UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Souvik Paul

Energy-efficient Federated Learning for Data Analytics in Fog Networks

Master's Thesis (30 ECTS)

Supervisor:	Dr. Mainak Adhikari, PhD
Co Supervisor:	Dr. Satish Narayana Srirama, PhD

Tartu 2021

Energy-efficient Federated Learning for Data Analytics in Fog Networks

Abstract:

Federated Learning(FL) is a collaborative and distributed machine learning technique that enables training over many clients without sharing the client's data. The advent of a massive number of low-powered Internet of Things (IoT) devices and local fog devices with sufficient computational power have made it possible to take advantage of this distributed framework in real-life scenarios. However, the standard IoT-enabled fog framework suffers from significant energy expense due to the intercommunication between the computing devices. The existing state-of-the-art strategies have proposed altering the core architecture to reduce energy expenses that work only under ideal conditions on independent and identically distributed (IID) data. Nevertheless, the vast deployment of low-cost sensor devices in use cases like Smart Agriculture makes it impossible for such ideal conditions to prevail in real life. Motivated by the above-mentioned challenges, in this thesis, an energy-efficient fog framework for smart irrigation is proposed to mitigate these issues. The proposed algorithm utilizes data sampling and optimal resource provisioning methodologies to maximize resource utilization, which results in a significant energy reduction in the framework. Besides that, the local gateway devices of the proposed fog framework serve as functional units based on redundant data filtering, outlier removal, and lossy data aggregation to minimize data transmission. The analysis of this proposed model is done by training on data from agricultural field sensors using a data simulator to predict irrigation requirements. From the simulation results, it is observed that the proposed algorithm reduces the total energy consumption by 51.5% and 15.2% compared with Split Learning(SL) and standard FL, respectively, while achieving the prediction accuracy of 91.1%.

Keywords:
Federated Learning, Fog Computing, Internet of Things, Data Aggregation, Resource provisioning.

CERCS: P170 Computer science, numerical analysis, systems, control

Energiatõhus liitõpe andmeanalüütika jaoks udutöötlusvõrkudes

Lühikokkuvõte: Liitõpe (ingl Federated Learning) on koostööl põhinev hajutatud masinõppe tehnika, mis võimaldab treenimist kasutades mitmeid kliente, ilma klientide andmeid jagamata. Suure hulga energiasäästlike Asjade Interneti (ingl Internet of Things, IoT) seadmete ja piisava arvutusvõimsusega kohalike uduseadmete teke võimaldab taolist hajusraamistikku reaalsetes stsenaariumites ära kasutada. Ent standardse IoT-toega uduraamistiku puudujäägiks on seadmete omavahelisest suhtlusest tingitud märkimisväärne energiakulu. Olemasolevad strateegiad põhiarhitektuuri energiatõhususe tõstmiseks toimivad ainult ideaalsetes tingimustes, andmestikega mis põhinevad sõltumatuil ning identsete jaotustega (ingl Independent and Identically Distributed, lüh IID) muutujatel. Paraku muudab odavate andurseadmete laialdane kasutuselevõtt valdkondades nagu nutikas põllumajandus selliste ideaaltingimuste tegelikus elus kehtimise võimatuks. Nimetatud väljakutsetest ajendatuna esitleb käesolev lõputöö probleemide leevendamiseks energiatõhusat udutöötluse liitraamistik nutikaks irrigatsiooniks. Pakutud strateegia kasutab andmete diskreetimist ja optimaalset ressursside varustamise metoodikat ressursikasutuse maksimeerimiseks, mille tulemuseks on märkimisväärne energiakasutuse vähenemine raamistikus. Lisaks toimivad pakutava udutöötluse liitraamistiku kohalikud lüüsseadmed funktsionaalsete üksustena, teostades ülemääraste andmete filtreerimist, erindite eemaldust ja kadudega andmete agregeerimist, et minimeerida andmeedastust. Mudeli analüüs tehakse põllundusandurite andmetel treenimise põhjal, kasutades irrigatsiooninõuete prognoosimiseks andmesimulaatorit. Simulatsioonitulemuste põhjal täheldati, et esitletud strateegia vähendab kogu energiatarbimist võrreldes jagatud õppimise (ingl Split Learning) ja tavalise liitõppega vastavalt 51,5 % ja 15,2 %, saavutades samas prognoosimise täpsuse 91,1%.

Võtmesõnad:
Liitõpe, Udutöötlus, Asjade Internet, Andmete agregeerimine, ressursside varustamine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
CSV	Comma Separated Value
DL	Deep Learning
DoA	Degree of Aggