data. Figure 1.1 shows three different directions. In a centralized system, devices that own the data will send their dataset to a centralized cloud server. Subsequently, the cloud server will aggregate the data to form a training dataset and will train a machine learning model with this dataset. In this approach, the server has access to the private data of each client and will keep the trained model for future uses. As the whole dataset is merged in the server and the model will be trained with one comprehensive dataset, the training process will have a higher convergence speed. However, the server will lose the opportunity to use the dataset of participants who are unwilling to share their data. Also, it will impose a higher communication cost on the system as a large amount of data should be transferred from all clients to the server.

Another approach is edge learning, which utilizes some trusted edge servers close to mobile devices. Mobile devices will share their datasets with these edges as they trust them. Also, the data will be transferred in a more reliable network between these devices and edges close to them. The only required reliable communication links are between the edges and a cloud server. Using trusted edge servers can mitigate privacy and communicational cost problems. In this architecture, the data is sent to the trusted edge servers through local networks. This will decrease clients' communication costs and preserves clients' data privacy. Subsequently, edge servers will train a model with acquired data and send these models to a cloud server for the final aggregation.

Some machine learning (ML) applications employ user/device private data to train a model cooperatively, including traffic flow prediction [1], cancer prediction in healthcare [2], human digital twin [3], [4], and handwriting recognition [5]. Each mobile device offloads its data to the cloud server to facilitate the learning process in a centralized fashion [6]. However, this approach has drawbacks, including higher connection costs, decreased data privacy, and increased energy consumption in IoT devices with limited power. On the contrary to submitting the trained model to the cloud server, each device can distributively train the ML model using its data to get around these restrictions. Participants (i.e., devices) in such an architecture do not offload their private data to the server for the sake of learning.

Finally, federated learning (FL) is a technique that shifts the learning to the end devices or data owners. FL guarantees to keep data in the participants' storage since they only send the trained models' parameters, such as gradients, to an aggregation server. Therefore, a curious server cannot access or derive the private data from the trained model, and there is no need for a trusted edge. Figure 1.1 c) shows the direct communication between clients and the aggregation server without transferring the private dataset. FL was introduced first by Google in [7] using the distributed computation ideas described in [8] and [9] to employ it in the following word prediction of Gborad.

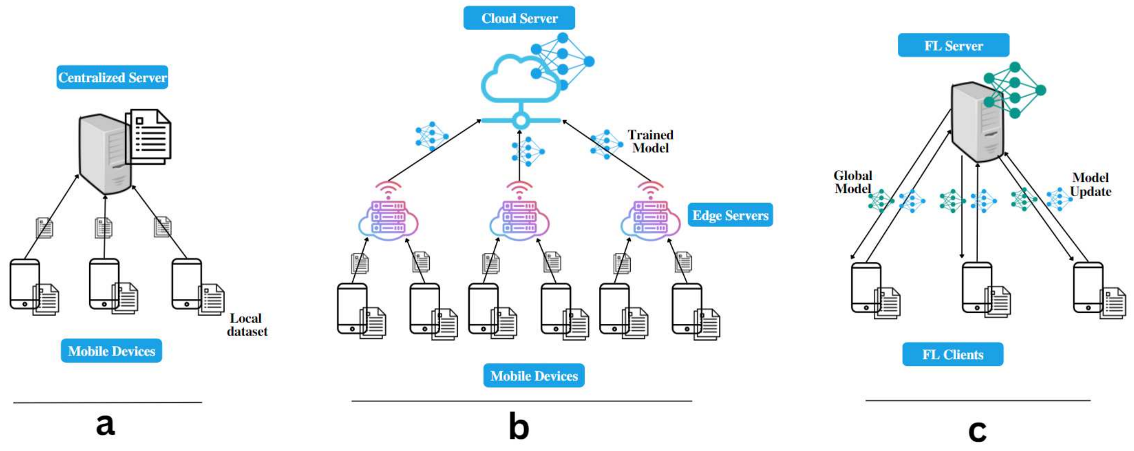


Figure 1.1: Different approaches in training ML models: a) Centralized learning b) Edge learning c) Federated learning

This technique allows several participants to train a global model using their data. In this privacy preserved approach, a server generates a global model with its constraints and sends it to some participants. Then the participants train the model using their data up to a certain level of accuracy defined in the constraints. Subsequently, the trained models are sent back to the server for aggregation into one global model again. This training sequence will continue until it converges to the desired accuracy.

1.1.2 Federated learning process

This section explains the steps of federated learning in detail. Overall, federated learning comprises three steps: 1- global model and constraints initialization, 2- local training, and 3- model aggregation. Among these steps, only the second step belongs to the participants, and the other two are done on the aggregation server side.

At first, the aggregation server generates a global model to be trained by the participants. These model's weights are defined by $W_{g,t}$, where *g* stands for the server's global model, and *t* is the federated learning training iteration. These weights could be randomly initialized weights or weights of a pre-trained model. In global model initialization, *t* equals zero. All required rules and parameters, such as learning rate, performance metrics, data quality requirement, and eligible participants, are defined in this step. Hence, this step is sometimes called "Task Initialization." After that, the server broadcasts the global model to the chosen participants.

Let K be the number of chosen participants for the local training step. After these participants receive the rules and the global model, they set $W_{k,t} = W_{g,t}$ and initiate their local training process. Each participant has a local dataset with n_k samples, where k is the index of the participating client. This dataset is usually made of the acquired data from the client's sensors or the typed words in the google Gboard example. There could be a constraint on the learning algorithms pre-defined by the server, or they can train the model by a user-specific algorithm such as stochastic Gradient Descent (SGD). We assume $x_{k,i}$, $y_{k,i}$, and $\hat{y}_{k,i}$ as the input and output of each dataset example, and the model estimate for input $x_{k,i}$, respectively, where i is the index of the example. Considering SGD and the least square function as the loss function, (1) defines the loss function for *i*'th example, and (2) defines the average loss function of client *k*, give the client's model weights at federated learning iteration *t*. Then each client computes g_k, the average gradients on its local data, defined by (3). Finally, with a learning rate of ε, the new model weights are calculated by (4). This training process may take a different time due to the heterogeneity of devices. Afterwards, each participant sends its new model weights, $W_{k,t+1}$, to the aggregation server.

$$f_{k,i}\left(W_{k,t}\right) = \left(\hat{y}_{k,i} - y_{k,i}\right)^2 \tag{1}$$

$$F_k\left(W_{k,t}\right) = \frac{1}{n_k} \sum_i f_{k,i}(W_{k,t}) \tag{2}$$

$$g_k = F_k(W_{k,t}) \tag{3}$$

$$W_{k,t+1} = W_{k,t} - \varepsilon g_k \tag{4}$$

Finally, the server aggregates all clients' trained models in an updated global model and sends it back to the participants for the subsequent iterations. Steps two and three will continue until a certain level of accuracy or convergence of the global model.

The aggregation server can employ various aggregation algorithms. [3] introduces Federated

Averaging (*FedAvg*) as a simple aggregation method. In this method, the aggregation server averages the received model weights from participants according to the following equation:

$$W_{g,t+1} = \sum_{k=1}^{K} \frac{n_k}{N} W_{k,t+1} \tag{5}$$

where $W_{g,t+1}$ is the updated global model weights, and N is the size of all datasets defined by equation (6), and K is the number of participants who have successfully sent their updated weights to the aggregation server.

$$N = \sum_{k=1}^{K} n_k \tag{6}$$

1.1.3 Pros and cons of federated learning

Besides the model architecture and learning algorithms, data is one of the crucial factors in machine learning. In a centralized approach, the cloud server loses the opportunity of acquiring the full information, as the clients might be reluctant to publish their local data. On the contrary, FL allows users to train the global model locally and send the trained model for aggregation. As a result, it raises the chance of employing a larger dataset, which preserves the participants' privacy. A comprehensive dataset diminishes the overfitting problem and can help the model generalize predictions since it is trained with a complete dataset. These datasets could include the acquired data from clients' sensors before each iteration. Hence, the whole dataset used for the training will be a sizeable real-time dataset.

Moreover, the trained model's size is less than the size of the data being used for training. In addition to the data size, the trained model could be compressed as it has a more straightforward format, such as matrices. Therefore, in FL, less communicational resource is needed to send the trained model, and network usage efficiency for the training process will significantly increase in limited bandwidth networks. Besides, since the model is stored locally in applications such as self-driving cars, it takes less time to use the model than cloud-based approaches. All in all, federated learning plays a significant role in real-time bandwidth-limited applications which use prediction models.

Distributed computing is gaining attention in recent research because of its advantages over centralized computing. Federated learning follows a similar architecture due to its essence. Participants share their computing power and data sources to form a final trained model. At each iteration, some participants, which the aggregation server will choose, replace the former ones. Overall, this architecture offers scalability, redundancy, capital expenditure, and performance improvement advantages.

Different participants can drop out or join the network to participate in the learning process in FL. The server will decide how many participants can receive the global model to train it by their data. Hence, scaling out the architecture is not challenging as the server demands more computational power as it can easily choose more participants from the pool. Also, if a participant fails during a learning iteration, the server will remove that device from the pool and join another one to the network. Therefore, it improves redundancy in the network as the server can immediately replace the defective participant. Also, these participants are training the global model voluntarily or at a low cost. Compared to the centralized learning approach that considerable capital should initially be invested in hardware, FL utilizes the economy of scale to make the learning process cost-efficient. As a result, there will be no need for significant capital expenditure.

On the other hand, the FL approach has some challenging disadvantages compared to the centralized approach. One aspect is related to the data stored by each participant. As the data belongs to different participants stored locally, the server has less governance over the datasets and their quality. Consequently, the general model could be trained with data irrelevant to the training's general goal, which does not generalize the model's outcome predictions. This also may cause overfitting to limited datasets. Also, having a sparse distribution of data from different participants' local data may hinder the model training's convergence or disrupt it. Overall, since the data is a black box for the server, there will be challenges in choosing the best participants regarding data quality.

Different devices, which are participating in the learning process, have non-identical communicational and computational power. The whole process of training the model includes updating the models' weights and sending them to the aggregation server. Each step's elapsed time depends on the number of CPU units, frequency of CPU units, allocated bandwidth, transmission channel parameters, etc. Besides, each participant's consumed energy depends on the size of their datasets and their processors' chip architecture. As these participants are heterogeneous, the learning process becomes more complex, and some techniques should be employed to eliminate the bottlenecks. On the other hand, in a centralized approach, the whole process of training is done locally on the cloud, computational power is pre-defined, and data transmission will be fast as it will be through a local network.

Another facet of FL is security since too many public participants could be effective in the global model's weights aggregation. As the training process is fully delegated to the participants, the server has less control over the model training procedure. One or multiple malfunctioning devices can update the global model to act in favour of those participants. For instance, some devices may train a recommendation system's model to recommend their desired product more frequently. Also, some participants may attack the system by training the model with false data, so the global model diverges. Conversely, the cloud employs pre-authenticated computational instances to form a trusted training workgroup in a centralized approach.

1.2 Current researches challenges and our objectives

The next chapter will provide more detail on federated learning research and practical challenges. In this section, we will briefly mention important issues to explain our research goals. Conventional federated learning uses a federated averaging (FedAvg) algorithm [8] to aggregate the clients' trained models. This algorithm assumes that the size of each client's dataset is known to the server to be used in a weighted average of the local models. This causes a security problem to leave such an important factor to clients.

Also, a data poisoning attack [10] is possible in conventional FL that a client may modify the dataset's characteristics, labels, model, etc. to affect the performance of the final model. Additionally, FedAvg allows for a free-riding attack [11], when customers refuse to contribute their data and computer resources to the training process.

Moreover, the datasets may be different across clients (non-IID). In this scenario, the gradients of the trained models from each participant will diverge after a certain number of cycles while aggregating to generate the global model.

The conventional methods lack an effective evaluation after receiving the clients' models. Our goal is to propose an effective evaluation method to measure the performance of each client during FL training. This will mitigate the dataset heterogeneity issues, such as non-IID datasets causing divergence and separating underperforming clients like malicious clients.

1.3 Contribution

We have employed the Shapley value (SV) [12] in FL to evaluate the performance of each client model. In this regard, SV helps separate the clients, causing a divergence in the global model due to non-IID datasets and identifying adversary clients that are not effectively improving the global model performance. Our work introduces a Shapley value-enhanced evaluation technique to evaluate the performance of each client during FL. The contributions of our research are summarized as follows:

- We propose an FL aggregation algorithm called Shapley averaging (ShapAvg), which evaluates the performance of each client's model based on SV. This algorithm generates a weight proportional to the client's contribution to the total performance gained in the global model.
- We proposed a model-based valuation instead of a training-based valuation to improve the efficiency of SV calculation in FL. Also, This algorithm actively removes adversary or underperforming users who may degrade the final results.
- By developing the ShapAvg solution in FL, we experimented and illustrated the improvements of our proposed algorithm in FL global model convergence. Our experiments show that ShapAvg outperforms FedAvg in the presence of both non-IID datasets and malicious clients such as poisoning attackers and free-riders.

A significant focus of this work's effort is on putting our aggregation algorithm into practice. We had several challenges during the development phases, which will be described in chapter 4. We created a framework to simulate our system model using the fewest internal and external possible dependencies, which is one crucial factor affecting performance metrics. In order to lower the overhead of external tools and improve control over all simulation aspects, a sizable chunk of the framework is created from scratch.

It is worth mentioning this work has resulted in the publication of the below research paper:

- Salarbashishahri et al. "A Shapley value-enhanced evaluation technique for effective aggregation in Federated Learning". In: *IEEE Future Networks World Forum* (2022)

1.4 Thesis Structure

The content of the thesis is organized as chapter 2 reviews the FL challenges and related research as the literature review. It is followed by chapter 3, which illustrates the system model and the proposed aggregation algorithm. Moreover, the mathematical equations in this work are explained in that chapter.

Chapter 4 represents the development process, employed technologies and related challenges. This chapter explains the codes, frameworks, and simulation infrastructure in detail. Chapter 5 provides the algorithm simulation results. In this chapter, the improvements compared to the conventional algorithms are demonstrated. Also, it mentions the simulation parameters and assumptions. Finally, this chapter concludes this work and its outcomes and proposes unsolved challenges, which could be potential future research.

Chapter 2

Literature Review

This chapter provides a detailed study of FL challenges and reviews some solutions proposed by recent research. It also mentions the result of a statistical survey we have done on research papers related to the FL challenges.

Authors in [13] categorize the FL challenges and research into four groups: 1- Communication cost, 2- Resource allocation, 3- Privacy and security issues, and 4- Applications of FL. Also, we reviewed the recent FL papers published since 2015 in IEEE with regards to the proposed categorization. Based on our review, we found that 45% of the papers were about resource allocation. Afterwards, 26% of them belonged to the communication cost. The third popular research field was applications of FL with 16%. And the last group was privacy and security issues, with 14%. It is worth mentioning that some of the research falls into two groups, as the challenges and proposed solutions might cover common areas. This might slightly change the survey result; nevertheless, this gives us an overall image of the popularity of the recent research. Figure 2.1 depicts the distribution of these papers.

In the following, we provide more explanations and examples of the conducted research in each category and subcategory.

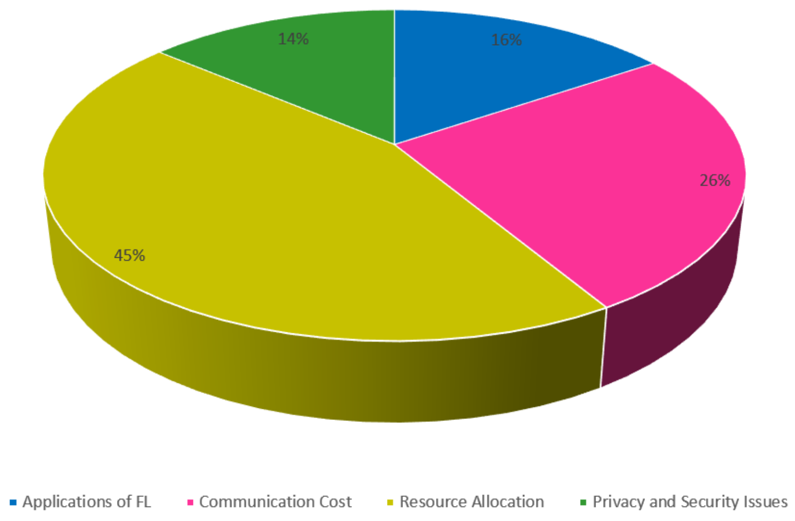


Figure 2.1: Distribution of FL research papers and their corresponding categories

2.1 Resource allocation

2.1.1 Computational resources

As mentioned in the architecture of federated learning, the computation of the learning process is offloaded to the participants. Typically, these participants are mobile or IoT devices with limited computational resources. Also, due to the significant number of participants, the different processing power of each device will form a heterogeneous group of participants. As a result, using a computational resource management strategy is vital. A computational resource management strategy will maximize the outcome and minimize resource utilization on the participants' side. Also, a computational resource management strategy could administer the heterogeneity of participants' computational resources in an aggregation server. Hence, a participant will not become the bottleneck of the whole iteration of training a model.

Different management strategies have been employed in federated learning computational resource management. The work in [14] employs reinforcement learning to determine the optimal mobile device resource management decisions since the decision parameters are dynamically changing. In this paper, the state space consists of the frequency and capacitance of participants' CPU cores. The action space comprises the number of data units and the server's energy demands from each participant. Finally, the reward function is proportional to the accumulated data and inversely proportional to the energy consumption and training latency. Applying reinforcement learning is advantageous since the aggregation server has slight information about the dynamic participants' environment. However, executing reinforcement learning algorithms will increase the complexity of the computation on the server's side.

2.1.2 Power consumption

As mentioned, the participants' pool mainly consists of power-limited mobile or IoT devices. As the power consumption increases for each iteration of local model training, the power-limited participants become more reluctant to participate in the training process. Also, another source of power consumption is due to the transmission between the model owner and participants. Since mobile devices are to minimize energy consumption, they will only be able to communicate with close devices.

The above reasons make power consumption management a critical factor in federated learning. This can limit the communicational and computational resources to diminish their power and form a Pareto-optimization problem. There are so many solutions for the Pareto optimization problem. Modelling the issue as a Stackelberg game is one of these solutions, explained in the next section.

Authors in [15] study the transmission of gradients to the edge server, considering a privacy constraint level. This study concludes that adding some noise actively to the local updates will improve privacy; however, it will increase the transmission power. Also, the required privacy could be obtained without affecting the learning performance until the privacy threshold is below a certain level, which decreases with the SNR.

2.1.3 Participant selection and mechanism design

Participant selection is a two-sided problem. On the aggregation server side, the server is willing to choose the best participants to maximize its outcome and minimize its cost. Similarly, the participants will minimize their costs and maximize their profit.

Its model accuracy measures the outcome of the server after the final aggregation and the time it has taken to train that model. In some cases, the server should also propose an incentive to encourage the participants to cooperate in the learning process. On the other hand, the participants' profit is the aggregation server's incentive or the trained model that it shares with the participants. The participants' cost consists of the resource they use, mentioned in the last three subsections.

This two-sided problem forms a Stackelberg game. In [16] Stackelberg game is defined as an economics model, which models a game with one or multiple leaders and followers. The leader takes the first action, then the followers' response by taking their action based on the leader's action. Once the leader has taken action, it cannot take it back. Besides, the leader knows the followers' reward structure, and the followers only know the leader's actions.

Equation (7) defines the followers' reward. Subscript 1 refers to the leader, and subscript 2 refers to the followers. Hence, the reward is the revenue minus cost. The revenue is the product of the price multiplied by the quantity the follower provides. Also, the cost of the follower is a function of the quantity it provides.

$$\pi_2 = P\left(q_1 + q_2\right).q_2 - C_2\left(q_2\right) \tag{7}$$

Subsequently, equation (8) defines the reward of the leader. The participant's quantity is a function of the leader's quantity. The leader's goal is to maximize (8), and the participant's goal is to maximize (7). Solving these two maximization problems will lead to an equilibrium that either maximizes the leader's profit and will maximize the participants' profit concerning the leader's.

$$\pi_1 = P\left(q_1 + q_2\left(q_1\right)\right).q_1 - C_1\left(q_1\right) \tag{8}$$

There are two approaches to employing the Stackelberg game in federated learning problems. Authors in [17] models the model owner as of the leader, which offers an incentive. Also, the mobile devices, which participate in the learning process, are defined as the followers who determine the optimum CPU power they will use. On the other hand, the work in [18] illustrates the participating mobile devices as the leaders who decide on the price for one unit of their training data. Also, the model owner is defined as the single follower, which determines the size of training data of each mobile device it needs.

Zhan *et al.* [19] define a mechanism to motivate the clients to participate in the training process.

In this method, the server is the leader, and participants are the followers. The amount of payment to each participant is proportional to the ratio of the data they use divided by the total data used in the training process. At first, the server defines its strategy by announcing the reward it is willing to pay regarding the mentioned data size ratio. Subsequently, each participant determines the size of data it is ready to use concerning the announced reward. Finally, a reinforcement learning approach is employed on the server and participants' sides to reach the optimality of received reward for each side.

Pandey *et al.* [20] define an incentive-based interaction between the server and participants. In this method, the server announces a reward rate at first. Then, participants will choose the accuracy they will offer concerning the advertised reward rate. Participants' reward will be proportional to the accuracy they offer multiplied by the reward rate. Finally, both server and participants are going to maximize their profit, which forms a Stackelberg game.

Kang *et al.* [21] introduce a reputation blockchain layer. In this method, task publishers will evaluate the quality of the locally trained model of each participant. Subsequently, these task publishers will publish their opinion called reputation about each participant by uploading it as a transaction to the blockchain. Consequently, other task publishers, or model owners, will combine their opinion with others to choose the best workers. Similarly, authors in [22] introduce a reputation for each participant, uploaded to the blockchain network after increasing or decreasing by one on each iteration.

Toyoda *et al.* [23] utilize blockchain for mechanism design. In this paper, a task is published through blockchain at first. Then, some participants will be chosen randomly to update the model. Subsequently, other participants will retrieve these updated models and vote for each of them based on their accuracy in the next rounds. Hence, each participant of the previous round will be rewarded based on its model ranking. These rewards are cryptocurrencies, which are published with the task by a smart contract.

Kim *et al.* [24] combine mechanism design and differential privacy. In this regard, the valuation function input includes the ε of differential privacy, and the participants' reward will be related to the differential privacy.

Ding *et al.* [25] separate different cases into three cases: complete information scenario, weakly incomplete information scenario, strongly incomplete information scenario. In the first case, the server has the complete information about the data type of the participants. Conversely, the last one is when the server only knows the distribution of the participants' data type. Then, it designs an optimum contract for each of these cases, in which the server and the participants could take the maximum profit.

Lee *et al.* [26] combine the energy consumption as a function of the number of CPU cycles and other factors and the amount of data each participant is willing to use to train the model. On the other hand, the server, which is called the coordinator in this paper, will reward the participants based on the performance of the trained models. In this regard, a quality function is defined primarily based on the model's performance, and the reward function uses this quality function.

In addition to rewarding in incentive mechanisms, the work in [27] defines a penalty policy. In this proposition, each participant deposits some blockchain token when subscribing. Subsequently, for each task, it defines a running time. By the end of that running time, if a participant fails to upload the trained model, or the uploaded trained model of a participant fails in verification, that participant will be penalized by deducting the amount of blockchain token it has deposited at first. This brings fairness to the designed incentive mechanism.

Table 2.1 summarizes the research in resource management category with their subcategory and key ideas.

Table 2.1: Key ideas of reviewed research papers in resource management

Challenge	Reference	Key idea
Computational resources	[14]	Employs reinforcement learning to determine optimal resource management decision.
Power consumption	[15]	Studies the effect of adding noise to the local updates on the power consumption.
Participant selection and mechanism design	[17], [18]	Models the model owner and mobile devices as leader and followers, and conversely.
	[19]	Defines an incentive mechanism in which the payment is proportional to the amount of used data in local training.
	[20], [26]	Defines an incentive mechanism in which the payment is proportional to the achieved accuracy or performance in local training.
	[21], [22], and [23]	Defines a reputation-based incentive mechanism. In this mechanism, participants will be chosen based on their gained reputation in the last training rounds.
	[24]	Defines an incentive mechanism in which the payment is proportional to the differential privacy.
	[25]	Defines the incentive mechanism contracts based on the model owner's knowledge about the participants' data type.
	[27]	Defines a penalty in addition to the reward. In this mechanism, if a participant fails in training, it will be penalized.

2.2 Communication cost

This category includes a broader area than just communication itself. Fields such as aggregation algorithms and data quality belong to this category as they affect the number of training rounds, which affects communication costs.

2.2.1 Communication resources

Communication of federated learning between the aggregation server and participants consists of 1- sending the created or aggregated model to the participants 2- sending the locally trained model to the aggregation server. These communication phases include data transmission by each participant individually. Compared to the centralized approach, federated learning increases the communication overhead significantly. Besides, mobile devices participating in the learning process are usually small devices with limited communicational resources. Consequently, the more the used communication bandwidth is managed, the more efficient federated learning will be.

Chen *et al.* [28] separate the model layer into shallow and deep layers. Shallow layers contain fewer parameters and learn general features of the dataset. Deep layers on the other hand, have more parameters and learn features related to specific datasets. With this regard, smaller number of parameters of the shallow layers have a more effect on the performance. In this method, the shallow layers will be updated more frequently than the deep layers. This will result in less communication overhead in each iteration. Another way to decrease the communicational overhead is to compress the gradients before sending. Despite most method, which only applies compression to the upstream communications from the client to the server, the work in [29] employs compression in the downstream transmission.

Luo *et al.* [30] utilizes edges to aggregate locally trained models. Subsequently, the edges will send the aggregated models to a cloud server for the final aggregation. In this architecture, all locally trained models are not needed to be sent to the cloud and edges are typically located near the participants. As a result, it will significantly decrease the transmission overhead and power consumption.

2.2.2 Aggregation algorithms

An essential difference between centralized and federated learning approaches is the need to aggregate the locally trained model in the model server. There is only one instance of a machine in centralized approaches that trains the model by the local data. On the other hand, in federated learning, the model owner sends a global structure of a shared model to some participants to be trained by their local data. Afterwards, each participant sends the trained model to the model owner. In this phase, the model owner will act as an aggregation server, which forms a final trained model based on the received trained models from the participants.

Obviously, the algorithm employed by the aggregation server plays a major role in the performance of training and the convergence time. FedAvg is the primary example of these algorithms. However, as the number of participants increase, it challenges the aggregation algorithm in terms of the prediction error of the aggregated model and the computational aggregation overhead.

Authors in [1] suggest an improvement to the FedAvg in traffic flow prediction. In this paper, the participants are grouped based on their spatio-temporal parameters. As a result, the correlation of the data results in a better convergence in each group's model. Finally, the aggregation server will choose the best model for the performance as the global model for the current iteration. This sequence will continue; therefore, other groups might form the global model in the successive iterations. It shows that the proposed method performs better than the FedAvg when the number of participants is greater than 8.

In each iteration of training the model, the aggregation server should wait until it receives models from all participants. Hence, the slowest participant limits the whole iteration of training the model. In addition to the fact that the slow participants become the system's bottleneck, if the transmission of the models encounters any problems or terminates unexpectedly, it will stall or hinder the training process. The work in [28] proposes an asynchronous approach, in which each trained model will participate in the aggregation of the global model as soon as it reaches the aggregation server. However, the gradients will be multiplied to weight proportional to the difference of current time and the time the gradient belongs to. In this case, the update of the global model will be based on an asynchronous important-based approach. This method is practical as long as the multiplied weight is chosen precisely; otherwise, slower participants could lead the global model to divergence since their trained models belongs to previous iterations and will be combined with new models.

In [31], Wu *et al.* propose a personalized federated learning architecture. This research employs federated transfer learning to utilize the parameters in the lower layers of the globally shared model. Then, each participant adds some customized higher layers to the shared model and trains it with a specific dataset. Hence, each participant could adapt the global model to its needs. This method will address the problem of model heterogeneity of participants.

Feng *et al.* [32] implement asynchronous federated learning in the blockchain. In this architecture, the local models participate in the aggregation to form the global model concerning the time difference and their model score. As for the asynchronous aggregation, an exponential function proportional to the freshness of the locally trained model is used. Also, the score of each model is proportional to the Euclidian distance between the local and global models. However, only using this Euclidian speed seems to have a drawback. A malicious user can send a similar model to the global model on each iteration to gain a better score without improving the accuracy.

Cahi *et al.* [33] propose a hierarchical blockchain-based federated learning architecture. In this architecture, the global model is trained at the first level; then, it will be sent to the higher levels. At the higher level, new participants will train this model with their new datasets. As a result, it will decrease the computational complexity of aggregating local models at each level. This approach will be suitable when the computational resource is limited at each level to aggregate all locally trained models. On the other hand, authors in [34] propose a cross-cluster blockchain-based federated learning architecture. In this paper, different participants are separated into different clusters. At first, these cluster members will train their models, which the consensus algorithm will aggregate. Subsequently, the aggregated model will be sent to other clusters to be trained by their datasets. Unlike previous architecture, these clusters will work together in this paper instead of working for the superior blockchain level. Similarly, in [35], authors define a collaborative approach between roadside units (RSU), which will aggregate the trained models of vehicles and share them with other RSUs in the blockchain platform.

2.2.3 Data quality

Federated learning is dependent on the local data of the participants. The more comprehensive the local datasets are, the more precise the model predictions will be. Compared to the centralized approach, where the whole data is present at the central server, information is distributed among participants in federated learning. Also, the model owner does not know the quality and statistical distribution of the participants' datasets. In the case of non-iid datasets, each participant's trained model's gradients will diverge after some iterations while aggregating to form the global trained model. This will result in accuracy degradation as [36] showed that the model's accuracy could decrease up to 55% using non-iid datasets. Besides, the convergence time will increase, and the global model predictions will not be generalized if the whole dataset comprising each participants' dataset does not form sufficient non-iid datasets. The work in [8] shows that the test accuracy of the trained model will either be less while using non-iid datasets or will need more training rounds to reach approximate equal accuracy.

With this in mind, finding a solution to gain knowledge about participants' private datasets and developing data management solutions are challenges of federated learning. Authors in [36] propose a method to improve the accuracy of the trained model prediction. This research separates the data into two groups of shared and private data. Shared data is stored on the aggregation server, and a portion will be sent to each participant. Then, each participant will train the model locally with that portion of shared data combined with the local data. It claims that the accuracy will increase by 30

Chiu *et al.* [37] propose an algorithm that the participants swap the locally trained models between the iterations before being aggregated by the server. In this case, the aggregation server switches the models between the participants instead of aggregating them in every round. After some rounds, received models will be aggregated and sent to the participants as the final trained model.

Zhang *et al.* [38] define a measure for the divergence of locally trained models' gradients. In this case, the server store a dataset called auxiliary data. At first, the server chooses some participants randomly. Subsequently, selects participants and the server train the model simultaneously by their local data. Then, using the divergence between the model trained by the server and the participants, the server will choose participants for the next round. These participants will be selected among those who have a minor divergence.

Similarly, the work in [39] uses the deviation of the model parameters from the average parameters. In the aggregation phase, each parameter will be multiplied to a weight. This weight is adversely proportional to the mentioned deviation. As a result, locally trained models with higher deviations will have a more negligible effect in the training of the global model.

Table 2.2 summarizes the research in communication resources category with their subcategory and key ideas.

Table 2.2: Key ideas of reviewed research papers in communication resources

Challenge	Reference	Key idea
Communication resources	[28]	Employs reinforcement learning to determine optimal resource management decision.
	[29]	Compresses both up and downstream.
	[30]	Employs edges for pre-aggregation.
Aggregation algorithms	[1]	Implements grouped aggregation based on spatio-temporal parameters.
	[28]	Develops asynchronous aggregation.
	[31]	Proposes personalized federated learning by employing federated transfer learning.
	[32]	Develops synchronous federated learning aggregation in the blockchain.
	[33]	Develops hierarchical blockchain-based federated learning aggregation architecture.
	[34]	Develops cross-cluster blockchain-based federated learning aggregation architecture.
	[35]	Implements collaborative aggregation between roadside units (RSU).
Data quality	[36]	Suggests keeping a shared dataset at the server and sending private portions to the participants.
	[37]	Proposes swapping the locally trained models between the participants.
	[38]	Implements selective aggregation by measuring the accuracy with auxiliary data.
	[39]	Uses the deviation of the model parameters from the average parameters for a weighted aggregation.

2.3 Privacy and security issues

When the communication becomes public, and the transmission includes private data, privacy and security become critical issues. The training process includes using participants' personal data and sending the trained model to the aggregation server in federated learning. As a result, the privacy and security in federated learning separate the issues into two groups: 1- authentication & authorization 2- privacy of participants' data. The former matters relate to the verification of participants. The latter relates to preserving the data privacy of participants, so no malicious user, including a curious aggregation server, can get access to the private data of participants.

2.3.1 Authentication & authorization

Federated learning is based on a primary trust between the aggregation server and some participants. In this regard, the received trained model from participants is considered to be trained with real data. Otherwise, malicious participants could deviate the whole model training process from its primary goal. This act is called poisoning in federated learning.

There are two types of poisoning attacks: 1- model poisoning attack 2- data poisoning attack. In a model poisoning attack, some malicious participants try to corrupt the global model by sending false gradients as the trained model. This attack can affect the convergence and the accuracy of the global trained model. In a data poisoning attack, one or multiple malicious users try to train the model with false data. These two attacks could be more destructive if the number of malicious users is enough to influence the FedAvg result significantly.

Moreover, a malicious device can fake the model and identity of the server to extract users' information. Authentication methods are to help verify the identity of users and the aggregation server. VerifyNet [40] is a framework that preserves users' data privacy and verifies the correctness of the aggregated results of the server on the clients' side by employing encryption. In this case, a malicious device cannot either forge the server aggregated model or infer users' private information.

Some participants might be reluctant to participate in the training process. Therefore, they should be motivated by some incentives. In some cases, obtaining a global trained model is the incentive between participants who want to collaborate to receive the global trained model. Consequently, all participants should collaborate honestly in the training process to be fair. In this regard, another security issue is called the free-riding attack. In this case, a participant exploits the trained model without participating in training and using its dataset.

2.3.2 Participants' data privacy

One of the most goals in federated learning applications is to preserve the users' privacy, if not the only one. The model owner needs to exploit users' data to train its model with recent real-time data derived from users' devices. On the other hand, multiple users are not willing to share their private data with the server. In this case, federated learning is a median solution, which can exploit the users' data and not share it simultaneously. Hence, keeping that privacy becomes a critical concern. The work in [41] has proved that important information about the dataset used to train a model could be obtained. It has used a classifier to extract crucial and valuable data about the training dataset of another trained classifier in greater detail. Hence, a curious server or a middle man could employ similar techniques to infer meaningful information and features about participants' private data.

Shen *et al.* [42] use an auxiliary dataset to train an attack model. Subsequently, this trained model is used to infer the properties of participants' data. Also, it defines different strategies to choose participants and separate them into groups to infer the properties of their datasets more precisely or more efficiently.

There are different approaches toward preserving the data privacy of the clients. Authors in [43] use the differential privacy platform by employing a Gaussian noise mechanism. In this method, after computing the gradients, it will add Gaussian noise to these gradients. In this case, extracting data from the trained model will be more challenging. However, this method comes at the cost of increasing the testing loss of the trained model.

2.4 Applications of federated learning

Federated learning allows mobile devices to train a model locally with their data. Using users' data to learn a model is facing more limitations with the rising privacy rules. Hence, federated learning could be applied to every problem, including using users' private data to train a model. In this case, the privacy is preserved, and the model is trained with the most recent real-time data generated by active users.

The work in [1] utilizes federated learning in traffic flow prediction. In this case, different organizations such as Uber, car rental companies, Hellobike, etc., have access to the spatio-temporal of the users. However, this data is not sufficient for real-time traffic flow forecasting, traffic management, car sharing, and personal travel applications. Moreover, they cannot share this data with other organizations since it contains sensitive users' data. Therefore, a federated learning approach is implemented to train a shared model locally by each organization.

Mowla *et al.* [44] employ federated learning in jamming defence in flying ad-hoc networks (FANET). Due to high mobility in FANET, a centralized knowledge base cannot predict the presence of a jammer. Hence, the UAVs participate in training a federated Q-learning table with their sensory data such as signal strength indicator (RSSI) and packet delivery rate (PDR). This model-free strategy will be updated in each iteration by mobile UAVs to predict the presence of a jammer in new environments.

Lim *et al.* [45] apply federated learning to the model owners who have basic information of workers, but this data cannot form a precise model. In this paper, different model owners create a federation to train a shared model. Finally, the federation could sell this trained model by profit. Also, each model owner will reward workers for the quality of information they share with the model owner.

Zhao *et al.* [22] employ federated learning in the home appliances feedback network. In this application, customers participate in training the model created by the manufacturer. In this case, the manufacturer will learn to predict the customers' consumption behaviours and gain data on improving home appliances.

Authors in [46] utilized the federated learning in autonomous vehicles. Since vehicles are moving most of the time, they might lose connectivity in a fraction of time. As a result, using a local model to make intelligent decisions will be a safer and more reliable solution for autonomous vehicles. Also, this paper uses blockchain to reward users for motivating them to participate in the learning process. This idea shapes the loyalty of users.

Thw work in [47] uses federated learning in health care. In this project, data is acquired from body sensors such as heart rate, temperature, EEG sensors. Afterwards, a client will send this data to an organization. These organizations use the private data of patients to train a global model. Finally, the trained model can predict whether the patient is infected by covid-19. The whole process of publishing and aggregation of the model is executed by blockchain and smart contracts.

Cui *et al.* [48] employs federated learning in edge node caching systems. In this system, a model will be trained to predict the demanding files to increase the hit rate of the caching system. Since the data used to train the model is users' private data, federated learning will preserve the users' data privacy. Also, the whole process is implemented by blockchain to tackle the security concerns of the transmitted data.

Pokhrel *et al.* [49] propose employing the federated learning between UAVs. Then, it applies the same architecture to the LEO satellites. Also, the communication and sharing of the model are done through blockchain. Unlike typical blockchain systems, in this system, wireless mobile nodes, such as UAVs can act as blockchain miners.

Authors in [50] utilize federated learning to detect the infection of COVID-19 from the CT pictures. In this paper, multiple hospitals collaboratively train a global model to predict the disease from CT images. In this regard, data privacy is preserved. Also, all hospitals requests and training process is executed in blockchain.

Figure 2.2 summarizes the challenges and potential solutions offered by novel researches. The inner circle categorizes the challenges into four groups: 1- Resource allocation 2- Communication cost 3- Privacy and Security and 4- Applications. The next circle includes challenging fields and parameters of each group. Finally, the outer circle contains the potential solutions proposed by recent researches.

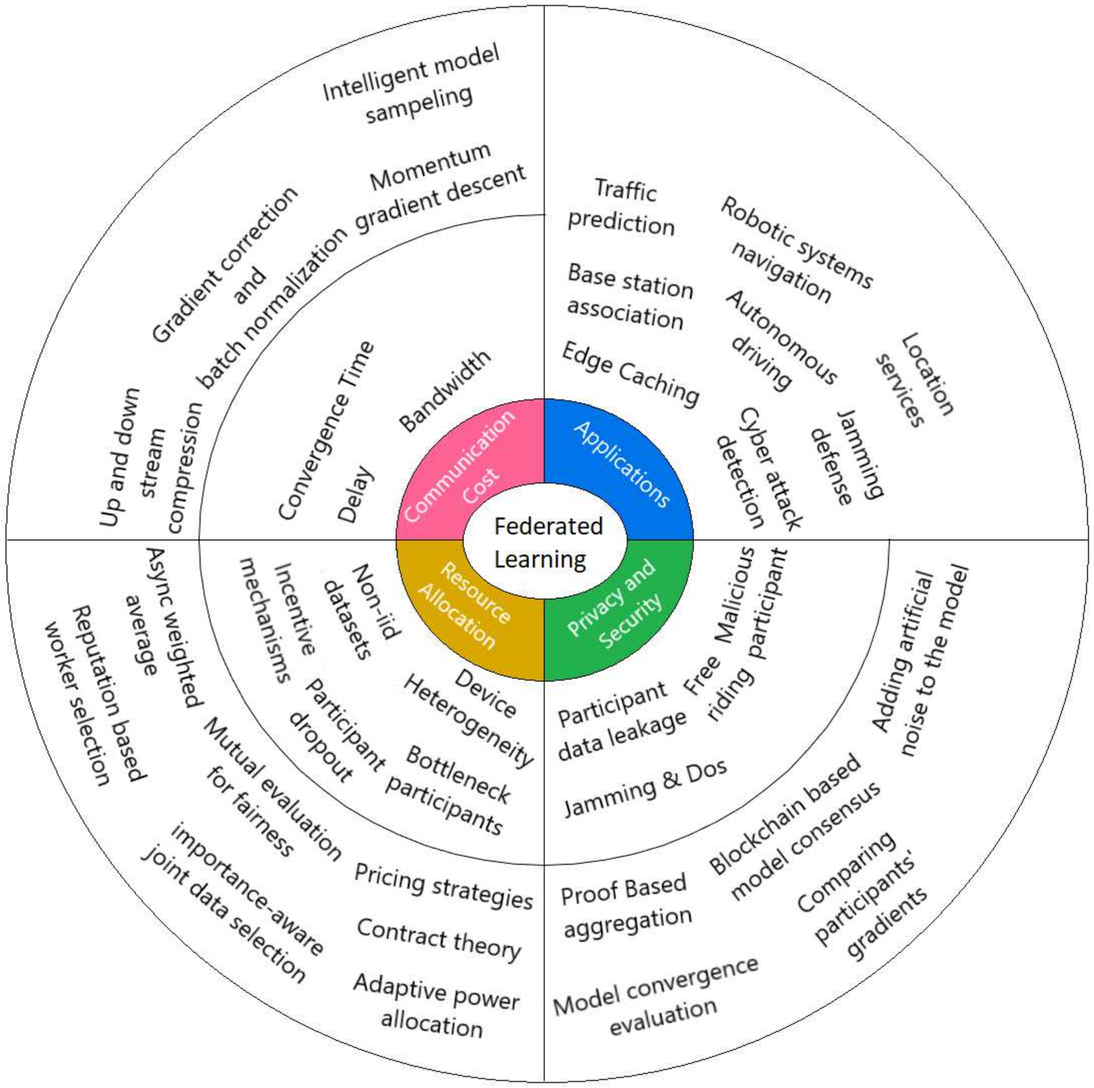


Figure 2.2: Challenges and potential solutions of federated learning

Chapter 3

ShapAvg Algorithm

In this chapter, we explain our proposed algorithm (ShapAvg) and its contributions. We begin by mentioning the current issues of the conventional method, FedAvg, by describing its mathematical process and the parts that will cause the problems. Then, we continue with the requirements of an effective aggregation algorithm. This will lead us to our proposed algorithm, which satisfies the requirements.

3.1 Current challenges in conventional FL aggregation

In this section, we briefly describe the FedAvg [8] equations to explain the issues. Consider an FL system with K participating clients, as shown in Fig. 3.1. Each client has a local data set D_i with data samples (x_j, y_j) and dataset size of $|D_i|$, where $i \in \{1, 2, \ldots, K\}$ and $j \in \{1, 2, \ldots, J\}$. Considering a loss function $\mathcal{L}(x_j, y_j, \omega)$ on a sample (x_j, y_j) with model ω, FL consists of three steps. First, the FL server will create a global model Ω^t, where t denotes the iteration, and sends it to all participating clients. Subsequently, each client i will set its local model ω_i^t to Ω^t and train it with local data set for multiple epochs. The clients' average loss is calculated as:

$$l_i\left(D_i, \omega_i^t\right) = \frac{1}{|D_i|} \sum_{j=1}^{J} \mathcal{L}\left(x_j, y_j, \omega_i^t\right); \quad (x_j, y_j) \in D_i, \tag{9}$$

the gradients are given as:

$$\Delta_i = \nabla l_i, \tag{10}$$

and the new local model will be calculated by:

$$\omega_i^{t+1} = \omega_i^t - \eta \Delta_i, \tag{11}$$

where η is the learning rate. Then, it sends the new local model to the server. Finally, the FL server will aggregate all received models into one global model based on FedAvg [8] according to:

$$\Omega^{t+1} = \frac{1}{|D|} \sum_{i=1}^{K} |D_i| \, \omega_i^{t+1}, \tag{12}$$

where $|D| = \sum_{i=1}^{K} |D_i|$. These steps continue until the specific performance metric set by the server is met.

As we can see in 12, this algorithm assumes that the size of each client's dataset is known to the server. Then the aggregated model will be a weighted average of the local models proportional to their dataset sizes. When clients announce their dataset sizes voluntarily, it could lead to security problems since a malicious user will report a large dataset size to change the global model effectively. Considering that the size of the dataset can logarithmically improve the model's performance [51], leaving such an important factor to clients' self-report seems to be not empirical. Also, one essential advantage of FL is its ability to preserve clients' privacy since the server does not have any information about the clients' datasets.

As a result of the FL platform's distributed and open design, malicious users may take part in training and control their local models before uploading them to the server to influence the performance of the global model. A data poisoning attack is an instance; [10]. A client may modify the dataset's characteristics, labels, model, etc., in a data poisoning attack to affect the performance of the final model. When malicious users declare a big dataset size in order to receive a greater weight in the global model weighted average aggregation, the negative effect worsens. It is impossible to

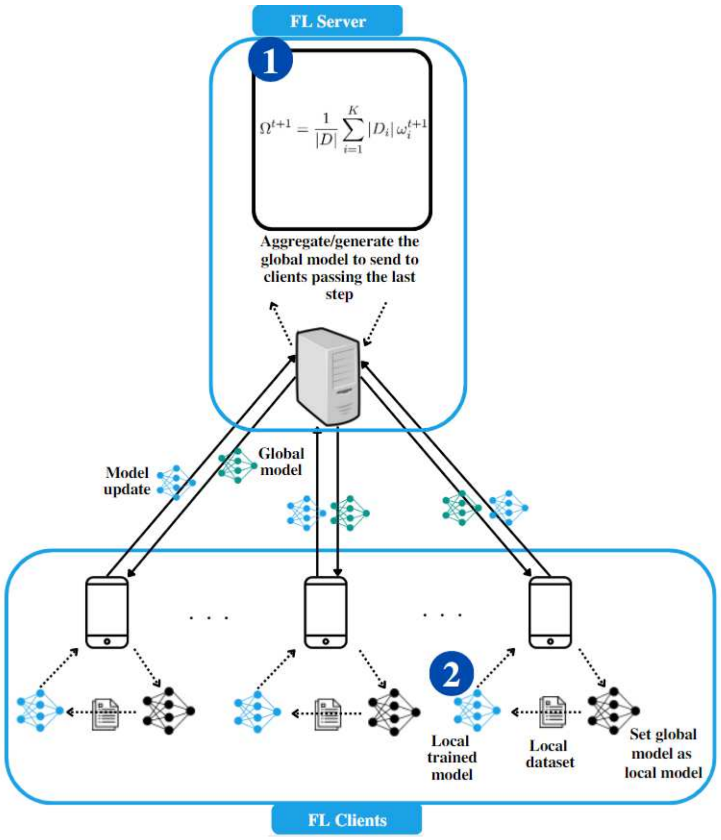

FL Server
1
$\Omega^{t+1} = \frac{1}{|D|}\sum_{i=1}^{K}|D_i|\,\omega_i^{t+1}$
Aggregate/generate the global model to send to clients passing the last step
Model update
Global model
2
Local trained model
Local dataset
Set global model as local model
FL Clients

Figure 3.1: Conventional FL architecture

recognize these malicious clients at the aggregation step due to the lack of performance evaluation of client models.

Additionally, FedAvg [10] allows for the possibility of a free-riding attack. When clients refuse to contribute their data and computer resources to the training process, a free-riding attack takes place. These clients will attempt to upload false updates to the server for their local models. In this scenario, they won't have to use any of their resources [11] in order to obtain the most recent edition of the global model, which has a high commercial value. Free-riders will not only profit from the free global model but also will slow down the convergence pace because their faulty models will impair the performance of the final model. Due to the lack of an evaluation technique, it is not possible to identify adversarial clients. Additionally, the datasets of various clients may be different across clients (non-IID). In this scenario, the gradients of the trained models from each participant will diverge after a certain number of cycles while aggregating to generate the global model. The study in [36] demonstrated that the model accuracy might decline by up to 55% when using non-IID datasets, leading to accuracy degradation.

As mentioned before, the conventional methods lack an effective evaluation after receiving the clients' models. Our algorithm proposes an effective evaluation method to measure the performance of each client during FL training. This will mitigate the dataset heterogeneity issues, such as non-IID datasets causing divergence and separating underperforming clients like malicious clients.

3.2 Requirements of an effective aggregation algorithm

The necessity of adding an evaluation method to the aggregation algorithm is described in the previous section. However, this evaluation should meet some requirements to be effective. As mentioned, the dataset size D_i is announced by participating clients, allowing them to manipulate the learning process. Hence, we need an evaluation method with three responsibilities: 1) fairly evaluate the contribution of each client in the learning process, 2) generate corresponding weights for the global model aggregation according to each client's contribution to the final performance; and 3) remove malicious client or clients with datasets which cause divergence.

We assume $\varphi_i^t(\omega_i^t)$ is the evaluation function for a client i at iteration t, which uses the ith client model to generate a value relative to its performance. Also, let S be a subset of participating clients and ω_S be the global model aggregating these clients' trained models. Then we expect the evaluation function to satisfy the following requirement to be effective:

- The value generated for the model of the client subset should be distributed among those clients, i.e., $\varphi_S^t\left(\omega_S^t\right) = \sum_{i\in S}\varphi_i^t(\omega_i^t)$. This is called group rationality which satisfies the linearity property. In this regard, for two clients, u and v, we have:

$$\varphi_{u\cup v}^t\left(\omega_{u\cup v}^t\right) = \varphi_u^t\left(\omega_u^t\right) + \varphi_v^t(\omega_v^t),$$

  where $\varphi_{u\cup v}^t\left(\omega_{u\cup v}^t\right)$ is the evaluation function for the aggregated model of u and v clients.

- A participating client with no effect on the global model performance or a malicious user with a negative or negligible impact should receive a zero value. Formally, if $\varphi_{S\cup u}^t\left(\omega_{S\cup u}^t\right) \le \varphi_S^t\left(\omega_S^t\right)$, then $\varphi_u^t = 0$.

- Two clients with the same contribution to the global model performance should receive an equal value. In other words, if clients u and v are equal in the value they add to a subset S value $\varphi_{S\cup u}^t\left(\omega_{S\cup u}^t\right) = \varphi_{S\cup v}^t\left(\omega_{S\cup v}^t\right)$, then they will receive an exact same value: $\varphi_u^t\left(\omega_u^t\right) = \varphi_v^t\left(\omega_v^t\right)$. This property is called fairness.

3.3 Proposed system model

Except for assigning zero value to damaging or underperforming clients, it has been proved that the SV meets all requirements [12]. Employing SV in FL, a client's SV will be the client's average marginal contribution to the global model performance trained by different subsets of clients. Fig. 3.2 depicts the overall picture of the proposed system architecture with the steps we have added to the conventional FL. If we assume I is the coalition of all participating clients, we propose SV for a client i at iteration t in our system as:

$$\varphi_i^t = \frac{1}{K}\sum_{S\subseteq I\setminus\{i\}}\binom{K-1}{|S|}^{-1}\left[V\left(\Omega_{S\bigcup i}^t\right) - V\left(\Omega_S^t\right)\right]. \tag{13}$$

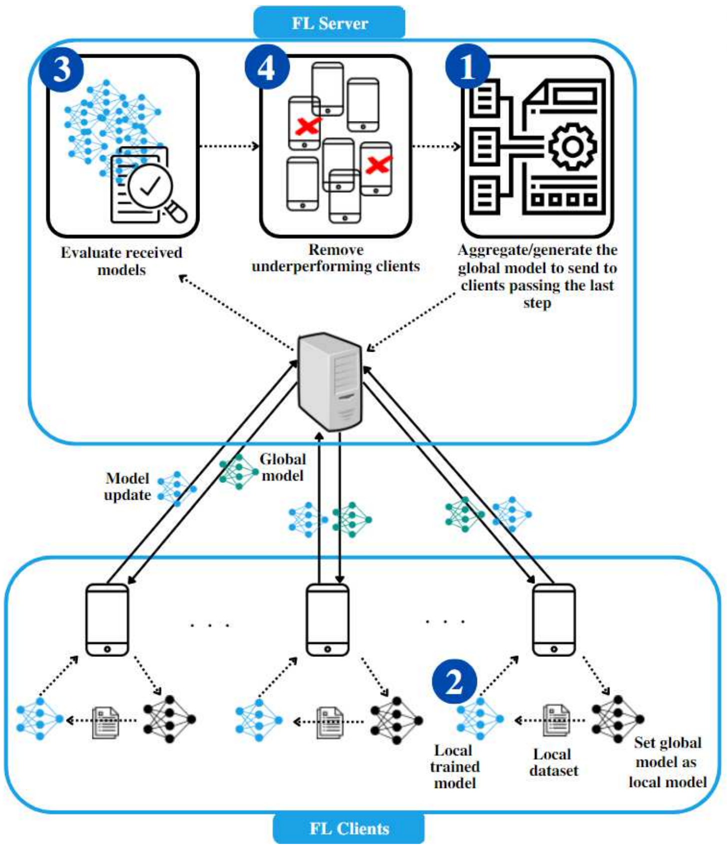

FL Server
3
4
1
Evaluate received models
Remove underperforming clients
Aggregate/generate the global model to send to clients passing the last step
Model update
Global model
2
Local trained model
Local dataset
Set global model as local model
FL Clients

Figure 3.2: Proposed system architecture

The expression in (13) calculates the average contribution of the client i's model to all models trained by all possible combinations of clients. The parameter V denotes a valuation function which takes the aggregated model Ω_S of a set S of clients and returns a corresponding value.

However, calculating the Shapley value requires retraining the model by the members of different combinations of clients. This imposes a high computation complexity on the system. Also, it can significantly increase the delay and communication cost since the models should be transferred multiple times between the server and clients. Hence, we propose a model-based valuation in ShapAvg instead of a training-based valuation. In this regard, the server will keep a record of clients' models. Afterwards, it will generate the global model for different combinations of clients for the valuation function. Therefore, the need to retrain the models and massive communication will be removed. We take the average of the client's models to generate the value for the clients' coalition model as:

$$V\left(\Omega_S\right) = V\left(\frac{1}{|S|}\sum_{i\subseteq S}\omega_i\right), \tag{14}$$

where $|S|$ is the number of clients in that coalition.

ShapAvg assesses the contribution by excluding the desired client from the coalition and measuring the impact. The measured performance will increase when a malicious client with a large announced dataset and a managed model is excluded. Consequently, it will detect the difference if a client tries manipulating the learning process by intentionally sending a managed model and announcing a large dataset. It will be robust to this issue compared to FedAvg.

Subsequently, we want to replace the weights in FedAvg, so that the impact of each client on the global model performance is proportional to their effectiveness. For this purpose, we utilize the calculated contribution value to assign an appropriate weight. We used the categorical cross-entropy loss as the valuation function V to represent the convergence speed of the global model. It efficiently shows the distance between the desired state and the output of the global model. This loss function is defined in (15), where t_i is the truth label, $\Omega(d_k)$ is the output vector of model for a given input training data sample, and p_i is a softmax function defined as in (16).

$$V\left(\Omega\right) = -\frac{1}{N}\left[\sum_{k=1}^{N}\sum_{i=1}^{n} t_i \log\left(p_i\left(\Omega(d_k)\right)\right)\right], \quad \text{for n classes and N data samples} \tag{15}$$

$$p_i\left(\Omega(d_k)\right) = \frac{e^{\Omega(d_k)_i}}{\sum_{j=1}^{n} e^{\Omega(d_k)_j}} \tag{16}$$

The model $\Omega(d_k)$ takes d_k, the training dataset sample, as the input and generates a vector. Since we need the probability of belonging to a class, we need to change the range of the vector output to between 0 and 1 as a probability. Hence, we use the softmax function. This function, (16), calculates the probability by dividing the vector elements by the sum of elements. The higher the belonging probability of the desired class after softmax is, the lower the logarithm of it will be to decrease the loss value. Also, t is the desired softmax output vector derived from the dataset labels. The corresponding element of this vector of which class the input belongs is equal to 1, and others are equal to 0. Finally, it takes the loss average for all training data samples as the loss function value for the given model and training dataset.

The loss function needs a model and a test dataset, as in (9). Since this calculation is done on the FL server, it needs a sample dataset to evaluate clients' models. It also estimates the global model performance metric at each iteration to stop the learning process after meeting the desired value. Typically, a model which performs better will receive a lower loss value. However, we need to assign a higher weight to those clients outperforming others. If we assume the contribution value for a client i as φ_i^t, the average and standard deviation of clients' contribution values, $\overline{\varphi}^t$ and σ^t, respectively, all at iteration t, then λ_i^t will be the weight for that client in the aggregation. The calculation of λ_i^t will be shown in the following.

First, ShapAvg sets new contribution values, $\varphi_i^{t'}$, by comparing the difference from the average and the standard deviation as:

$$\varphi_i^{t'} = \begin{cases} 0, & \varphi_i^t - \overline{\varphi}^t > \sigma^t \\ \varphi_i^t, & \varphi_i^t - \overline{\varphi}^t \leq \sigma^t \end{cases}. \tag{17}$$

This process repeats at each iteration to remove the clients who are significantly or intentionally underperforming other clients. Also, the FL server will not send the global model to those clients with $\varphi_i^{t-1'} = 0$, which indicates free-riders with no proper contribution in previous iterations. This guarantees that clients will receive the newest model only after they constructively participate in the learning process. These clients can receive the latest model as soon as their contribution meets the requirement by (17) to receive a contribution value of more than zero. Subsequently, a weight for each client passing the last step will be generated as:

$$\lambda_i^t = \frac{\frac{\sum_{h=1}^{K} \varphi_h^{t\,'}}{\varphi_i^{t'}}}{\sum_{l=1}^{K} \frac{\sum_{h=1}^{K} \varphi_h^{t\,'}}{\varphi_l^{t'}}}, \tag{18}$$

based on new contribution values in (17). These weights have a sum equal to one, $\sum_{i=1}^{K} \lambda_i^t = 1$; hence, it can directly be used for the weighted average.

Finally, we substitute the calculated weights in (12) to aggregate the global model from clients' models at each iteration as:

$$\Omega^{t+1} = \sum_{i=1}^{K} \lambda_i^t \omega_i^{t+1}. \tag{19}$$

In this regard, the impact of each client's model on the global model will be based on the quality of the dataset it has used and its effective contribution. Hence, if a non-IID dataset declines the global model convergence speed, ShapAvg will assign a lower weight to tackle the divergence problem but still helps the model generalize predictions. Algorithm 1 summarizes the overall process of ShapAvg in FL server and Clients.

Algorithm 1 Shapley Averaging (ShapAvg)

Input: $\mathcal{L}$: Loss function; ω_i^t: Client i's local model at iteration t; Ω^t: Global model at iteration t; D_i: Client i's local dataset; D_s: Server's test dataset; K: Number of clients; T: Number of iterations

Output: Final global model Ω^T

```
/* FL Server */
Initialize global model $\Omega^0$;
while Server performance metric is not met do
    for Each iteration $t \leftarrow 0, 1, \ldots, T$ do
        Broadcast $\Omega_t$ to clients with $\varphi_i^{t-1'} > 0$;
        for Each client $i \leftarrow 1, 2, \ldots, K$ do
            $\omega_i^{t+1} \leftarrow LocalTraining(i, \Omega^t)$;
        end for
        $\varphi_i^t = \frac{1}{K} \sum_{S \subseteq I \setminus \{i\}} \binom{K-1}{|S|}^{-1} \left[ V\left(D_s, \Omega^t_{S \bigcup \{i\}}\right) - V\left(D_s, \Omega^t_S\right) \right]$;
        Calculate $\overline{\varphi}^t$ and $\sigma^t$ and set $\varphi_i^{t'}$ values as in (17);
        $\lambda_i^t = \frac{\frac{\sum_{h=1}^K \varphi_h^{t\,'}}{\varphi_i^{t'}}}{\sum_{l=1}^K \frac{\sum_{h=1}^K \varphi_h^{t\,'}}{\varphi_l^{t'}}}$;
        $\Omega^{t+1} = \sum_{i=1}^K \lambda_i^t \omega_i^{t+1}$;
    end for
end while

/* FL Clients */
LocalTraining(i,$\omega^t$):
$\Delta_i = \nabla l_i(D_i, \omega^t)$;
$\omega_i^{t+1} \leftarrow \omega_i^t - \eta \Delta_i$;
return $\omega_i^{t+1}$ to server;
```

Chapter 4

Implementation

As a significant part of our work is developing an FL platform to simulate and analyze our proposed algorithm, this chapter represents the steps and tools we used to implement our system model. In greater detail, we developed a platform which could be used to implement and simulate any aggregation algorithm or in different FL architectures with a slight change and development. Each corresponding module section will explain the design techniques used for this modularity feature. In the following, we first explain the crucial modules of the platform. Then, we describe the technical challenges and techniques we employed to develop each module.

4.1 Architecture of FL platform

Fig. 4.1 depicts the overall architecture of the FL platform. It consists of five major modules: 1- Dataset manager, 2- Server manager, 3- Clients manager, 4- Model manager, and 5- Federated learning.

The dataset manager is responsible for fetching the data out of a desired dataset and preparing it for the FL platform in terms of format and other aspects. The server manager follows all the processes which will be done in the FL aggregation server as generating the global model and aggregating the models sent by the clients. On the other hand, the client manager implements the training process on the client side to be sent to the server for the final aggregation. Since the primary goal of FL is preserving privacy, the model manager is isolated from the server manager to

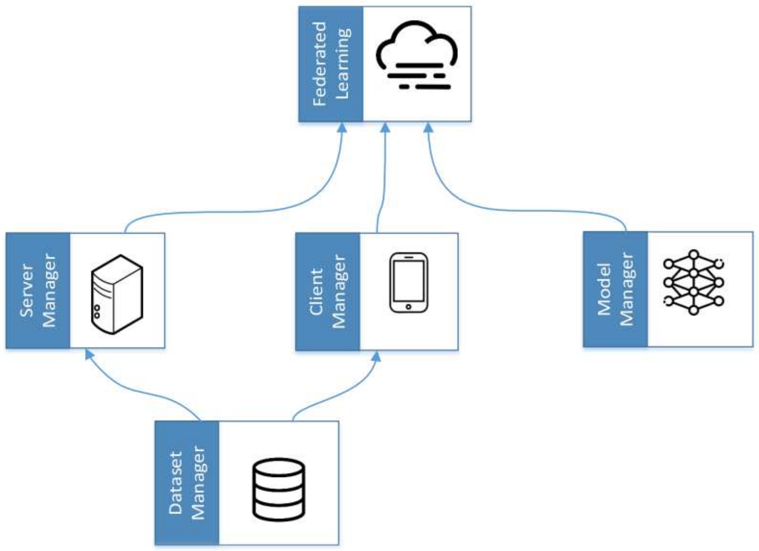


Figure 4.1: Overall architecture of the FL platform

provide only the necessary model information without compromising the clients' privacy. Finally, the federated learning module wraps all of these modules and their provided information into one integrated platform as a federated learning platform. In the following sections, we explain the detail of these modules. snapshots of all module implementation codes could be found in the appendix of this document.

4.1.1 Employed framework

We chose TensorFlow [52] as the underlying machine learning framework for the platform's development. TensorFlow has another framework, TensorFlow Federated [53]. TensorFlow Federated is a branch of TensorFlow which supports the development of FL applications. In the primary phases of platform development, we realized that it does not have a modest method to access the detailed parameters. This framework was useful in simulating FL applications in the early phases when we required the trained global model with the given number of clients and datasets.

In the next phases, we needed to simulate different scenarios where custom or biased datasets, having access to all clients' models after each iteration, evaluating clients' models after each iteration, saving or evaluating the global model in each iteration, and other accesses were required. This was not feasible with TensorFlow Federated. The table 4.1 summarizes the advantages and disadvantages of employing each TensorFlow or TensorFlow Federated framework. Therefore, we employed TensorFlow and developed the FL modules in TensorFlow Federated to thoroughly access all parameters. This development phase led our work to develop the modules mentioned above. We will explain each module in the following sections.

Table 4.1: Summary of key advantages of employing TensorFlow

	TensorFlow Federated	**TensorFlow**
Advantages	Needs less time to develop the app since it has ready modules	(1) Custom federated database generation (2) Access to the local models (3) Access to the global model after each iteration
Disadvantages	(1) Federated databases are pre-generated (2) Do not have access to the local models (3) Only have access to the global model after the last iteration	Many modules should be developed from scratch, which is time-consuming

4.2 Dataset manager

Data is one of the most important factors in federated learning and all machine learning applications. The data quality determines the performance of the final trained machine learning models. The distributed nature of FL affects the training process and adds to its complexity as the data features are unknown to the aggregation server. Also, the privacy of clients' data should always be preserved not to be accessed by a curious server.

In addition to the theoretical challenges of FL datasets, there are some practical ones. As datasets do not have any specific format in terms of the input size, label formats, image encoding in case of an image database, etc., they need some changes to be prepared for the training process. This is called pre-processing. For our application, we need two types of pre-processes. First, preparing the data samples as the input and output of a model like typical machine learning applications. Second, generating a distributed dataset from an aggregated dataset to be used by FL clients.

To follow the modularity design approach, the preparing pre-process is separated into a function, which will be used later by the dataset manager. As the format of each dataset could be different, the responsibility of this function is adapting the dataset data samples for the desired model. This function could be rewritten based on the employed dataset. In our case, we reshaped the input images to the model input size and changed the range of input elements from discrete [0,255] to continuous [0,1].

The approach of splitting the dataset into multiple datasets for the second pre-process crucially affects the final results. These include heterogeneity of the datasets, such as generating IID or Non-IID datasets. To examine our algorithm in both conditions, we developed the dataset manager to generate both IID and Non-IID datasets.

First, we shuffled the dataset elements, including the training and testing data samples. Different datasets have different orders of data samples; by shuffling, we ensure the order of the preliminary dataset samples does not affect the desired heterogeneity of the generated datasets. Then we grouped the samples based on their labels. This was possible since our case was a classification application with limited data labels. This process gives us independent sample groups to draw to generate a dataset with a favourable distribution.

For IID dataset generation, we need a uniform distribution of data samples. For this purpose, we first generated a uniform distribution based on the dataset size. Then, we drew the data samples based on this uniform distribution from the shuffled elements. This guarantees IID datasets as there are no common data samples between the generated datasets, and they have the same distribution measured by their labels.

On the other hand, for Non-IID dataset generation, first, we generated a random distribution for each dataset. These distributions include a different number of data samples of each element of the groups separated by their labels before. Each dataset's distribution generation was repeated to ensure they had unique distributions. Finally, we drew data samples based on these dataset distributions with replacements. Having shared data samples and different distributions simulates allocating Non-IID datasets to FL clients. The next chapter will depict examples of generated datasets employed in the simulation process.

Last, the dataset manager has the following functions: WholeTrainingDatasetSize, WholeTestDatasetSize, GetNextTrainingDataset, GetNextiidTrainingDataset, GetNextTestDataset, GetNextiidTestDataset, GetNextTrainingDatasets, GetNextiidTrainingDatasets, GetNextTestDatasets, GetNextiidTestDatasets, GetAllTestDataset.

The first two functions return training and testing dataset source sizes; the following four functions generate single FL Non-IID and IID training and test datasets. Subsequently, the next four functions generate multiple FL Non-IID and IID training and test datasets based on the number of FL clients as input. And the last function returns the whole test dataset in case an aggregation server needs a comprehensive test dataset for evaluation. After each function call, the dataset manager keeps a record of the last used data samples indexes to avoid generating repetative datasets if called multiple times in the main application.

There is another sub-module called visualizer. This module can depict the distribution of the generated FL client datasets, plot lines, and bar graphs to show the final results of the aggregation server model or clients' contribution used in the subsequent chapter figures.

4.3 Server manager

As the name says, the server manager is responsible for all aggregation server processes. The most important tasks of the aggregation server are global model generation and clients' model aggregation. We have also added the client models and global model evaluation feature to this module. Any proposed aggregation algorithm or evaluation method should be developed and added to this module. We have already added FedAvg and ShapAvg as aggregation algorithms and aggregation and evaluation methods to this model.

First, it generates a global model following the given model parameters from the application. This model will be returned to the clients for local training. After receiving the trained model in each iteration, this module will be called for aggregation.

It has two aggregation options: FedAvg and ShapAvg. For the former, it needs the dataset size of each client's dataset. This information is only available through a separate module, model manager, which serves the clients' privacy. After filling out an array with dataset sizes, it will perform a weighted average over the trained models received from the clients. Subsequently, it will send it back to all clients for further training.

First, ShapAvg training needs an evaluation of the received trained models. Hence, it performs an SV calculation based on the equations provided in the previous chapter. After mentioned adjustments on the contribution values, it aggregates the trained model as explained in (19) to generate the global model of a corresponding iteration. Like the FedAvg, this will send the aggregated model back to all clients for further training.

This module also has a ShapleyValue function, which could be used for both final global model evaluation and a specific client's model if needed. This evaluation is a simple performance metric evaluation by the test dataset from the dataset manager. The performance metrics include accuracy and loss of the provided model.

4.4 Clients manager

We have developed the clients' side program in the client manager module. It has main tasks: 1- Setting clients' parameters, 2- Handling the received global model, 3- Training the local model based on requested performance metrics, and 4- Sending the trained model to the aggregation server.

First, it will set a few parameters which all clients will use. This architecture is for meeting the modular design feature, and they are received from the aggregation server, called constraints. These parameters include the number of local iterations, verbose, optimizer, loss function, and performance metric. Later, we will elaborate on how we set these parameters in the federated learning module section.

The number of iterations determines the minimum number of training rounds a client should perform in the local training. Verbose is for debugging purposes. If verbose is enabled, the code will generate an output reporting the training variables of each client during the local training. Optimizer is the responsible function for adjusting the model parameters. Examples of the optimizers which could be set are stochastic gradient descent (SGD), RMSprop, Adam, Adadelta, Adagrad, and Nadam. The loss function calculates the loss of the local model after each local training round. Examples of the available loss functions are MeanSquaredError, MeanAbsoluteError, Categorical-Crossentropy, BinaryCrossentropy, and LogCosh. Finally, there are only two options for performance metrics: loss or accuracy.

First, the client manager will create an array with the size of the number of participating clients. Upon receiving the global model from the aggregation server as an input, it will insert replicas of the global model into that array as the local model.

There is a function called ”ClientUpdate.” This function trains an input model with an input dataset and given parameters for the requested number of rounds. The client manager will call this function for each client with its model, dataset, and global constraints mentioned before, such as the number of local iterations (rounds) and receives the final accuracy of the training and the trained model. It will record the accuracy in a ” history ” variable for research purposes and returns all clients' locally trained models. Finally, it returns those models, which the server will receive for the aggregation.

It is worth mentioning that in the development and testing of the client manager, the accuracy increase was not identical to the rise while developing the program with TensorFlow Federated. After further debugging, we observed all clients generate the same outputs when enabling the verbose option. Finally, we realized that TensorFlow generates copies of the same model by reference if the model replicas are added to the mentioned model array. Copy by reference is when the array contains different replicas of a model, but they all refer to the same memory location. As a result, a change to one of them will change the value for all, and after all clients' training, only the last client's trained model could be retrieved. To solve this problem, we developed an independent function to create a new model from scratch instead of replicating the model. Then we set the generated model's parameters to the global model's parameters. This approach is called copying by value, which solved our issue.

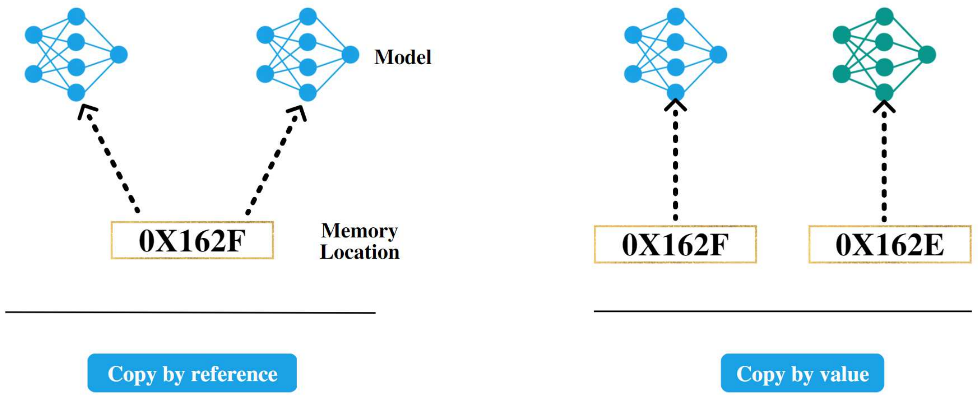


Figure 4.2: Copy by reference or value

4.5 Model manager

Everything related explicitly to either global or local models falls in this module. This simplifies the system's modularity and, in some cases, preserves the model's privacy. For example, the FedAvg aggregation algorithm needs the dataset size of each client who has sent their trained model for the aggregation. As the aggregation server is not supposed to gain any knowledge about clients' datasets, the model manager can get the models and return them with a corresponding dataset size. Since the aggregation server does not receive the models and dataset sizes directly from the clients, it limits a curious server from retrieving dataset information.

Other applications of the model manager module could be the calculations, which are done on the models. An example is differential privacy, which includes deliberately adding noise to the clients' models to prevent the server from gaining knowledge about clients' datasets. In this case, this implementation will be added to the model manager to avoid implementation complexity in the client manager and isolate this operation from the aggregation server for privacy. In case of additive implementation complexity, even the model evaluation task by a test dataset could be delegated to this module.

4.6 Federated learning wrapper module

This module employs all the abovementioned modules as in figure 4.1 to form a single FL platform. First, it will set a few parameters, which the server manager and client manager modules will use. These are the parameters explained in the corresponding module sections and will receive their value in the initiation phase of this module. Hence, the simulation platform user can integrate the platform settings only in one area of the code for usage simplicity. These parameters include all mentioned parameters of clients and the aggregation server plus the number of FL iterations, Shapley evaluation, aggregation algorithm, the number of clients, and the global model.

As the name indicates, the number of FL iterations is the number of rounds that the FL process from global model aggregation by the server to receiving the trained models from the clients must be repeated. If the Shapley evaluation is enabled, the final global model will be evaluated by Shapley using a dataset that must be provided in advance to measure the final contribution of each client in the final global model. This should not be mistaken for the contribution value that we explained in our system model. In fact, this value is the final contribution value. The value we proposed in the previous chapter is a contribution value calculated after each FL iteration to be employed in aggregation.

The aggregation algorithm determines the algorithm the user will employ for the simulation. Currently, the available options are FedAvg and ShapAvg to compare the improvements in the results. The number of clients denotes the number of created instances of clients using the client manager to be employed in the FL simulation. Finally, the global model is the model architecture of the global model, which will be transmitted between the server manager and client manager based on the FL process. The architecture of this model will be fixed for the whole simulation duration; only its parameters will be adjusted based on the FL process.

In addition to the required simulation reports printed during the simulation, this module will return the final simulation results (accuracy or loss depending on the performance metric in the server manager) and an array of Shapley values in case of enabling the Shapley evaluation in the parameters of the initiation phase.

Chapter 5

Simulation and Results

In this chapter, we explain the parameters and simulation environment we have used. Then, we show the improvements of our proposed algorithm over the typical aggregation algorithm.

5.1 Simulation

To simulate our algorithm, we utilized a classification example. We used the MNIST dataset [54] as the source dataset and Google Colab as the hardware infrastructure. We will explain the generation of a federated dataset from the MNIST dataset in the following section. Google Colab is a Google Research product. This is particularly well suited to machine learning, data analysis, and education. It enables anyone to create and execute arbitrary Python code through the browser. Technically speaking, Google Colab is a hosted Jupyter notebook service that offers free access to computer resources, including GPUs, and requires no setup to use [55].

5.1.1 Dataset generation and adversary clients simulation

We used the MNIST dataset for the simulation, a handwritten digits dataset of 60,000 training and 10,000 testing examples. We first shuffled the training examples; then, we generated three types of federated datasets from this source: IID, Non-IID, and datasets for poisoning or free-riding attackers.

IID dataset generation

First, we grouped the examples by their labels into ten-digit groups to generate federated IID datasets. This forms the main pool to draw the samples to generate a federated dataset, and this pool will be used for all types of dataset generation. However, the dataset generation should be based on a specific distribution or features, which conforms to the mentioned dataset types. Then, we generated a digit distribution that will be used for all dataset generation. Finally, we did a non-replacing sampling of 110 examples based on the generated distribution for each dataset. These datasets have the same digit distribution, equal size, and no common examples.

Non-IID dataset generation

To generate federated non-IID datasets, we assigned indexes to the training examples after the shuffling. Then, we drew between 100 to 120 samples (with an average of 110 samples as in IID datasets for an analogous simulation result) with randomly chosen indexes for each dataset. These datasets have different sizes with different handwriting digit distributions. Some common examples between the generated non-IID datasets are probable due to randomly chosen indexes during the sampling.

Poisoning or free-riding attackers' dataset generation

To simulate the adversary clients in our scenario, we tried to generate the federated datasets which will represent their effect on their collaboration in the FL training by changing the quality of their datasets. The threat model for a data poisoning attacked is described in [10]. Accordingly, we replaced half of the labels for data poisoning attackers. Also, these clients will report twice the size of their dataset to the FL server. This affects the FedAvg algorithm as it takes this data from the clients as trusted data to be used in the aggregation algorithm.

We followed the naive attacker strategy in [11] to simulate the free-riding attackers by setting the model weights to a random value. We added a level of complexity to this strategy to make the attack detection more challenging. We generated the random model weights in the range of the global model weights they received in the last iteration, so the model weights are not completely random and much far from other clients' model weights. However, the transferred model will have a low accuracy as it is a random model. However, the model weights are in the range of the previously received global model, so it represents a smart client who has not employed its resources but is trying to disguise its fake model.

Validation datasets

Finally, we used 15 examples for each client as a validation dataset and 100 examples for the FL server as a test dataset. Figure 5.1 and figure 5.2 show a visualized bar chart of the federated IID dataset with the size of 110 samples and the federated Non-IID dataset with a size between 100 and 120 samples for each dataset, generated by this method, respectively. The horizontal axis shows the sample's label, and the vertical axis shows the available number of samples of that label. All IID datasets have the same size and distribution, and Non-IID datasets have different sizes between 100 and 120 samples and different distributions.

5.1.2 Results acquisition technique

In every code execution, Google Colab gives the client a live session to see the simulation phases and the log outputs. This live session has a few drawbacks. One of the most important drawbacks is that the connection and session, as a result, are not always stable. Google Colab tries to establish a new connection to resume the live session, but based on our experience, this does not always result in a successful session recovery. Hence, for a simulation that takes a long time, a loss of the connection could result in losing the final results.

The technique we used to tackle this issue is saving each step's results, settings, and simulation variables in safe storage. This includes saving the number of FL clients, the number of iterations, the final accuracy, the final loss, the date and time, and the global model. We saved the last item based on the simulation requirements. For instance, if a simulation requires many iterations, which increases the chance of connection loss, we save the global model after each global iteration to resume the work from the last saved iteration in case of permanently losing the session. The reason behind the fact that we do not always use this feature is that saving the model after each iteration could affect the speed of the simulation. Therefore, we only use this if needed.

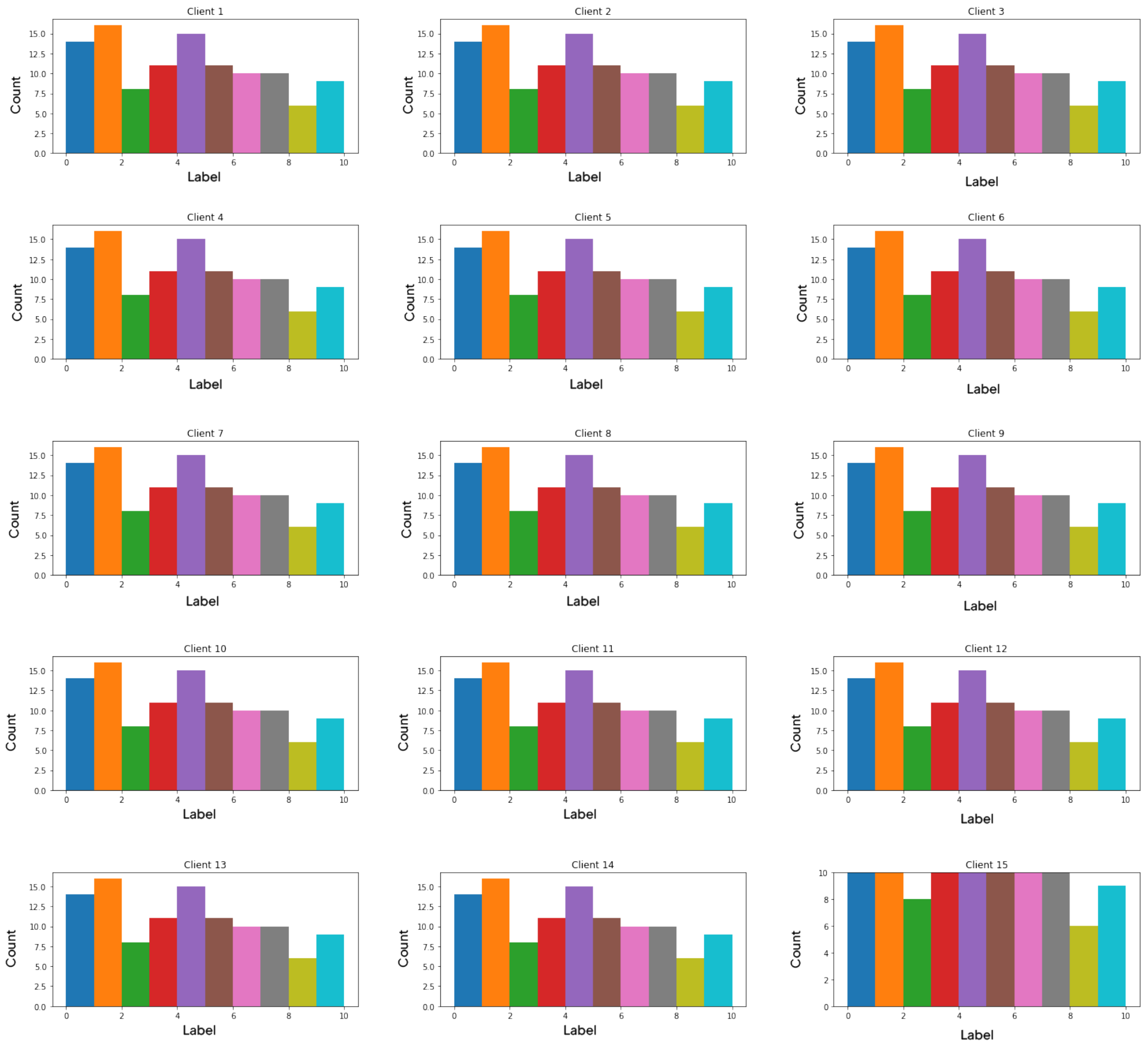


Figure 5.1: The generated federated IID Dataset, showing the number of samples of each label

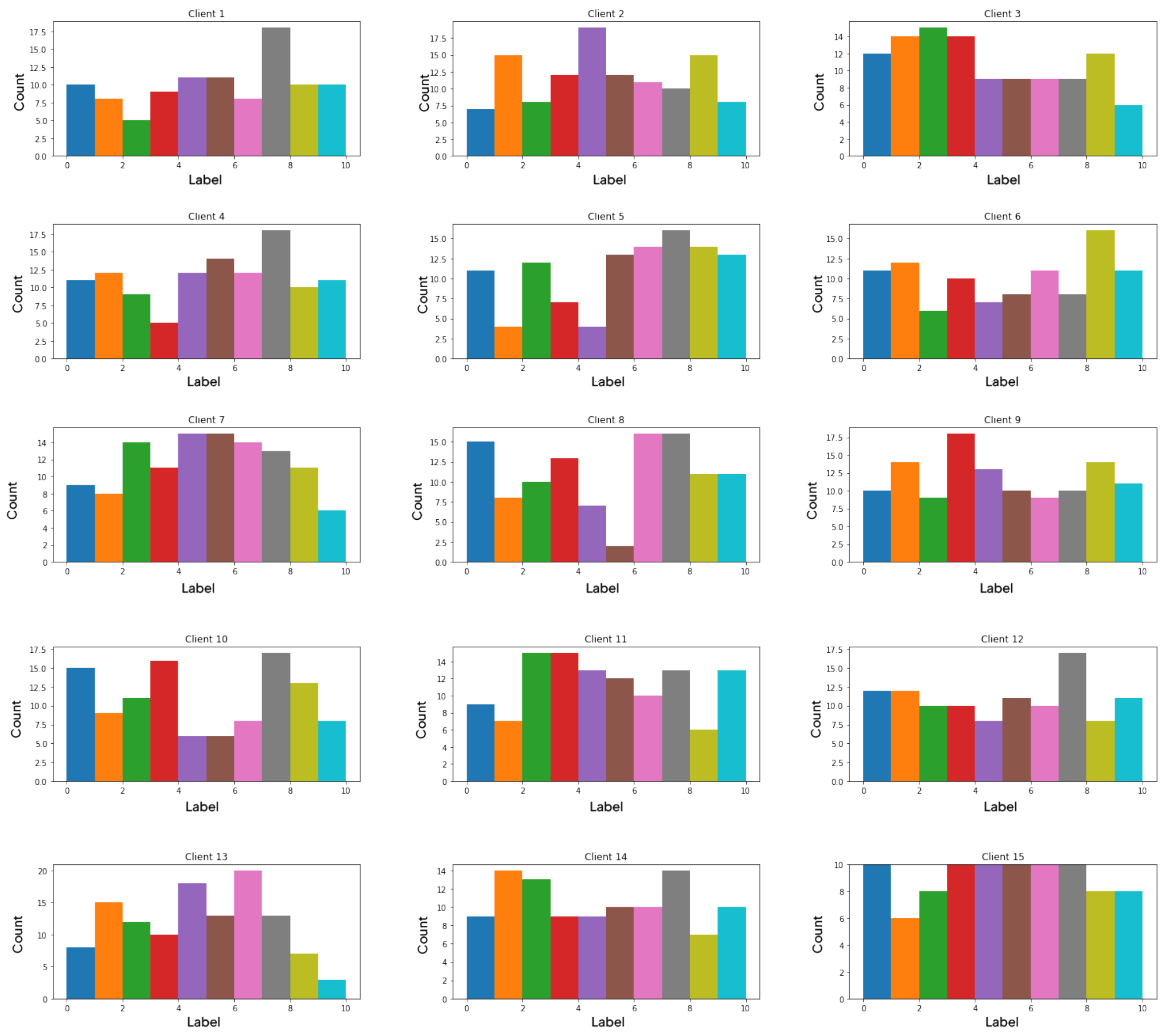


Figure 5.2: The generated federated Non-IID Dataset, showing the number of samples of each label

5.1.3 Simulation scenarios and results

Scenarios and settings

We simulated 15 FL clients and an aggregation server with a light classification model as the global model. To test our proposed algorithm, our experiment includes three scenarios: a) clients with IID datasets, b) clients with non-IID datasets, and c) two free-riders, three data-poisoning attackers, and ten normal clients. We simulated these scenarios both with our ShapAvg and the typical FedAvg aggregation algorithms. The complexity of the simulation increases from the first to the third scenario to challenge our algorithm performance compared to the FedAvg.

The global model is the same for all three scenarios to have analogous results, but the employed datasets differ. These are the datasets mentioned in the previous section. The third scenario contains adversary clients such as free riders, data-poisoning attackers, and regular clients who will participate adequately. In this regard, we can examine the performance of our model in evaluating different types of clients in the presence of all client types.

The FL training contains one hundred global rounds to train the global model. Plus, each client trains the model for ten iterations before sending it to the aggregation server. We set the number of local iterations to a fixed number because of three reasons. First, by having a similar number of iterations and using the same resources for simulating all FL clients, we attempt to simulate a synchronous FL where all clients submit their models at roughly the same time, and the aggregation server does not have to wait a long time for the bottleneck client. Second, evaluating the client's model after the same number of iterations gives the aggregation server a more fair insight into those clients' dataset quality. Third, this simplifies the implementation and accelerates the simulation as there is no need to wait for the slower clients who reach a specified accuracy because of their low-quality dataset.

Results and discussion

For comparing our algorithm outcome with FedAvg we used the loss as the performance metric to represent the convergence speed of the global model. It efficiently shows the distance between the desired state and the output of the global model. We compared ShapAvg and FedAvg for the three scenarios. The following figures depict the results of these scenarios by comparing the speed of loss decrease over the number of FL iterations.

Figure 5.3 showed roughly similar convergence speed while using the IID datasets; however, ShapAvg reached the final loss of FedAvg 11% faster. Using IID datasets is the least challenging scenario for aggregation algorithms. As a result, we can see that the final loss values for both aggregation algorithms are close to each other. However, it takes ShapAvg a shorter time to reach the final FedAvg loss value. This could be well observed in the middle of the chart, where after twenty iterations, ShapAvg reduction speed differs more noticeably from FedAvg reduction speed.

In figure 5.4, ShapAvg outperformed the FedAvg while using non-IID datasets in terms of the final loss by 17% and the convergence speed. In this scenario, ShapAvg outperforms the FedAvg more significantly after only a few iterations. It is not only faster than the FedAvg but also more effective in decreasing the loss value after one hundred iterations. Another point in this chart is that FedAvg has a smoother chart than the ShapAvg. This happens because FedAvg takes a weighted average of the clients' models with fixed weights at the end of all iterations. On the other hand, ShapAvg weights in its weighted average differ based on the clients' evaluation. The weight for a client will be higher in an iteration if it has performed better. This results in small bounces in its chart.

Finally, in figure 5.5, the final loss by ShapAvg was 27% lower than that of the FedAvg if there are free-riders and poisoning attackers among the participating clients. Also, it shows that the loss reduction speed difference between FedAvg and ShapAvg is at its highest in the last scenario. In this scenario, using random and false models by free riders by data-poisoning attackers affects FedAvg. Although both charts have bounces, those of ShapAvg are smaller than FedAvg. This is because, unlike the previous scenario, which using the same weights for weighted averaging resulted in a smoother chart, this scenario will affect FedAvg negatively since the adversary clients have announced a big dataset size with a low-quality model. On the other hand, ShapAvg is less prone to this security risk as it evaluates the clients' performance. It will remove adversary clients from the aggregation if needed until they perform better.

The results show that ShapAvg is more robust than FedAvg when using non-IID datasets or in the presence of data-poisoning attackers and free-riders. ShapAvg benefits from evaluating the received model to tackle the divergence problem of FedAvg in the global model. Therefore, the gap between the final achieved loss of ShapAvg and FedAvg significantly increases when the received models from clients have low quality in terms of performance metrics. This effect is well presented in figure 5.5, where one-third of the clients are free-riders or data-poisoning attackers. Table 5.1 shows the final loss value of FedAvg and ShapAvg in the mentioned three simulation scenarios.

Table 5.1: Final loss value of FedAvg and ShapAvg in three scenarios

	FedAvg	ShapAvg
Scenario A: IID datasets	0.6263	0.5893
Scenario B: Non-IID datasets	0.8993	0.7455
Scenario C: Adversary clients	1.1218	0.8184

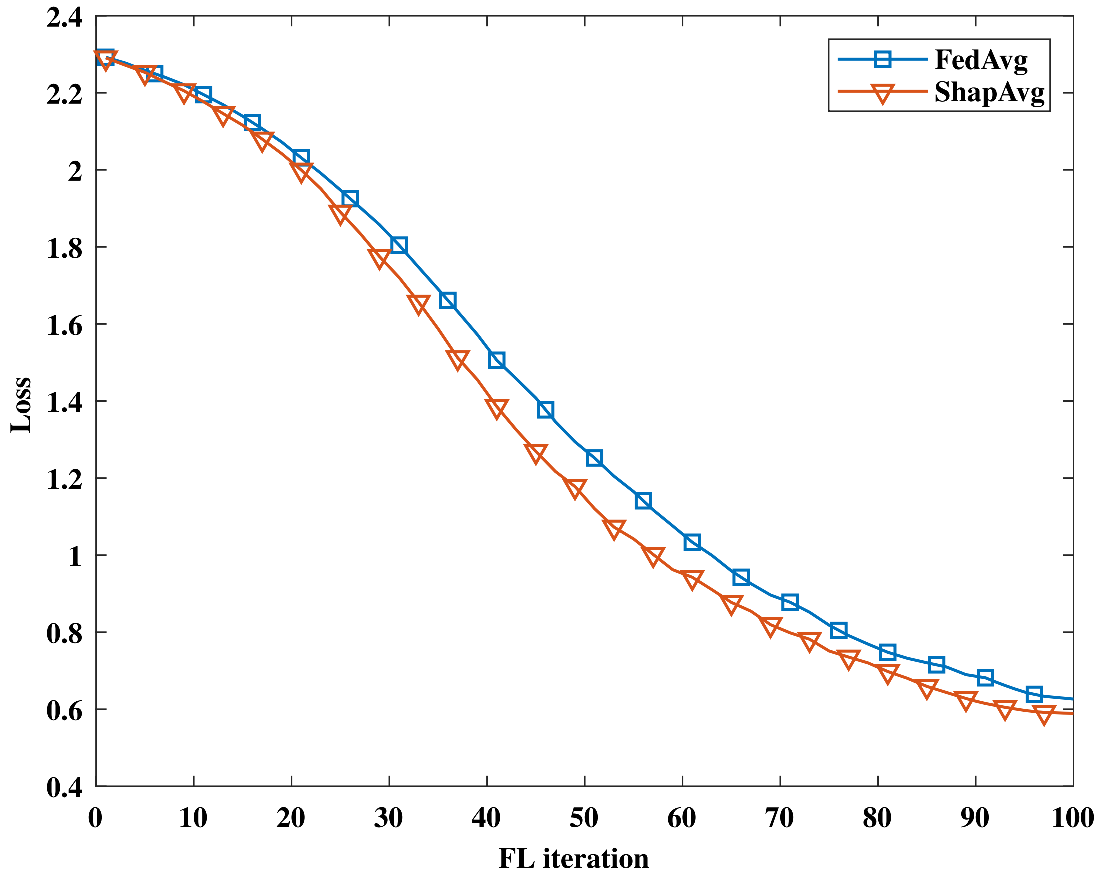


Figure 5.3: Convergence speed comparison of FedAvg and ShapAvg with IID-datasets

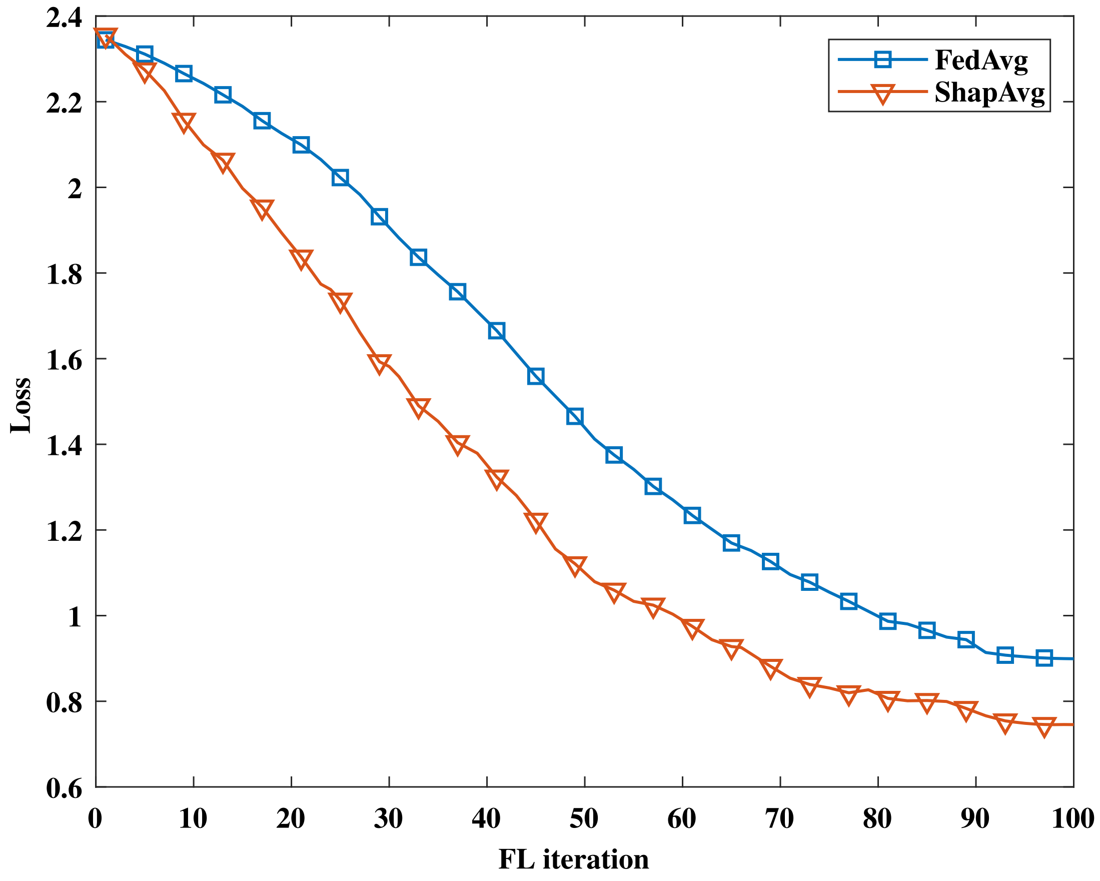


Figure 5.4: Convergence speed comparison of FedAvg and ShapAvg with non-IID datasets

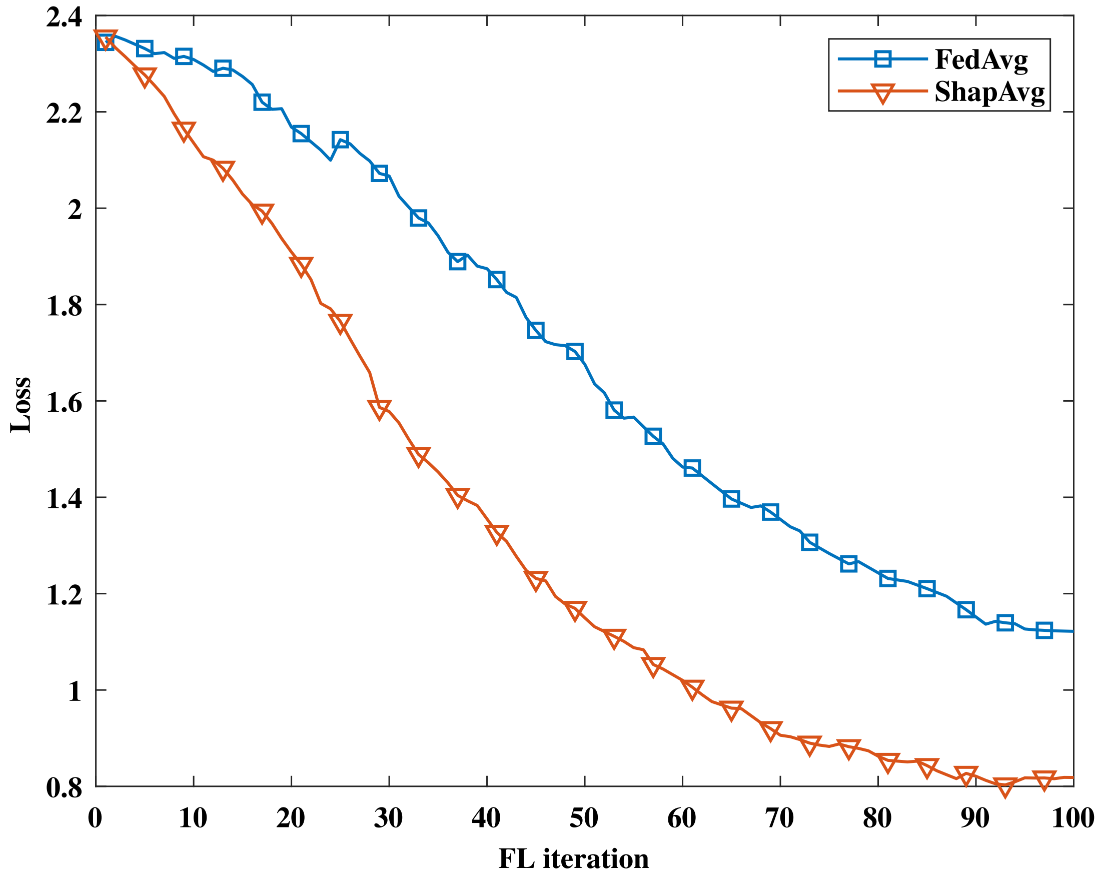


Figure 5.5: Convergence speed comparison of FedAvg and ShapAvg with data poisoning and free-riding attacks

Chapter 6

Conclusion and Future Work

This chapter concludes the contributions of this work, and finally, we mention potential research tips which could be considered in future work.

6.1 Conclusion

FL helps preserve the clients' data privacy while participating in the learning process. However, utilizing such a platform is challenging due to its implementation complexity. This work proposed an aggregation algorithm with an evaluation technique to tackle some security and divergence problems of conventional FedAvg. Also, the evaluation algorithm paves the way for profit distribution in FL with few modifications. The observed simulation results show that our proposed algorithm can optimize the FL global model performance in the presence of malicious clients and non-IID datasets by effectively aggregating their models based on their contribution to the total performance.

6.2 Future work

We recognize our research as an essential preliminary step in adding evaluation to FL. Hence, in this section, we mention the parts of our work which could be Future work that can focus on two goals: first, improve the performance of the contribution value calculation by the FL server; second, develop the algorithm to assess the clients based on more than one iteration so that the clients do not lose the participation opportunity due to one iteration under-performance.

As mentioned, the lack of an evaluation method makes the system vulnerable to attacks and failures. We solved this issue by proposing an evaluation method added to the aggregation algorithm. However, as adding every feature could cause complexity, adding an evaluation method uses more resources. Although this added complexity is inevitable, working on the method's efficiency could be a potential future work. One possible solution could be approximating the evaluation value with an approximation error depending on the required accuracy level in employing the aggregation contribution.

Currently, the clients are evaluated based on their performance in an FL training iteration. If clients underperform others by the metrics defined in chapter 3, they will not receive the most up-to-date model in the following training iteration. This helps to remove the free-rider or poisoning attackers from the participating clients. However, in some cases, an under-performance in one of the iterations will hinder a client from catching up with others to receive the updated model in the next iteration if it does not significantly improve its model quality. This depends on their dataset quality and available resources to train the model. Future work could develop the algorithm to have an adjustable threshold on the level of strictness of removing the client from the training process by not sending the model. This could give them an adjustable failing chance based on the aggregation server constraints.

Bibliography

[1] Yi Liu, James J. Q. Yu, Jiawen Kang, Dusit Niyato, and Shuyu Zhang. “Privacy-Preserving Traffic Flow Prediction: A Federated Learning Approach”. In: *IEEE Internet of Things Journal* 7.8 (Aug. 2020), pp. 7751–7763. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2991401.

[2] Ganta Sruthi, Chokkakula Likitha Ram, Malegam Koushik Sai, Bhanu Pratap Singh, Nikhil Majhotra, and Neha Sharma. “Cancer Prediction using Machine Learning”. In: *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)*. Vol. 2. Feb. 2022, pp. 217–221. DOI: 10.1109/ICIPTM54933.2022.9754059.

[3] S. D. Okegbile and Jun Cai. “Edge-assisted human-to-virtual twin connectivity scheme for human digital twin frameworks”. In: *IEEE Vehicular Technology Conference (Spring)*. Helsinki, June 2022, pp. 1–6.

[4] S. D. Okegbile, Jun Cai, C. Yi, and D. Niyato. “Human Digital Twin for Personalized Healthcare: Vision, Architecture and Future Directions”. In: *IEEE Network*. May 2022.

[5] Shuaicheng Ma, Yang Cao, and Li Xiong. “Transparent Contribution Evaluation for Secure Federated Learning on Blockchain”. In: *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. ISSN: 2473-3490. Apr. 2021, pp. 88–91. DOI: 10.1109/ICDEW53142.2021.00023.

[6] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong-Zhi Gao, Siu-Ming Yiu, and Kai Chen. “Multi-key privacy-preserving deep learning in cloud computing”. en. In: *Future Generation Computer Systems* 74 (Sept. 2017), pp. 76–85. ISSN: 0167-739X. DOI: 10.1016/j.future.2017.02.006.

[7] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. "Federated Learning for Mobile Keyboard Prediction". In: *arXiv:1811.03604 [cs]* (Feb. 2019).

[8] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". en. In: *Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 1273–1282.

[9] Reza Shokri and Vitaly Shmatikov. "Privacy-Preserving Deep Learning". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 1310–1321. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813687.

[10] Pengrui Liu, Xiangrui Xu, and Wei Wang. "Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives". In: *Cybersecurity* 5.1 (Feb. 2022), p. 4. ISSN: 2523-3246. DOI: 10.1186/s42400-021-00105-6.

[11] Jierui Lin, Min Du, and Jian Liu. *Free-riders in Federated Learning: Attacks and Defenses*. arXiv:1911.12560 [cs, stat]. Nov. 2019. DOI: 10.48550/arXiv.1911.12560.

[12] L. S. Shapley. "17. A Value for n-Person Games". en. In: *Contributions to the Theory of Games (AM-28)*. Vol. II. Princeton University Press, Mar. 2016, pp. 307–318. ISBN: 978-1-4008-8197-0. DOI: 10.1515/9781400881970-018.

[13] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020). Conference Name: IEEE Communications Surveys & Tutorials, pp. 2031–2063. ISSN: 1553-877X. DOI: 10.1109/COMST.2020.2986024.

[14] Tran The Anh, Nguyen Cong Luong, Dusit Niyato, Dong In Kim, and Li-Chun Wang. "Efficient Training Management for Mobile Crowd-Machine Learning: A Deep Reinforcement Learning Approach". In: *IEEE Wireless Communications Letters* 8.5 (Oct. 2019), pp. 1345–1348. ISSN: 2162-2345. DOI: 10.1109/LWC.2019.2917133.

[15] Dongzhu Liu and Osvaldo Simeone. "Privacy for Free: Wireless Federated Learning via Uncoded Transmission With Adaptive Power Control". In: *IEEE Journal on Selected Areas in Communications* 39.1 (Jan. 2021), pp. 170–185. ISSN: 1558-0008. DOI: 10.1109/JSAC.2020.3036948.

[16] Heinrich von Stackelberg. *Market Structure and Equilibrium*. en. Berlin Heidelberg: Springer-Verlag, 2011. ISBN: 978-3-642-12585-0. DOI: 10.1007/978-3-642-12586-7.

[17] Yunus Sarikaya and Ozgur Ercetin. "Motivating Workers in Federated Learning: A Stackelberg Game Perspective". In: *IEEE Networking Letters* 2.1 (Mar. 2020), pp. 23–27. ISSN: 2576-3156. DOI: 10.1109/LNET.2019.2947144.

[18] Shaohan Feng, Dusit Niyato, Ping Wang, Dong In Kim, and Ying-Chang Liang. "Joint Service Pricing and Cooperative Relay Communication for Federated Learning". In: *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. July 2019, pp. 815–820. DOI: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00148.

[19] Yufeng Zhan, Peng Li, Zhihao Qu, Deze Zeng, and Song Guo. "A Learning-Based Incentive Mechanism for Federated Learning". In: *IEEE Internet of Things Journal* 7.7 (July 2020), pp. 6360–6368. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2967772.

[20] Shashi Raj Pandey, Nguyen H. Tran, Mehdi Bennis, Yan Kyaw Tun, Aunas Manzoor, and Choong Seon Hong. "A Crowdsourcing Framework for On-Device Federated Learning". In: *IEEE Transactions on Wireless Communications* 19.5 (May 2020), pp. 3241–3256. ISSN: 1558-2248. DOI: 10.1109/TWC.2020.2971981.

[21] Jiawen Kang, Zehui Xiong, Dusit Niyato, Shengli Xie, and Junshan Zhang. "Incentive Mechanism for Reliable Federated Learning: A Joint Optimization Approach to Combining Reputation and Contract Theory". In: *IEEE Internet of Things Journal* 6.6 (Dec. 2019), pp. 10700–10714. ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2940820.

[22] Yang Zhao, Jun Zhao, Linshan Jiang, Rui Tan, Dusit Niyato, Zengxiang Li, Lingjuan Lyu, and Yingbo Liu. “Privacy-Preserving Blockchain-Based Federated Learning for IoT Devices”. In: *IEEE Internet of Things Journal* 8.3 (Feb. 2021), pp. 1817–1829. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3017377.

[23] Kentaroh Toyoda, Jun Zhao, Allan Neng Sheng Zhang, and P. Takis Mathiopoulos. “Blockchain-Enabled Federated Learning With Mechanism Design”. In: *IEEE Access* 8 (2020), pp. 219744–219756. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3043037.

[24] Sungwook Kim. “Incentive Design and Differential Privacy Based Federated Learning: A Mechanism Design Perspective”. In: *IEEE Access* 8 (2020), pp. 187317–187325. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3030888.

[25] Ningning Ding, Zhixuan Fang, and Jianwei Huang. “Optimal Contract Design for Efficient Federated Learning With Multi-Dimensional Private Information”. In: *IEEE Journal on Selected Areas in Communications* 39.1 (Jan. 2021), pp. 186–200. ISSN: 1558-0008. DOI: 10.1109/JSAC.2020.3036944.

[26] Joohyung Lee, DaeJin Kim, and Dusit Niyato. “Market Analysis of Distributed Learning Resource Management for Internet of Things: A Game-Theoretic Approach”. In: *IEEE Internet of Things Journal* 7.9 (Sept. 2020), pp. 8430–8439. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2991725.

[27] Zhe Peng, Jianliang Xu, Xiaowen Chu, Shang Gao, Yuan Yao, Rong Gu, and Yuzhe Tang. “VFChain: Enabling Verifiable and Auditable Federated Learning via Blockchain Systems”. In: *IEEE Transactions on Network Science and Engineering* (2021), pp. 1–1. ISSN: 2327-4697. DOI: 10.1109/TNSE.2021.3050781.

[28] Yang Chen, Xiaoyan Sun, and Yaochu Jin. “Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.10 (Oct. 2020), pp. 4229–4238. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2019.2953131.

[29] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. “Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (Sept. 2020), pp. 3400–3413. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2019.2944481.

[30] Siqi Luo, Xu Chen, Qiong Wu, Zhi Zhou, and Shuai Yu. “HFEL: Joint Edge Association and Resource Allocation for Cost-Efficient Hierarchical Federated Edge Learning”. In: *IEEE Transactions on Wireless Communications* 19.10 (Oct. 2020), pp. 6535–6548. ISSN: 1558-2248. DOI: 10.1109/TWC.2020.3003744.

[31] Qiong Wu, Kaiwen He, and Xu Chen. “Personalized Federated Learning for Intelligent IoT Applications: A Cloud-Edge Based Framework”. In: *IEEE Open Journal of the Computer Society* 1 (2020), pp. 35–44. ISSN: 2644-1268. DOI: 10.1109/OJCS.2020.2993259.

[32] Lei Feng, Yiqi Zhao, Shaoyong Guo, Xuesong Qiu, Wenjing Li, and Peng Yu. “Blockchain-based Asynchronous Federated Learning for Internet of Things”. In: *IEEE Transactions on Computers* (2021), pp. 1–1. ISSN: 1557-9956. DOI: 10.1109/TC.2021.3072033.

[33] Haoye Chai, Supeng Leng, Yijin Chen, and Ke Zhang. “A Hierarchical Blockchain-Enabled Federated Learning Algorithm for Knowledge Sharing in Internet of Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–12. ISSN: 1558-0016. DOI: 10.1109/TITS.2020.3002712.

[34] Hai Jin, Xiaohai Dai, Jiang Xiao, Baochun Li, Huichuwu Li, and Yan Zhang. “Cross-Cluster Federated Learning and Blockchain for Internet of Medical Things”. In: *IEEE Internet of Things Journal* (2021), pp. 1–1. ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3081578.

[35] Hong Liu, Shuaipeng Zhang, Pengfei Zhang, Xinqiang Zhou, Xuebin Shao, Geguang Pu, and Yan Zhang. “Blockchain and Federated Learning for Collaborative Intrusion Detection in Vehicular Edge Computing”. In: *IEEE Transactions on Vehicular Technology* (2021), pp. 1–1. ISSN: 1939-9359. DOI: 10.1109/TVT.2021.3076780.

[36] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. “Federated Learning with Non-IID Data”. In: *arXiv:1806.00582 [cs, stat]* (June 2018).

[37] Te-Chuan Chiu, Yuan-Yao Shih, Ai-Chun Pang, Chieh-Sheng Wang, Wei Weng, and Chun-Ting Chou. “Semisupervised Distributed Learning With Non-IID Data for AIoT Service Platform”. In: *IEEE Internet of Things Journal* 7.10 (Oct. 2020), pp. 9266–9277. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2995162.

[38] Wenyu Zhang, Xiumin Wang, Pan Zhou, Weiwei Wu, and Xinglin Zhang. “Client Selection for Federated Learning With Non-IID Data in Mobile Edge Computing”. In: *IEEE Access* 9 (2021), pp. 24462–24474. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3056919.

[39] Yunfeng Li, Xiaoling Li, Gangqiang Li, and Zhitao Li. “Privacy Protection in Prosumer Energy Management Based on Federated Learning”. In: *IEEE Access* 9 (2021), pp. 16707–16715. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3053573.

[40] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. “VerifyNet: Secure and Verifiable Federated Learning”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 911–926. ISSN: 1556-6021. DOI: 10.1109/TIFS.2019.2929409.

[41] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. “Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers”. In: *International Journal of Security and Networks* 10.3 (Sept. 2015), pp. 137–150. ISSN: 1747-8405. DOI: 10.1504/IJSN.2015.071829.

[42] Meng Shen, Huan Wang, Bin Zhang, Liehuang Zhu, Ke Xu, Qi Li, and Xiaojiang Du. “Exploiting Unintended Property Leakage in Blockchain-Assisted Federated Learning for Intelligent Edge Computing”. In: *IEEE Internet of Things Journal* 8.4 (Feb. 2021), pp. 2265–2275. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3028110.

[43] Martin Abadi, Andy Chu, Ian Goodfellow, Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. “Deep Learning with Differential Privacy”. In: 2016, pp. 308–318. URL: https://arxiv.org/abs/1607.00133 (visited on 05/23/2021).

[44] Nishat I. Mowla, Nguyen H. Tran, Inshil Doh, and Kijoon Chae. “AFRL: Adaptive federated reinforcement learning for intelligent jamming defense in FANET”. In: *Journal of Communications and Networks* 22.3 (June 2020), pp. 244–258. ISSN: 1976-5541. DOI: 10.1109/JCN.2020.000015.

[45] Wei Yang Bryan Lim, Zehui Xiong, Chunyan Miao, Dusit Niyato, Qiang Yang, Cyril Leung, and H. Vincent Poor. “Hierarchical Incentive Mechanism Design for Federated Machine Learning in Mobile Networks”. In: *IEEE Internet of Things Journal* 7.10 (Oct. 2020), pp. 9575–9588. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2985694.

[46] Shiva Raj Pokhrel and Jinho Choi. “Federated Learning With Blockchain for Autonomous Vehicles: Analysis and Design Challenges”. In: *IEEE Transactions on Communications* 68.8 (Aug. 2020), pp. 4734–4746. ISSN: 1558-0857. DOI: 10.1109/TCOMM.2020.2990686.

[47] Mohamed Abdur Rahman, M. Shamim Hossain, Mohammad Saiful Islam, Nabil A. Alrajeh, and Ghulam Muhammad. “Secure and Provenance Enhanced Internet of Health Things Framework: A Blockchain Managed Federated Learning Approach”. In: *IEEE Access* 8 (2020), pp. 205071–205087. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3037474.

[48] Laizhong Cui, Xiaoxin Su, Zhongxing Ming, Ziteng Chen, Shu Yang, Yipeng Zhou, and Wei Xiao. “CREAT: Blockchain-assisted Compression Algorithm of Federated Learning for Content Caching in Edge Computing”. In: *IEEE Internet of Things Journal* (2020), pp. 1–1. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3014370.

[49] Shiva Raj Pokhrel. “Blockchain Brings Trust to Collaborative Drones and LEO Satellites: An Intelligent Decentralized Learning in the Space”. In: *IEEE Sensors Journal* (2021), pp. 1–1. ISSN: 1558-1748. DOI: 10.1109/JSEN.2021.3060185.

[50] Rajesh Kumar, Abdullah Aman Khan, Jay Kumar, A. Zakria, Noorbakhsh Amiri Golilarz, Simin Zhang, Yang Ting, Chengyu Zheng, and Wenyong Wang. “Blockchain-Federated-Learning and Deep Learning Models for COVID-19 detection using CT Imaging”. In: *IEEE Sensors Journal* (2021), pp. 1–1. ISSN: 1558-1748. DOI: 10.1109/JSEN.2021.3076767.

[51] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 843–852. DOI: 10.1109/ICCV.2017.97.

[52] *TensorFlow*. en. URL: https://www.tensorflow.org/ (visited on 03/11/2023).

[53] *TensorFlow Federated*. en. URL: https://www.tensorflow.org/federated (visited on 03/11/2023).

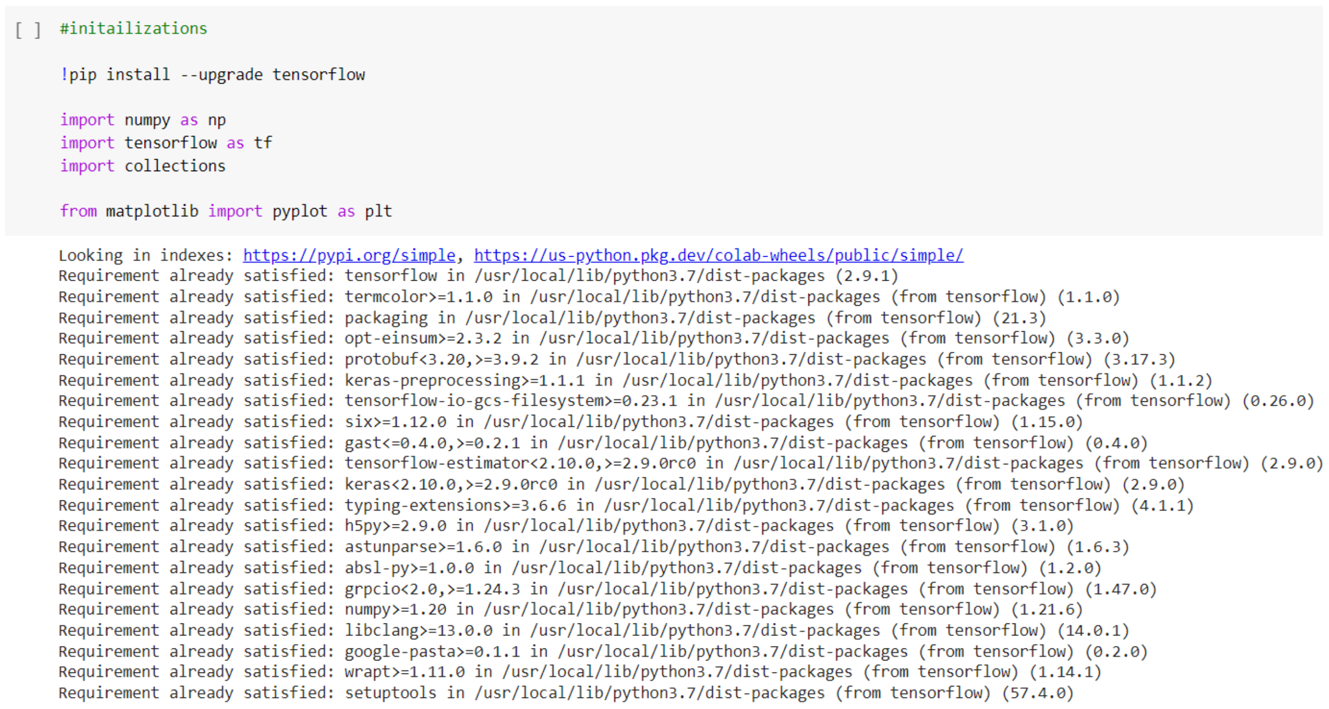
[54] *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 07/10/2022).

[55] *Google Colaboratory*. en. URL: https://colab.research.google.com/ (visited on 03/11/2023).

Appendix A

Implementation Codes

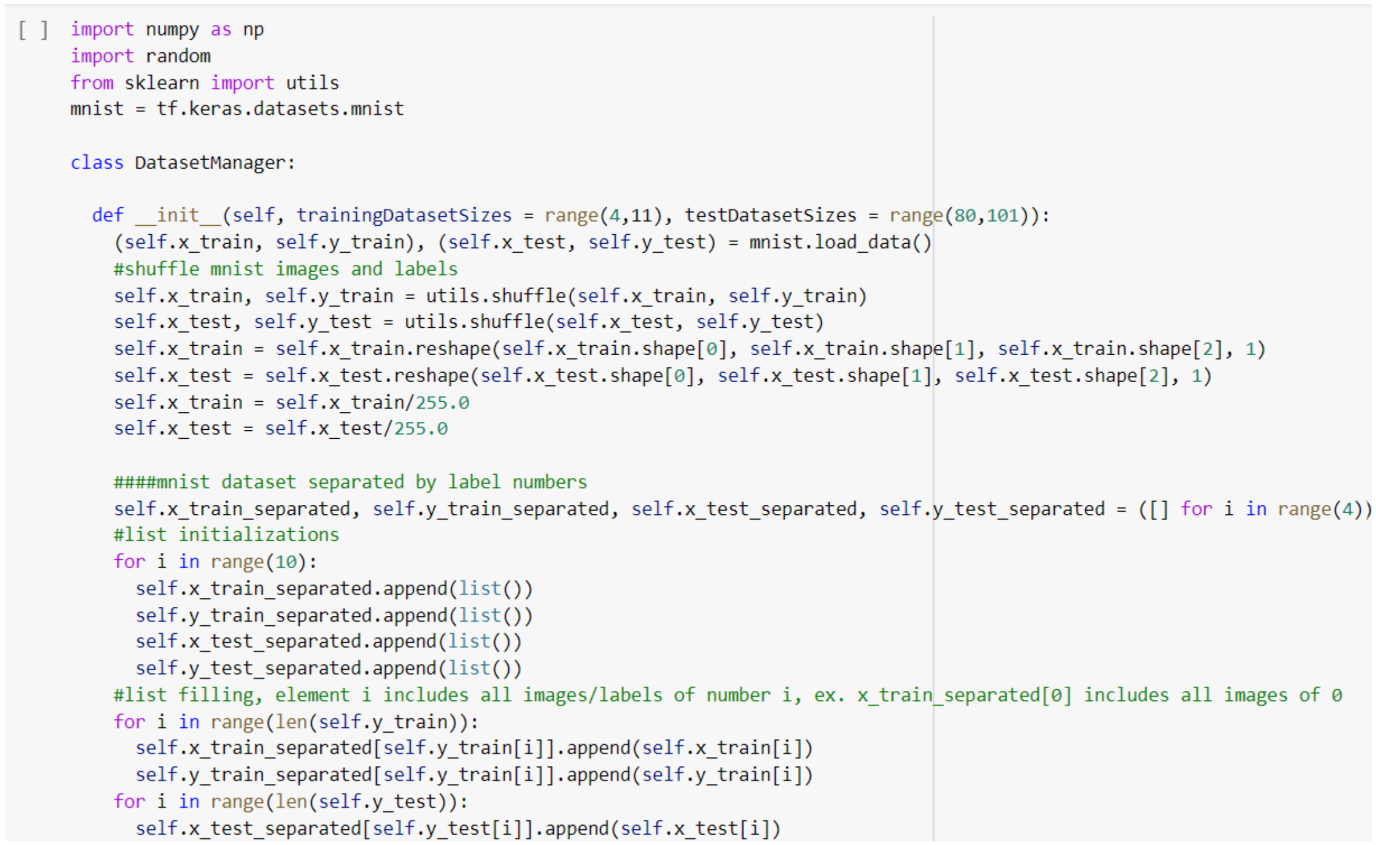
The appendix contains the development codes of the FL platform.



```
[ ] #initailizations

    !pip install --upgrade tensorflow

    import numpy as np
    import tensorflow as tf
    import collections

    from matplotlib import pyplot as plt
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.9.1)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow) (21.3)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.17.3)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.2)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.26.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (4.1.1)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.47.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.21.6)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (14.0.1)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow) (57.4.0)
```

Figure A.1: Initialization



```
[ ]  import numpy as np
     import random
     from sklearn import utils
     mnist = tf.keras.datasets.mnist

     class DatasetManager:

       def __init__(self, trainingDatasetSizes = range(4,11), testDatasetSizes = range(80,101)):
         (self.x_train, self.y_train), (self.x_test, self.y_test) = mnist.load_data()
         #shuffle mnist images and labels
         self.x_train, self.y_train = utils.shuffle(self.x_train, self.y_train)
         self.x_test, self.y_test = utils.shuffle(self.x_test, self.y_test)
         self.x_train = self.x_train.reshape(self.x_train.shape[0], self.x_train.shape[1], self.x_train.shape[2], 1)
         self.x_test = self.x_test.reshape(self.x_test.shape[0], self.x_test.shape[1], self.x_test.shape[2], 1)
         self.x_train = self.x_train/255.0
         self.x_test = self.x_test/255.0

         ####mnist dataset separated by label numbers
         self.x_train_separated, self.y_train_separated, self.x_test_separated, self.y_test_separated = ([] for i in range(4))
         #list initializations
         for i in range(10):
           self.x_train_separated.append(list())
           self.y_train_separated.append(list())
           self.x_test_separated.append(list())
           self.y_test_separated.append(list())
         #list filling, element i includes all images/labels of number i, ex. x_train_separated[0] includes all images of 0
         for i in range(len(self.y_train)):
           self.x_train_separated[self.y_train[i]].append(self.x_train[i])
           self.y_train_separated[self.y_train[i]].append(self.y_train[i])
         for i in range(len(self.y_test)):
           self.x_test_separated[self.y_test[i]].append(self.x_test[i])
```

Figure A.2: Dataset manager module

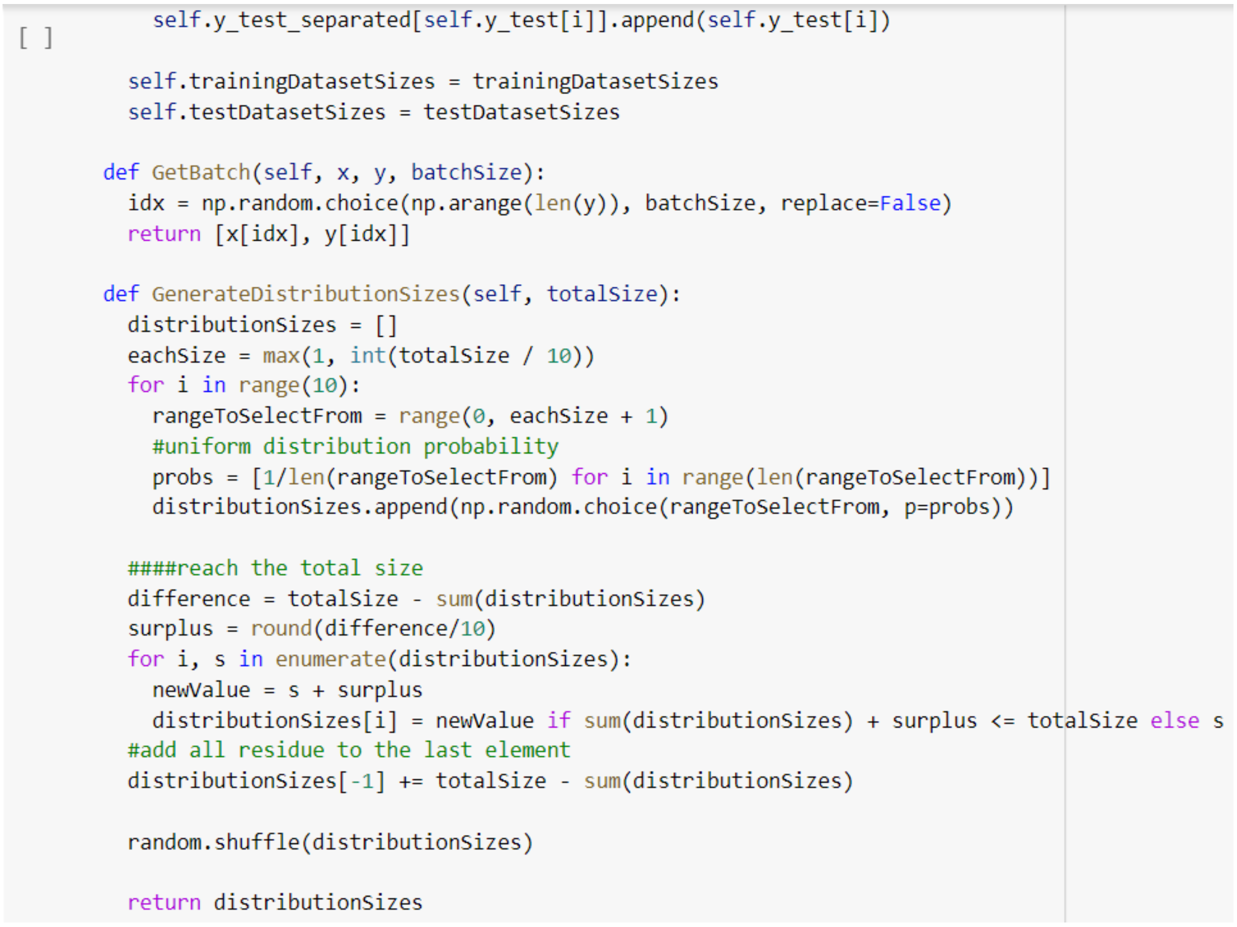
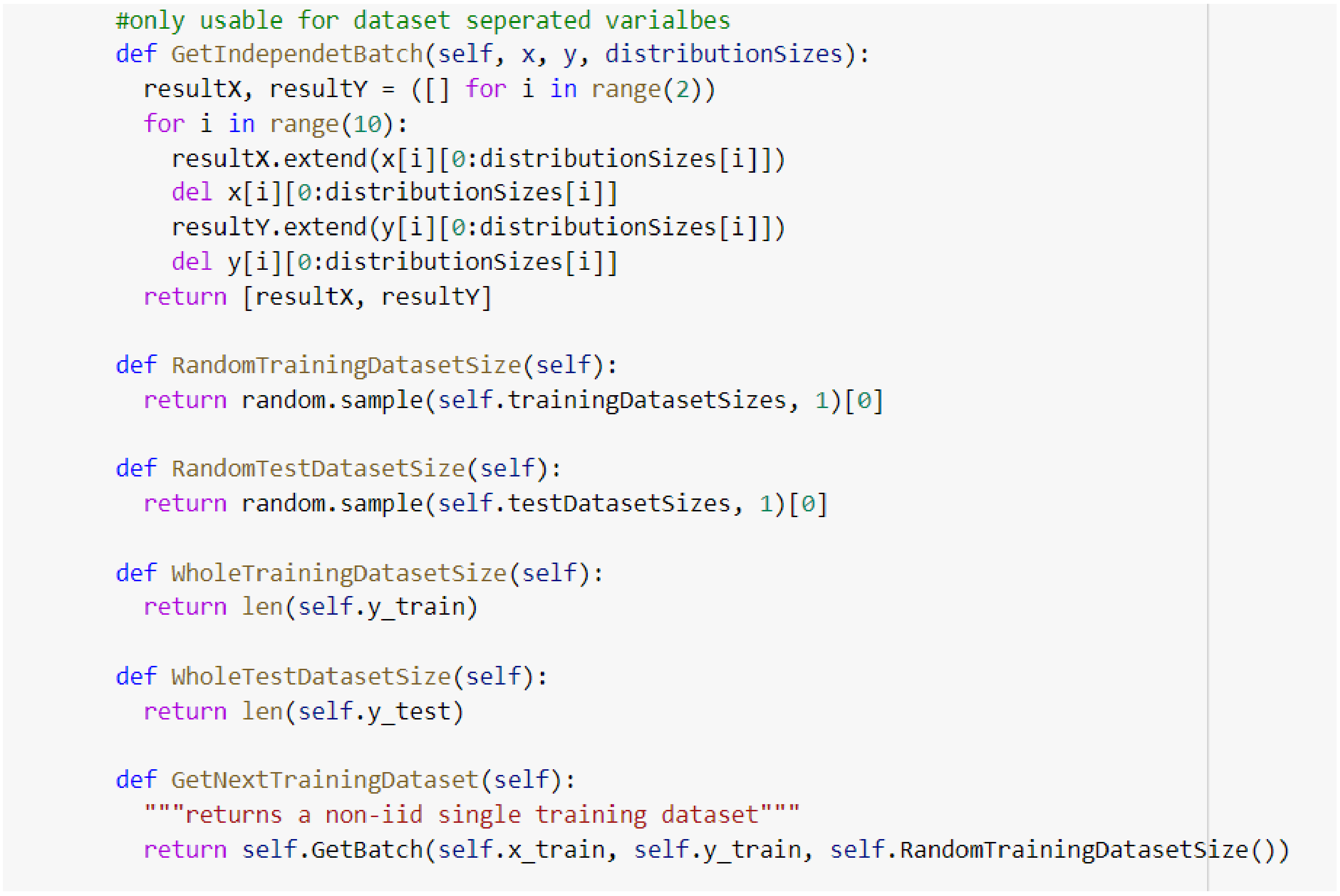
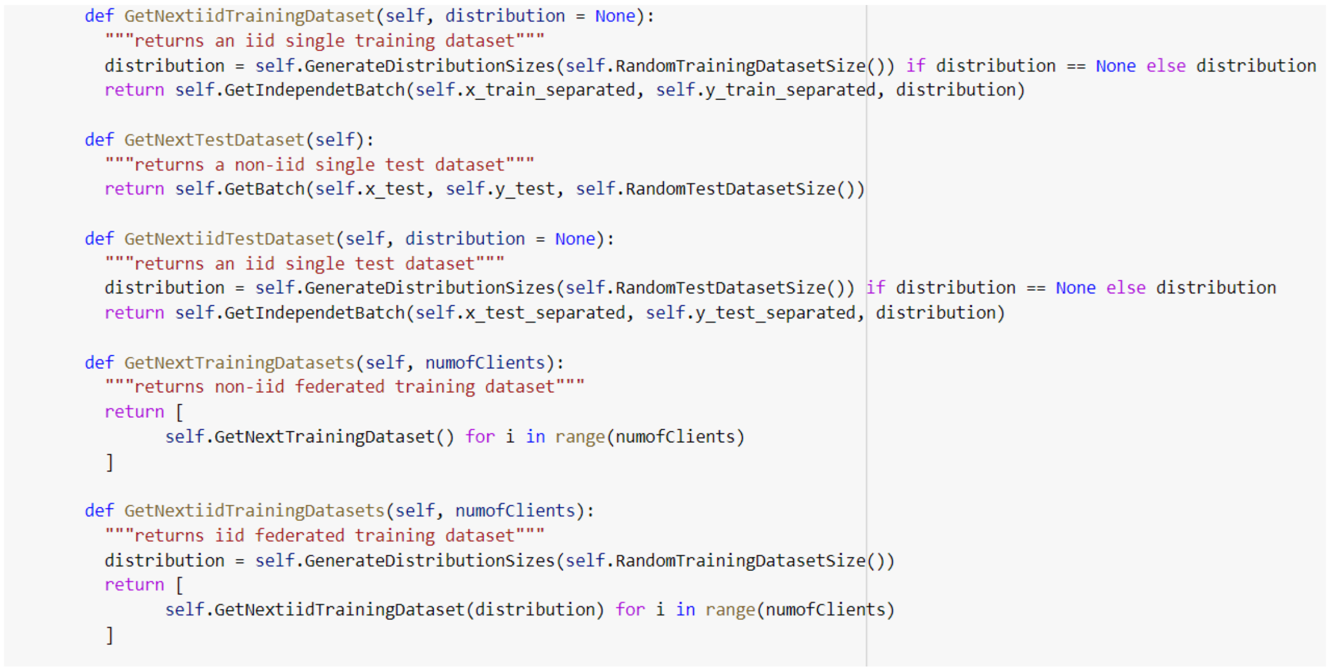

```
[ ]         self.y_test_separated[self.y_test[i]].append(self.y_test[i])

          self.trainingDatasetSizes = trainingDatasetSizes
          self.testDatasetSizes = testDatasetSizes

        def GetBatch(self, x, y, batchSize):
          idx = np.random.choice(np.arange(len(y)), batchSize, replace=False)
          return [x[idx], y[idx]]

        def GenerateDistributionSizes(self, totalSize):
          distributionSizes = []
          eachSize = max(1, int(totalSize / 10))
          for i in range(10):
            rangeToSelectFrom = range(0, eachSize + 1)
            #uniform distribution probability
            probs = [1/len(rangeToSelectFrom) for i in range(len(rangeToSelectFrom))]
            distributionSizes.append(np.random.choice(rangeToSelectFrom, p=probs))

          ####reach the total size
          difference = totalSize - sum(distributionSizes)
          surplus = round(difference/10)
          for i, s in enumerate(distributionSizes):
            newValue = s + surplus
            distributionSizes[i] = newValue if sum(distributionSizes) + surplus <= totalSize else s
          #add all residue to the last element
          distributionSizes[-1] += totalSize - sum(distributionSizes)

          random.shuffle(distributionSizes)

          return distributionSizes
```

Figure A.3: Dataset manager module



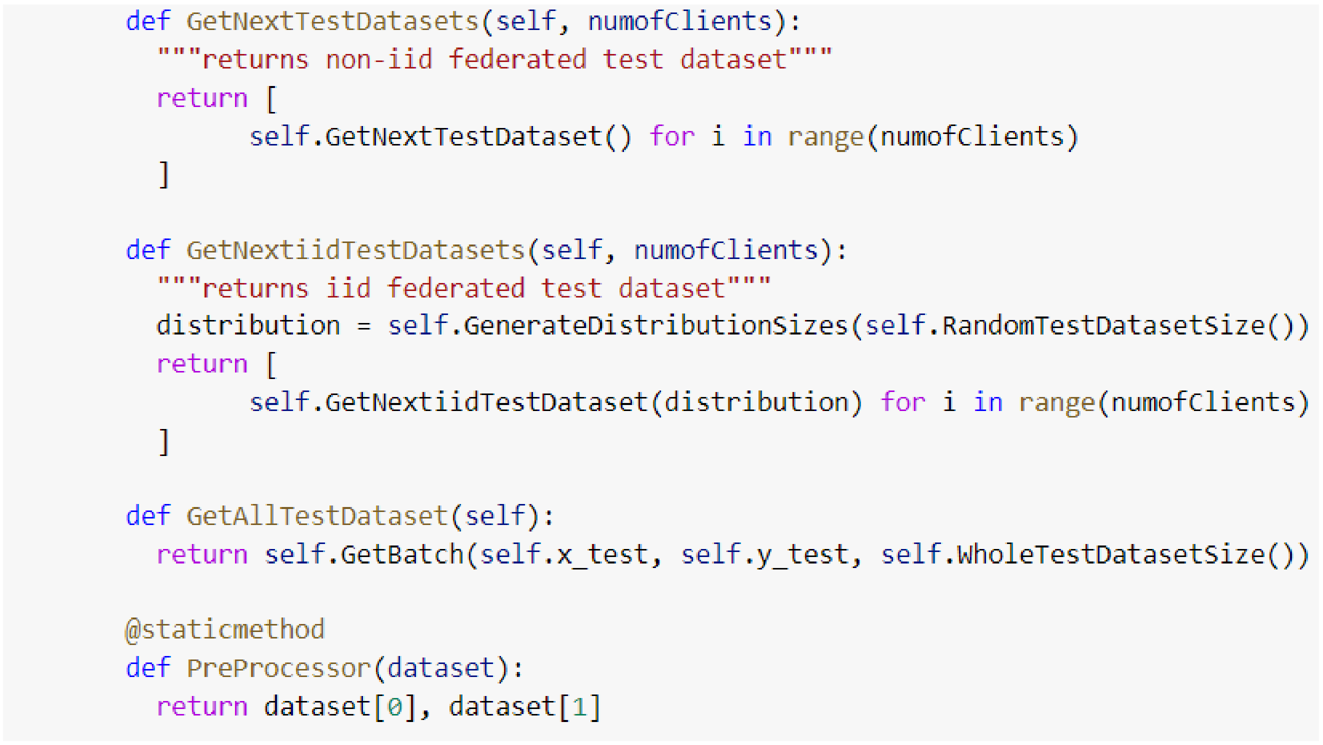
```
  #only usable for dataset seperated varialbes
  def GetIndependetBatch(self, x, y, distributionSizes):
    resultX, resultY = ([] for i in range(2))
    for i in range(10):
      resultX.extend(x[i][0:distributionSizes[i]])
      del x[i][0:distributionSizes[i]]
      resultY.extend(y[i][0:distributionSizes[i]])
      del y[i][0:distributionSizes[i]]
    return [resultX, resultY]

  def RandomTrainingDatasetSize(self):
    return random.sample(self.trainingDatasetSizes, 1)[0]

  def RandomTestDatasetSize(self):
    return random.sample(self.testDatasetSizes, 1)[0]

  def WholeTrainingDatasetSize(self):
    return len(self.y_train)

  def WholeTestDatasetSize(self):
    return len(self.y_test)

  def GetNextTrainingDataset(self):
    """returns a non-iid single training dataset"""
    return self.GetBatch(self.x_train, self.y_train, self.RandomTrainingDatasetSize())
```

Figure A.4: Dataset manager module



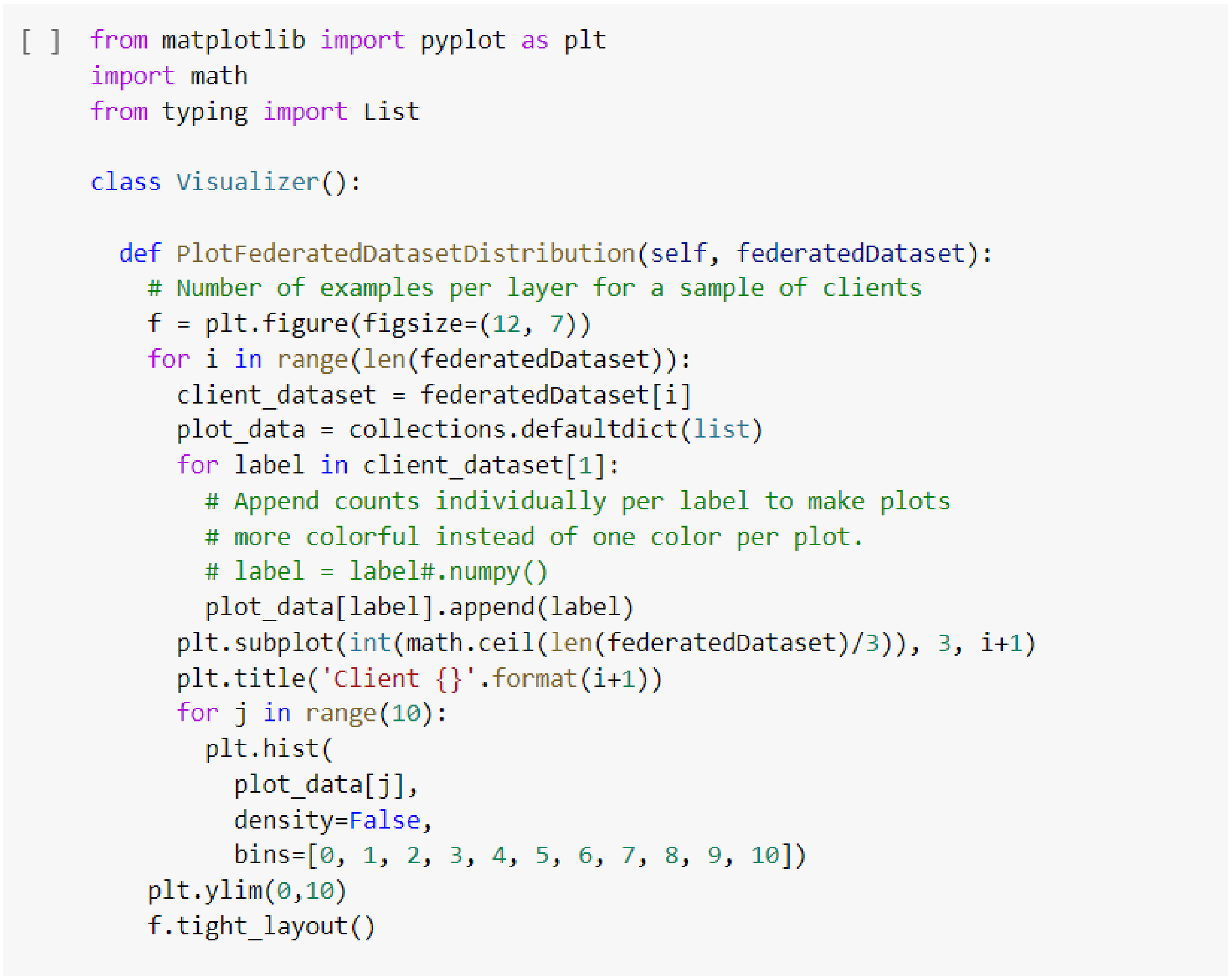
```
    def GetNextiidTrainingDataset(self, distribution = None):
      """returns an iid single training dataset"""
      distribution = self.GenerateDistributionSizes(self.RandomTrainingDatasetSize()) if distribution == None else distribution
      return self.GetIndependetBatch(self.x_train_separated, self.y_train_separated, distribution)

    def GetNextTestDataset(self):
      """returns a non-iid single test dataset"""
      return self.GetBatch(self.x_test, self.y_test, self.RandomTestDatasetSize())

    def GetNextiidTestDataset(self, distribution = None):
      """returns an iid single test dataset"""
      distribution = self.GenerateDistributionSizes(self.RandomTestDatasetSize()) if distribution == None else distribution
      return self.GetIndependetBatch(self.x_test_separated, self.y_test_separated, distribution)

    def GetNextTrainingDatasets(self, numofClients):
      """returns non-iid federated training dataset"""
      return [
            self.GetNextTrainingDataset() for i in range(numofClients)
      ]

    def GetNextiidTrainingDatasets(self, numofClients):
      """returns iid federated training dataset"""
      distribution = self.GenerateDistributionSizes(self.RandomTrainingDatasetSize())
      return [
            self.GetNextiidTrainingDataset(distribution) for i in range(numofClients)
      ]
```

Figure A.5: Dataset manager module



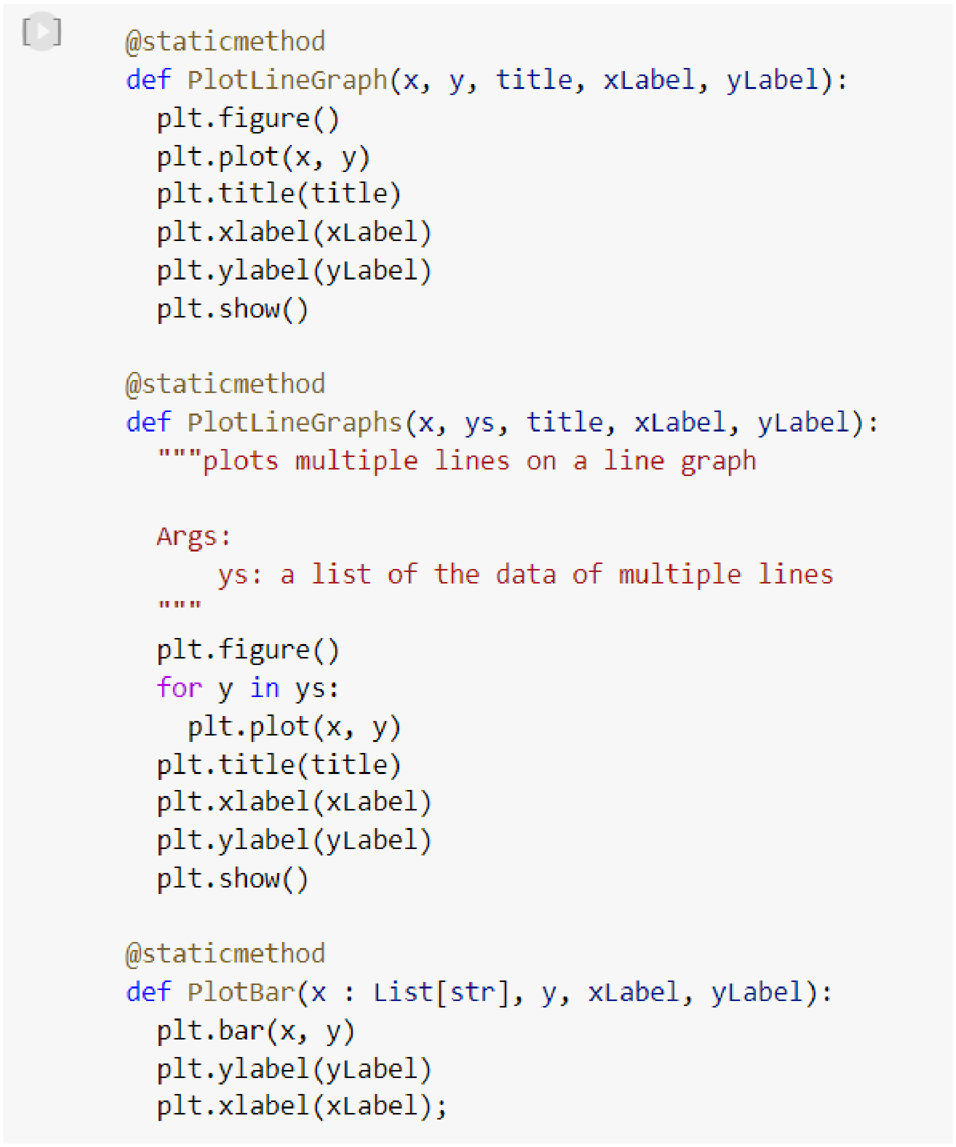
```python
def GetNextTestDatasets(self, numofClients):
  """returns non-iid federated test dataset"""
  return [
        self.GetNextTestDataset() for i in range(numofClients)
  ]

def GetNextiidTestDatasets(self, numofClients):
  """returns iid federated test dataset"""
  distribution = self.GenerateDistributionSizes(self.RandomTestDatasetSize())
  return [
        self.GetNextiidTestDataset(distribution) for i in range(numofClients)
  ]

def GetAllTestDataset(self):
  return self.GetBatch(self.x_test, self.y_test, self.WholeTestDatasetSize())

@staticmethod
def PreProcessor(dataset):
  return dataset[0], dataset[1]
```

Figure A.6: Dataset manager module



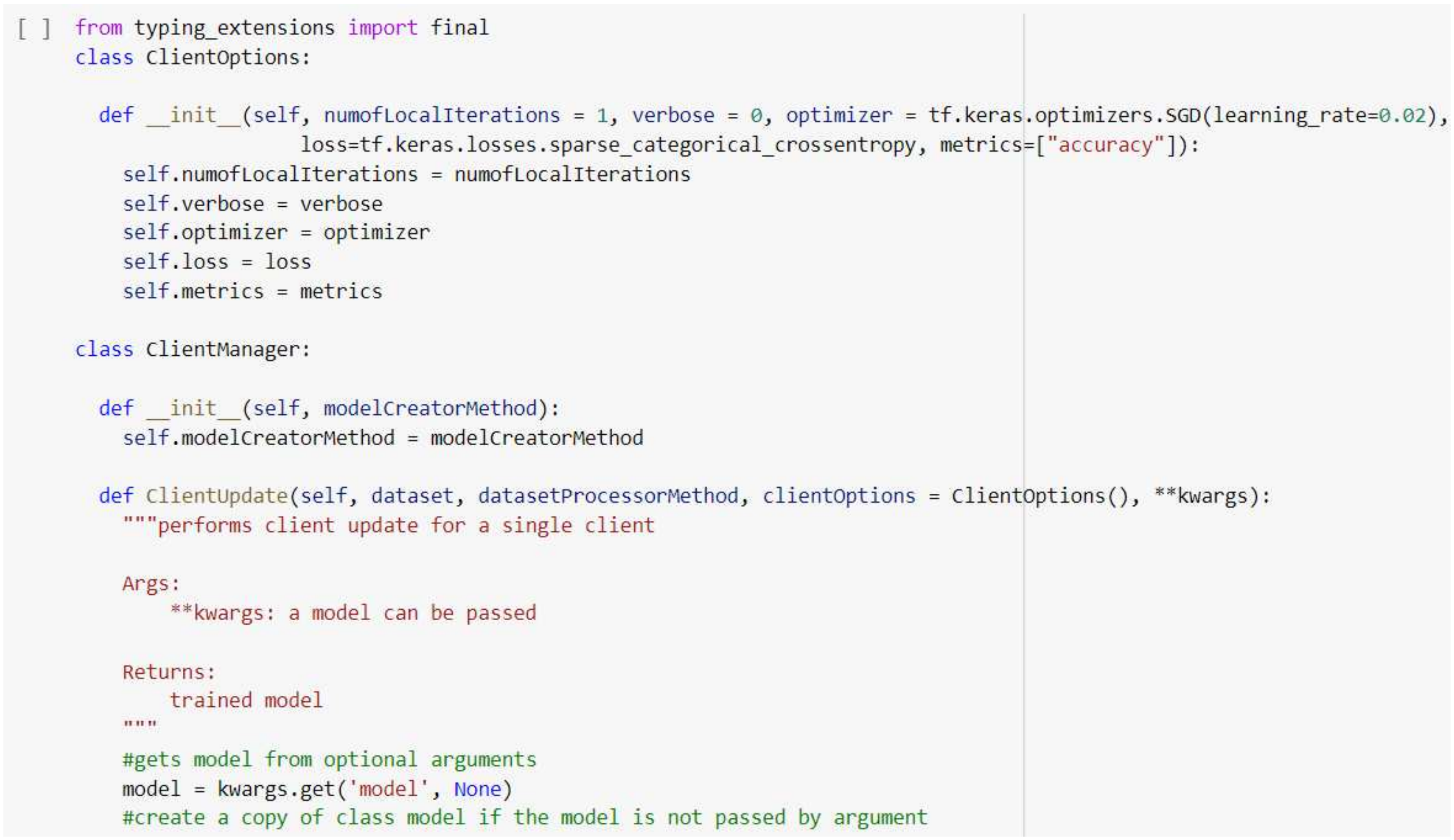
```python
from matplotlib import pyplot as plt
import math
from typing import List

class Visualizer():

  def PlotFederatedDatasetDistribution(self, federatedDataset):
    # Number of examples per layer for a sample of clients
    f = plt.figure(figsize=(12, 7))
    for i in range(len(federatedDataset)):
      client_dataset = federatedDataset[i]
      plot_data = collections.defaultdict(list)
      for label in client_dataset[1]:
        # Append counts individually per label to make plots
        # more colorful instead of one color per plot.
        # label = label#.numpy()
        plot_data[label].append(label)
      plt.subplot(int(math.ceil(len(federatedDataset)/3)), 3, i+1)
      plt.title('Client {}'.format(i+1))
      for j in range(10):
        plt.hist(
          plot_data[j],
          density=False,
          bins=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    plt.ylim(0,10)
    f.tight_layout()
```

Figure A.7: Visualizer sub-module



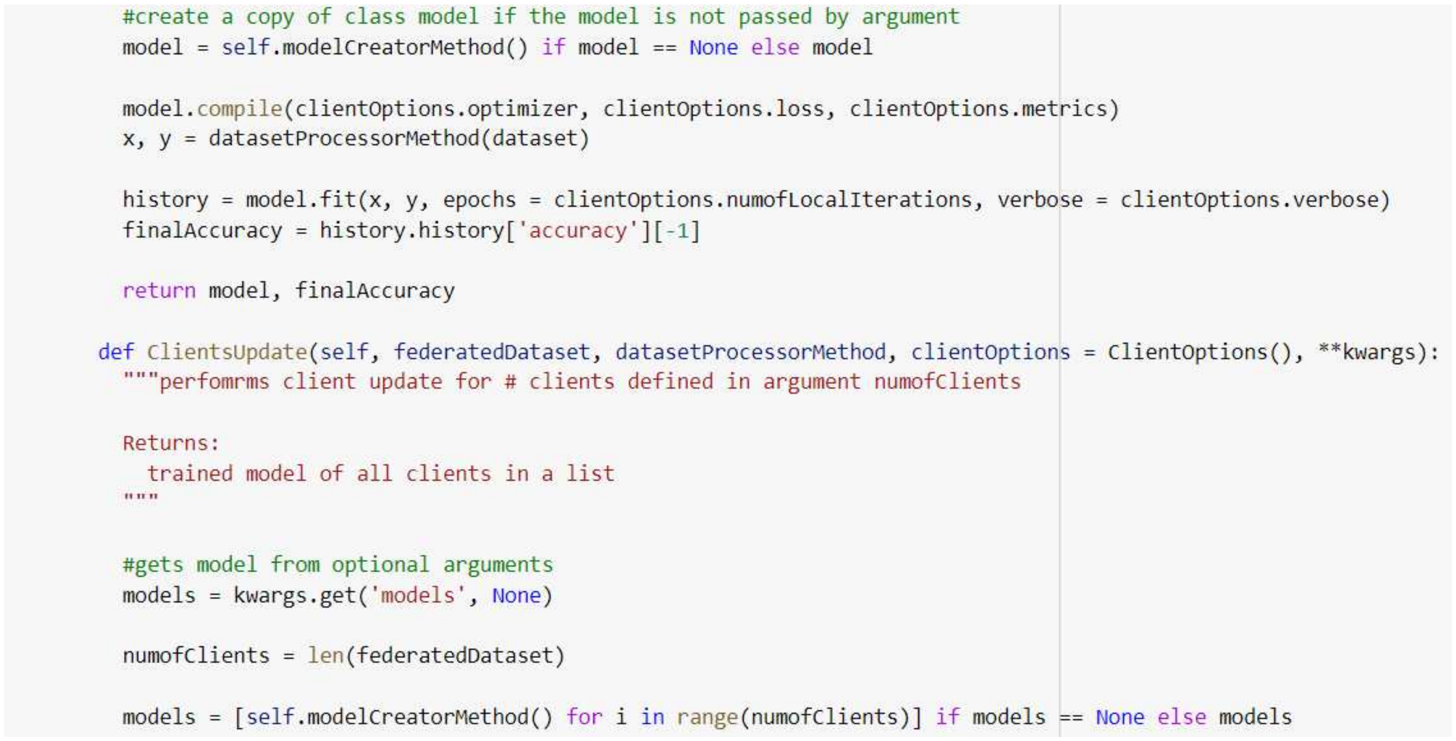
```python
@staticmethod
def PlotLineGraph(x, y, title, xLabel, yLabel):
  plt.figure()
  plt.plot(x, y)
  plt.title(title)
  plt.xlabel(xLabel)
  plt.ylabel(yLabel)
  plt.show()

@staticmethod
def PlotLineGraphs(x, ys, title, xLabel, yLabel):
  """plots multiple lines on a line graph

  Args:
      ys: a list of the data of multiple lines
  """
  plt.figure()
  for y in ys:
    plt.plot(x, y)
  plt.title(title)
  plt.xlabel(xLabel)
  plt.ylabel(yLabel)
  plt.show()

@staticmethod
def PlotBar(x : List[str], y, xLabel, yLabel):
  plt.bar(x, y)
  plt.ylabel(yLabel)
  plt.xlabel(xLabel);
```

Figure A.8: Visualizer sub-module



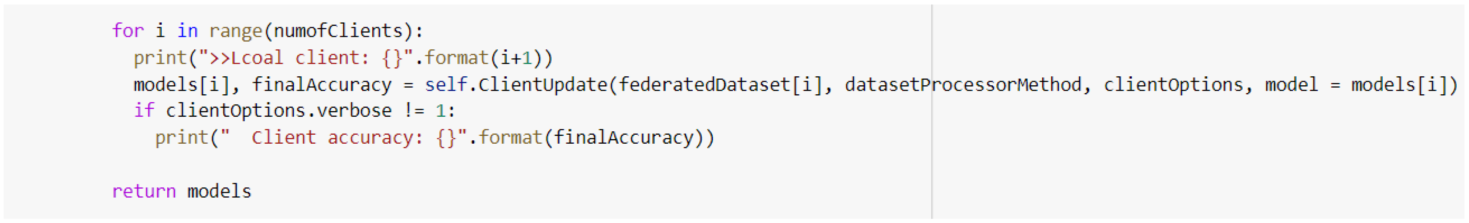
```
[ ]  from typing_extensions import final
     class ClientOptions:

       def __init__(self, numofLocalIterations = 1, verbose = 0, optimizer = tf.keras.optimizers.SGD(learning_rate=0.02),
                        loss=tf.keras.losses.sparse_categorical_crossentropy, metrics=["accuracy"]):
         self.numofLocalIterations = numofLocalIterations
         self.verbose = verbose
         self.optimizer = optimizer
         self.loss = loss
         self.metrics = metrics

     class ClientManager:

       def __init__(self, modelCreatorMethod):
         self.modelCreatorMethod = modelCreatorMethod

       def ClientUpdate(self, dataset, datasetProcessorMethod, clientOptions = ClientOptions(), **kwargs):
         """performs client update for a single client

         Args:
             **kwargs: a model can be passed

         Returns:
             trained model
         """
         #gets model from optional arguments
         model = kwargs.get('model', None)
         #create a copy of class model if the model is not passed by argument
```

Figure A.9: Client manager module



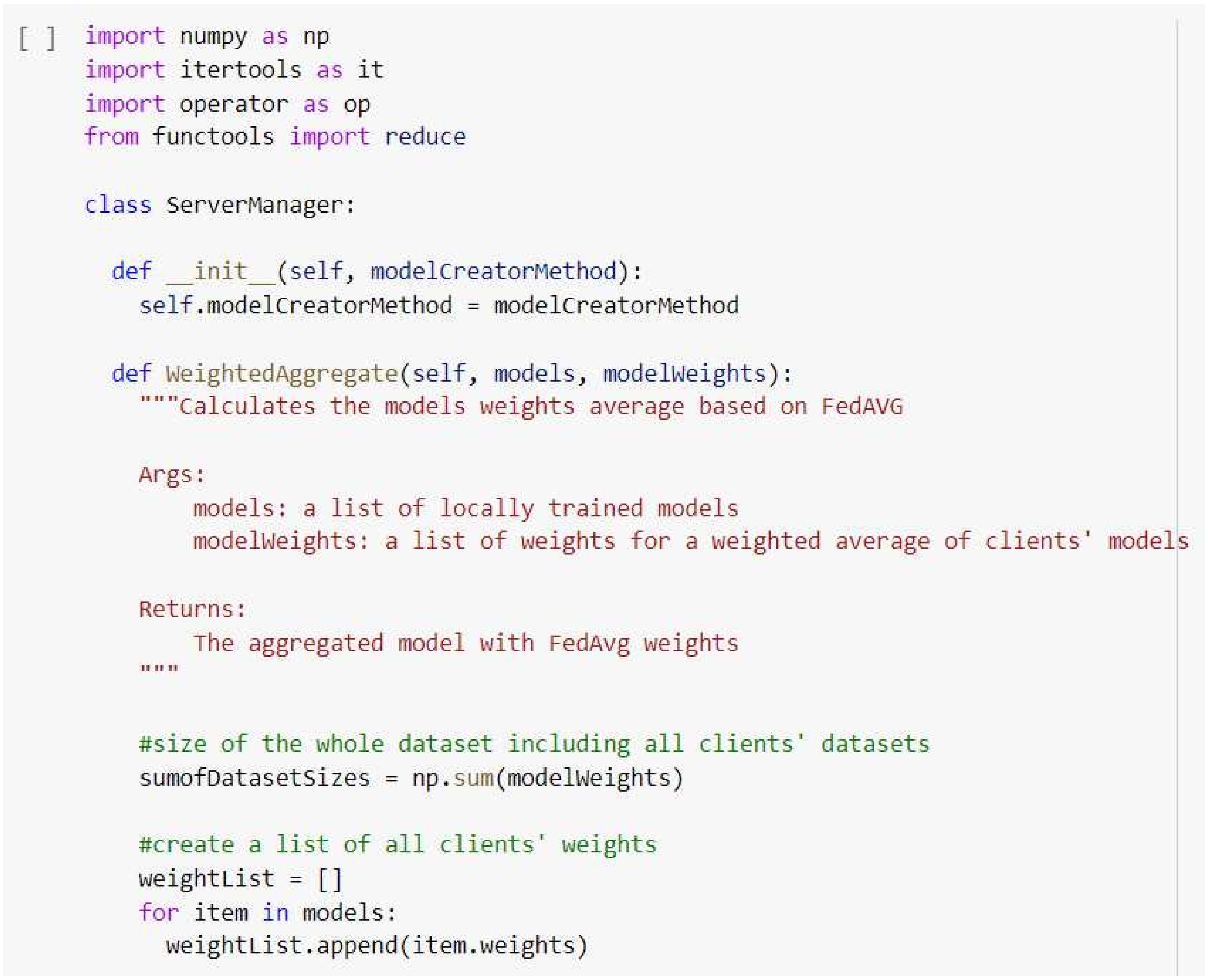
```
    #create a copy of class model if the model is not passed by argument
    model = self.modelCreatorMethod() if model == None else model

    model.compile(clientOptions.optimizer, clientOptions.loss, clientOptions.metrics)
    x, y = datasetProcessorMethod(dataset)

    history = model.fit(x, y, epochs = clientOptions.numofLocalIterations, verbose = clientOptions.verbose)
    finalAccuracy = history.history['accuracy'][-1]

    return model, finalAccuracy

  def ClientsUpdate(self, federatedDataset, datasetProcessorMethod, clientOptions = ClientOptions(), **kwargs):
    """perfomrms client update for # clients defined in argument numofClients

    Returns:
      trained model of all clients in a list
    """

    #gets model from optional arguments
    models = kwargs.get('models', None)

    numofClients = len(federatedDataset)

    models = [self.modelCreatorMethod() for i in range(numofClients)] if models == None else models
```

Figure A.10: Client manager module



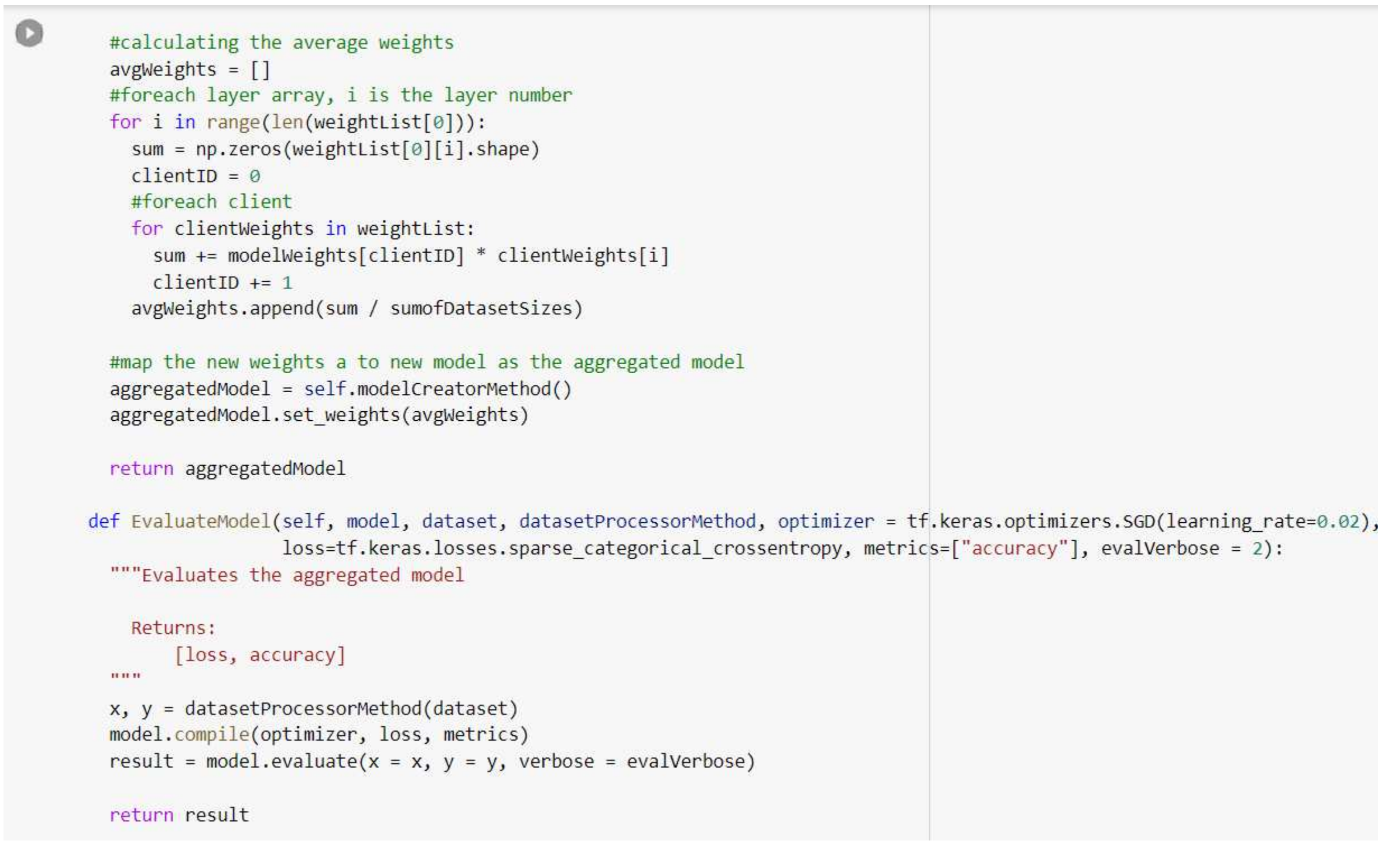
```
    for i in range(numofClients):
      print(">>Lcoal client: {}".format(i+1))
      models[i], finalAccuracy = self.ClientUpdate(federatedDataset[i], datasetProcessorMethod, clientOptions, model = models[i])
      if clientOptions.verbose != 1:
        print("  Client accuracy: {}".format(finalAccuracy))

    return models
```

Figure A.11: Client manager module



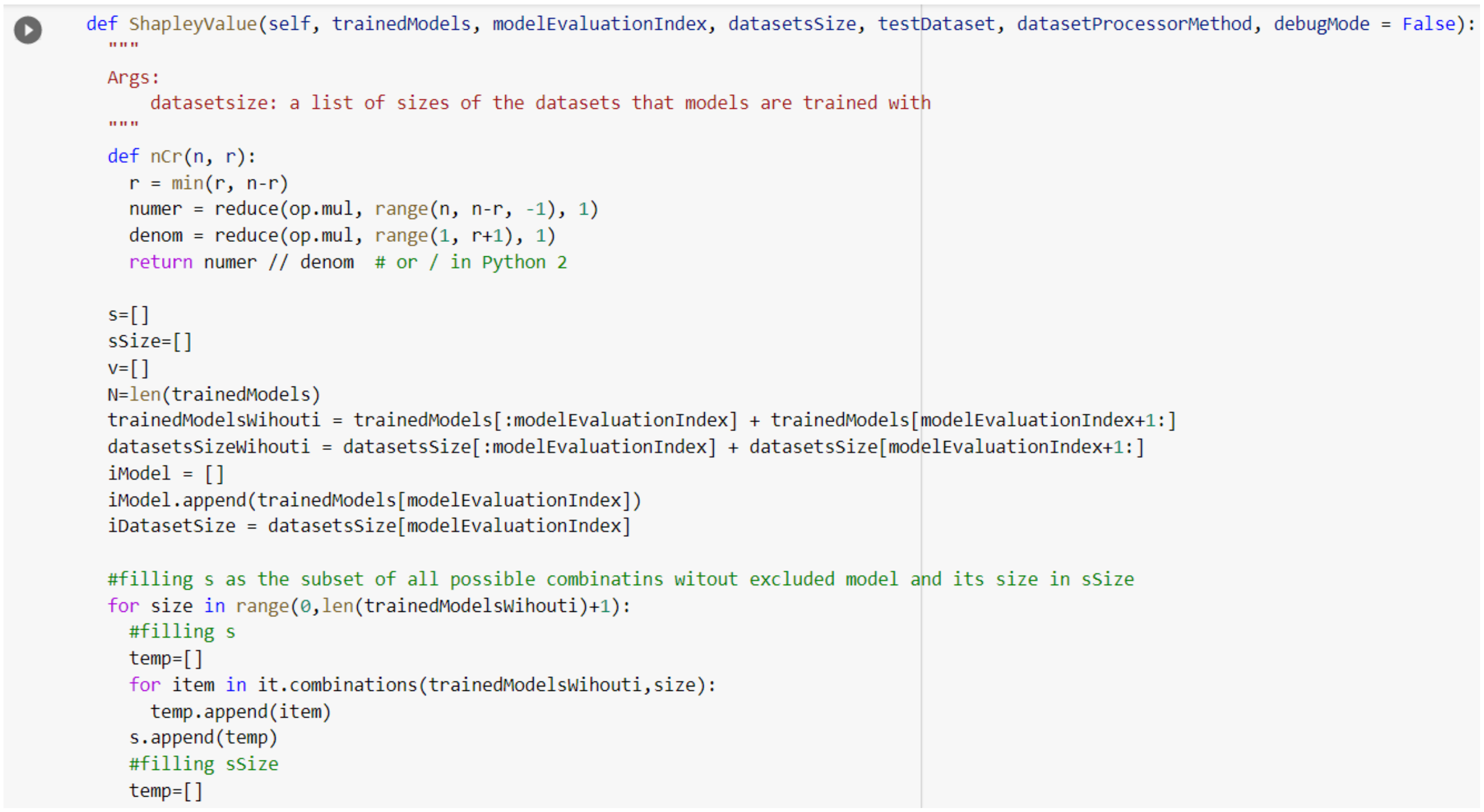
```
[ ]  import numpy as np
     import itertools as it
     import operator as op
     from functools import reduce

     class ServerManager:

       def __init__(self, modelCreatorMethod):
         self.modelCreatorMethod = modelCreatorMethod

       def WeightedAggregate(self, models, modelWeights):
         """Calculates the models weights average based on FedAVG

         Args:
             models: a list of locally trained models
             modelWeights: a list of weights for a weighted average of clients' models

         Returns:
             The aggregated model with FedAvg weights
         """

         #size of the whole dataset including all clients' datasets
         sumofDatasetSizes = np.sum(modelWeights)

         #create a list of all clients' weights
         weightList = []
         for item in models:
           weightList.append(item.weights)
```

Figure A.12: Server manager module



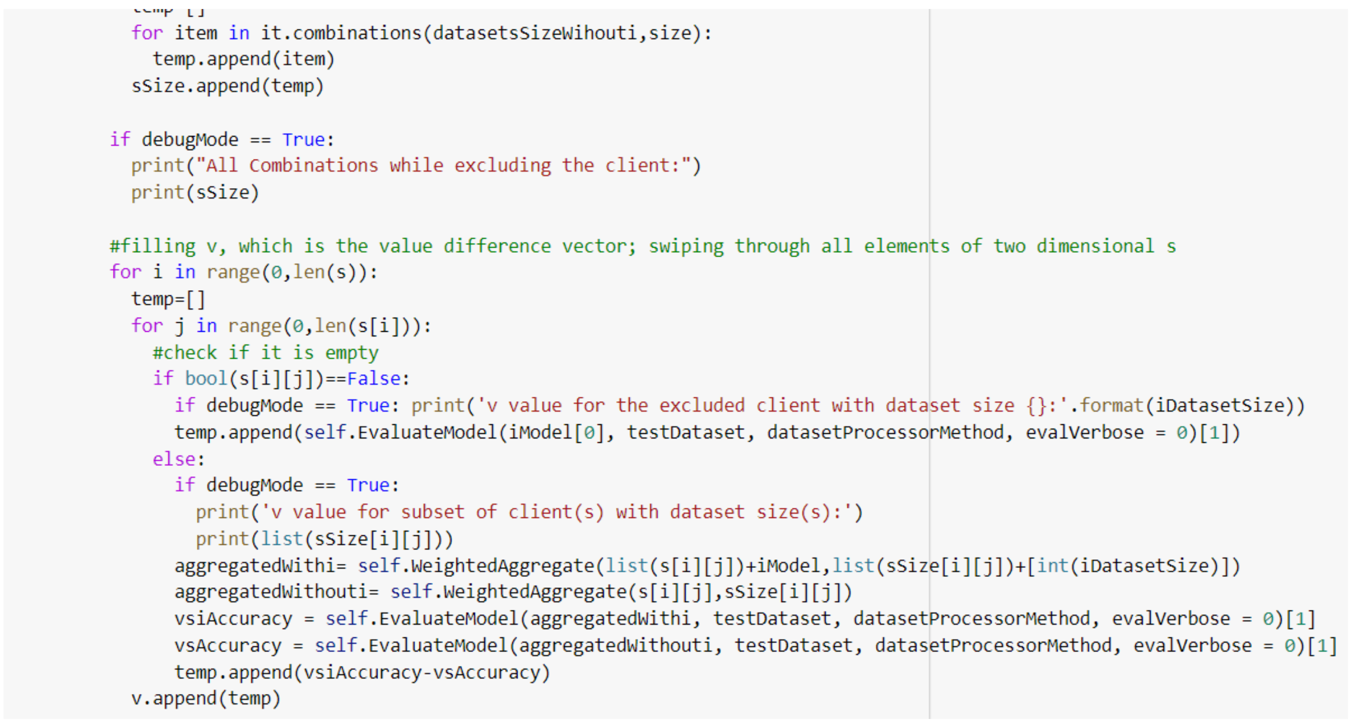
```
  #calculating the average weights
  avgWeights = []
  #foreach layer array, i is the layer number
  for i in range(len(weightList[0])):
    sum = np.zeros(weightList[0][i].shape)
    clientID = 0
    #foreach client
    for clientWeights in weightList:
      sum += modelWeights[clientID] * clientWeights[i]
      clientID += 1
    avgWeights.append(sum / sumofDatasetSizes)

  #map the new weights a to new model as the aggregated model
  aggregatedModel = self.modelCreatorMethod()
  aggregatedModel.set_weights(avgWeights)

  return aggregatedModel

def EvaluateModel(self, model, dataset, datasetProcessorMethod, optimizer = tf.keras.optimizers.SGD(learning_rate=0.02),
                  loss=tf.keras.losses.sparse_categorical_crossentropy, metrics=["accuracy"], evalVerbose = 2):
  """Evaluates the aggregated model

    Returns:
        [loss, accuracy]
  """
  x, y = datasetProcessorMethod(dataset)
  model.compile(optimizer, loss, metrics)
  result = model.evaluate(x = x, y = y, verbose = evalVerbose)

  return result
```

Figure A.13: Server manager module



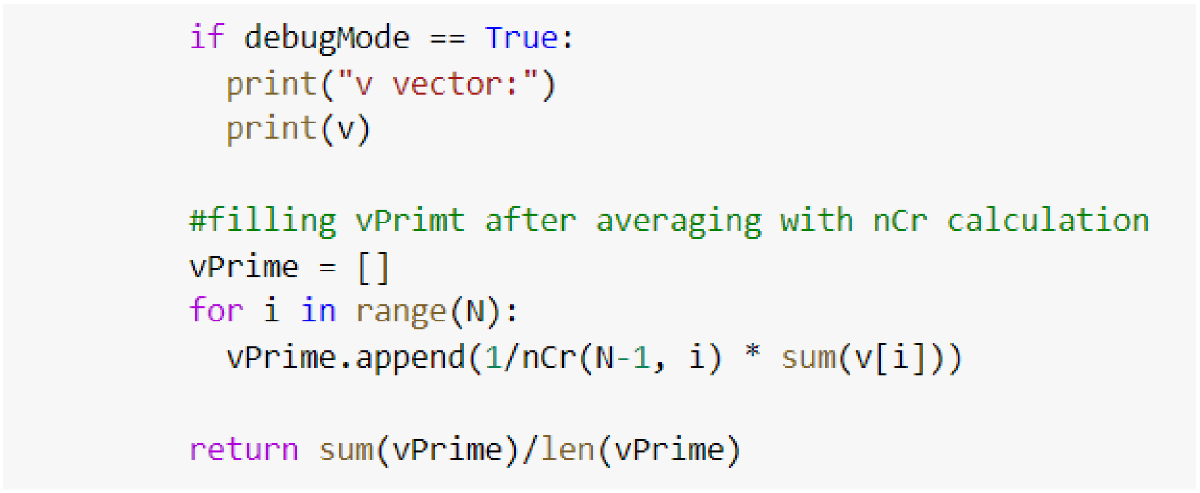
```
def ShapleyValue(self, trainedModels, modelEvaluationIndex, datasetsSize, testDataset, datasetProcessorMethod, debugMode = False):
  """
  Args:
      datasetsize: a list of sizes of the datasets that models are trained with
  """
  def nCr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer // denom  # or / in Python 2

  s=[]
  sSize=[]
  v=[]
  N=len(trainedModels)
  trainedModelsWihouti = trainedModels[:modelEvaluationIndex] + trainedModels[modelEvaluationIndex+1:]
  datasetsSizeWihouti = datasetsSize[:modelEvaluationIndex] + datasetsSize[modelEvaluationIndex+1:]
  iModel = []
  iModel.append(trainedModels[modelEvaluationIndex])
  iDatasetSize = datasetsSize[modelEvaluationIndex]

  #filling s as the subset of all possible combinatins witout excluded model and its size in sSize
  for size in range(0,len(trainedModelsWihouti)+1):
    #filling s
    temp=[]
    for item in it.combinations(trainedModelsWihouti,size):
      temp.append(item)
    s.append(temp)
    #filling sSize
    temp=[]
```

Figure A.14: Server manager module



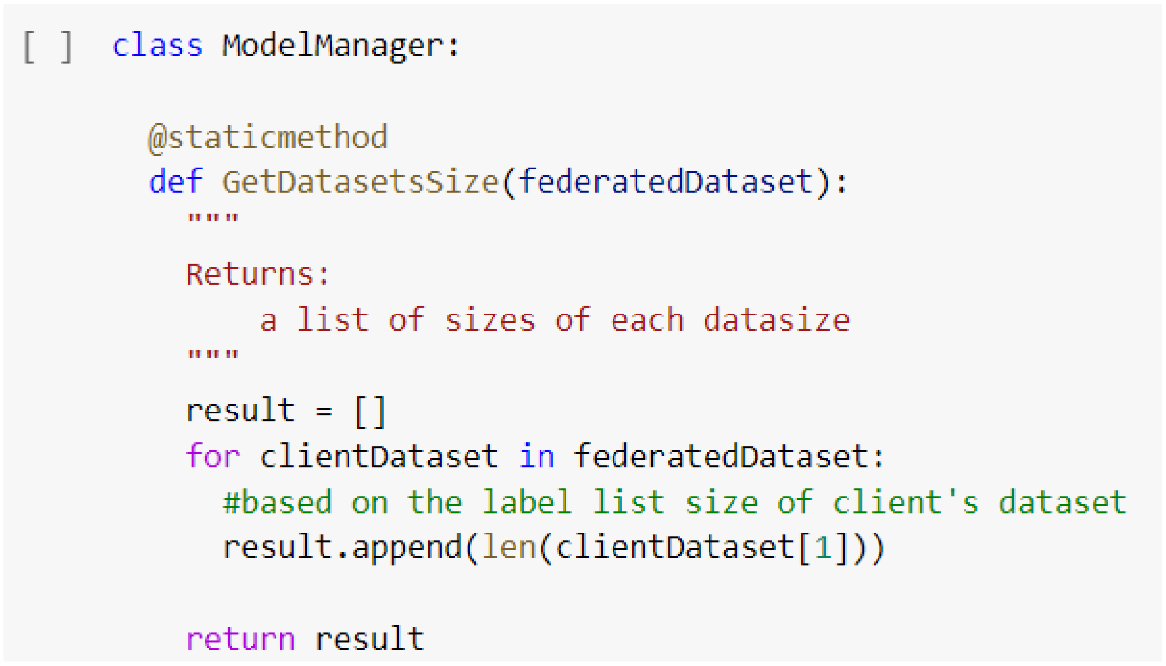
```
  for item in it.combinations(datasetsSizeWihouti,size):
    temp.append(item)
  sSize.append(temp)

if debugMode == True:
  print("All Combinations while excluding the client:")
  print(sSize)

#filling v, which is the value difference vector; swiping through all elements of two dimensional s
for i in range(0,len(s)):
  temp=[]
  for j in range(0,len(s[i])):
    #check if it is empty
    if bool(s[i][j])==False:
      if debugMode == True: print('v value for the excluded client with dataset size {}:'.format(iDatasetSize))
      temp.append(self.EvaluateModel(iModel[0], testDataset, datasetProcessorMethod, evalVerbose = 0)[1])
    else:
      if debugMode == True:
        print('v value for subset of client(s) with dataset size(s):')
        print(list(sSize[i][j]))
      aggregatedWithi= self.WeightedAggregate(list(s[i][j])+iModel,list(sSize[i][j])+[int(iDatasetSize)])
      aggregatedWithouti= self.WeightedAggregate(s[i][j],sSize[i][j])
      vsiAccuracy = self.EvaluateModel(aggregatedWithi, testDataset, datasetProcessorMethod, evalVerbose = 0)[1]
      vsAccuracy = self.EvaluateModel(aggregatedWithouti, testDataset, datasetProcessorMethod, evalVerbose = 0)[1]
      temp.append(vsiAccuracy-vsAccuracy)
  v.append(temp)
```

Figure A.15: Server manager module



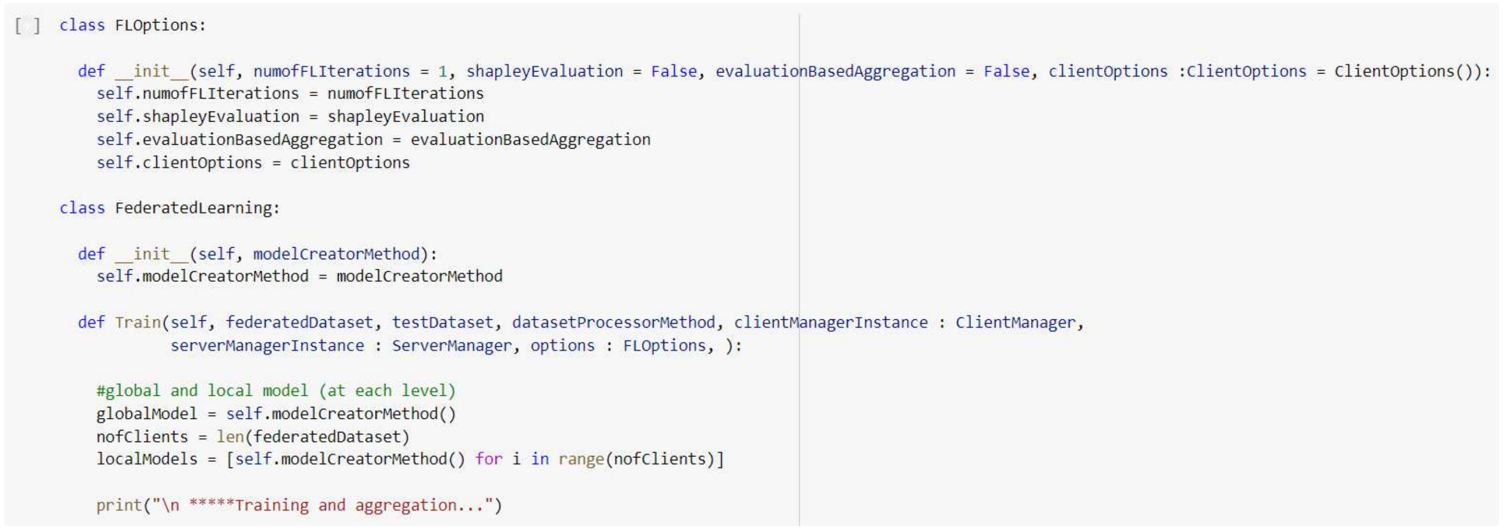
```
if debugMode == True:
  print("v vector:")
  print(v)

#filling vPrimt after averaging with nCr calculation
vPrime = []
for i in range(N):
  vPrime.append(1/nCr(N-1, i) * sum(v[i]))

return sum(vPrime)/len(vPrime)
```

Figure A.16: Server manager module



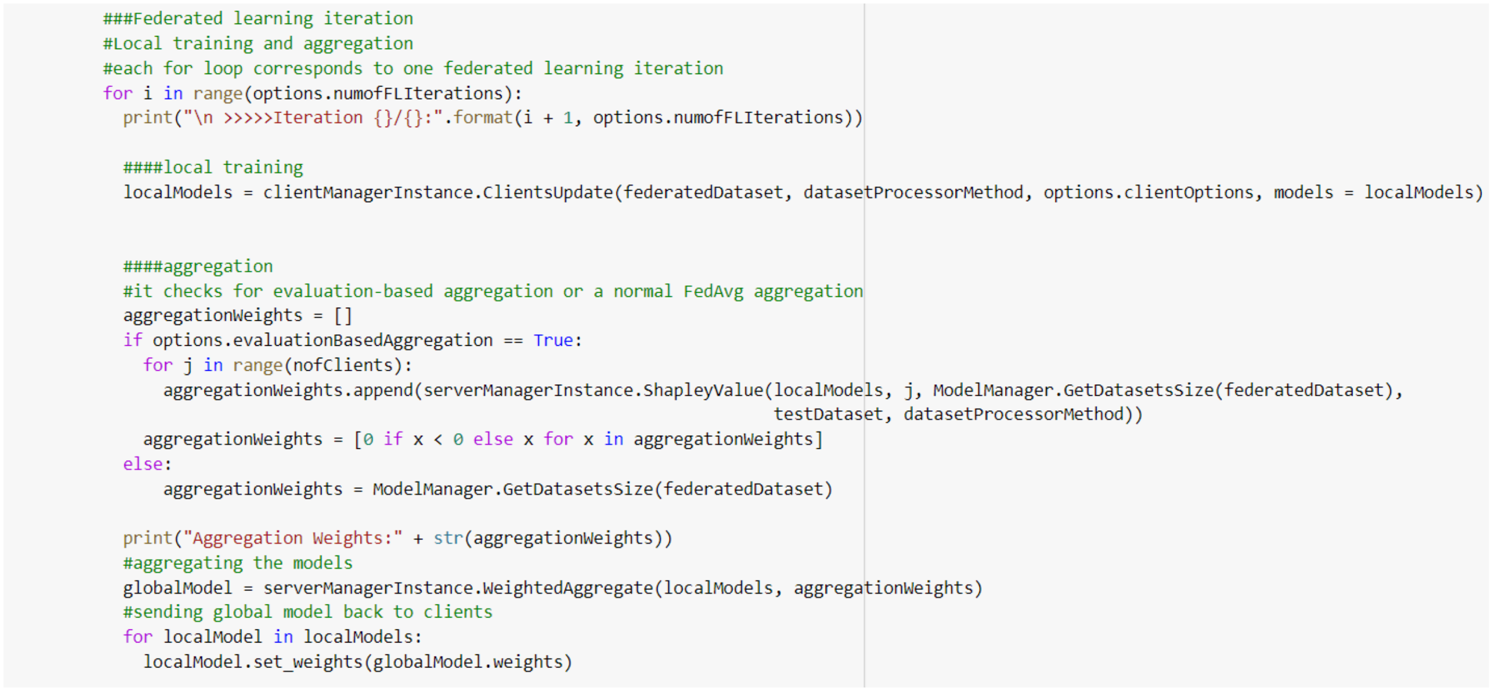
```
[ ]  class ModelManager:

       @staticmethod
       def GetDatasetsSize(federatedDataset):
         """
         Returns:
             a list of sizes of each datasize
         """
         result = []
         for clientDataset in federatedDataset:
           #based on the label list size of client's dataset
           result.append(len(clientDataset[1]))

         return result
```

Figure A.17: Model manager module



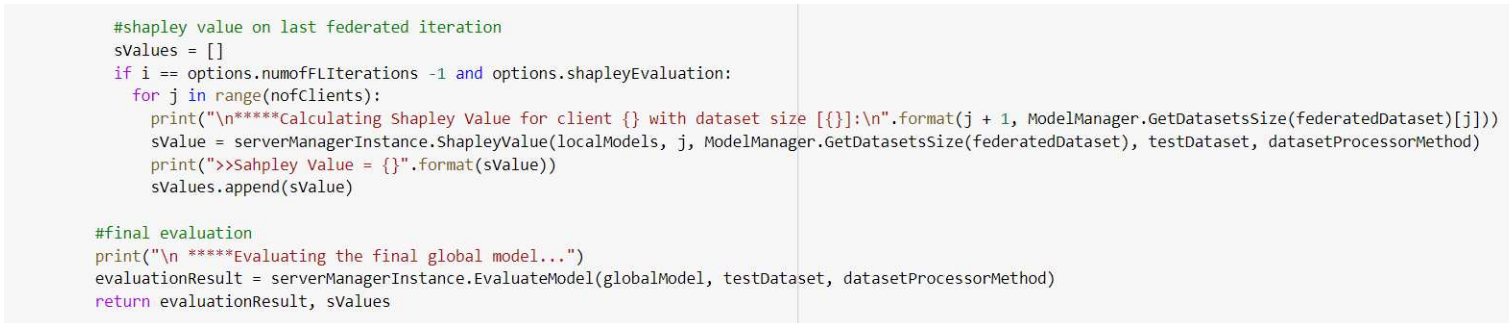
```
[ ]  class FLOptions:

       def __init__(self, numofFLIterations = 1, shapleyEvaluation = False, evaluationBasedAggregation = False, clientOptions :ClientOptions = ClientOptions()):
         self.numofFLIterations = numofFLIterations
         self.shapleyEvaluation = shapleyEvaluation
         self.evaluationBasedAggregation = evaluationBasedAggregation
         self.clientOptions = clientOptions

     class FederatedLearning:

       def __init__(self, modelCreatorMethod):
         self.modelCreatorMethod = modelCreatorMethod

       def Train(self, federatedDataset, testDataset, datasetProcessorMethod, clientManagerInstance : ClientManager,
               serverManagerInstance : ServerManager, options : FLOptions, ):

         #global and local model (at each level)
         globalModel = self.modelCreatorMethod()
         nofClients = len(federatedDataset)
         localModels = [self.modelCreatorMethod() for i in range(nofClients)]

         print("\n *****Training and aggregation...")
```

Figure A.18: Federated learning module



```
###Federated learning iteration
#Local training and aggregation
#each for loop corresponds to one federated learning iteration
for i in range(options.numofFLIterations):
  print("\n >>>>>Iteration {}/{}:".format(i + 1, options.numofFLIterations))

  ####local training
  localModels = clientManagerInstance.ClientsUpdate(federatedDataset, datasetProcessorMethod, options.clientOptions, models = localModels)


  ####aggregation
  #it checks for evaluation-based aggregation or a normal FedAvg aggregation
  aggregationWeights = []
  if options.evaluationBasedAggregation == True:
    for j in range(nofClients):
      aggregationWeights.append(serverManagerInstance.ShapleyValue(localModels, j, ModelManager.GetDatasetsSize(federatedDataset),
                                                                   testDataset, datasetProcessorMethod))
    aggregationWeights = [0 if x < 0 else x for x in aggregationWeights]
  else:
      aggregationWeights = ModelManager.GetDatasetsSize(federatedDataset)

  print("Aggregation Weights:" + str(aggregationWeights))
  #aggregating the models
  globalModel = serverManagerInstance.WeightedAggregate(localModels, aggregationWeights)
  #sending global model back to clients
  for localModel in localModels:
    localModel.set_weights(globalModel.weights)
```

Figure A.19: Federated learning module



```
  #shapley value on last federated iteration
  sValues = []
  if i == options.numofFLIterations -1 and options.shapleyEvaluation:
    for j in range(nofClients):
      print("\n*****Calculating Shapley Value for client {} with dataset size [{}]:\n".format(j + 1, ModelManager.GetDatasetsSize(federatedDataset)[j]))
      sValue = serverManagerInstance.ShapleyValue(localModels, j, ModelManager.GetDatasetsSize(federatedDataset), testDataset, datasetProcessorMethod)
      print(">>Sahpley Value = {}".format(sValue))
      sValues.append(sValue)

#final evaluation
print("\n *****Evaluating the final global model...")
evaluationResult = serverManagerInstance.EvaluateModel(globalModel, testDataset, datasetProcessorMethod)
return evaluationResult, sValues
```

Figure A.20: Federated learning module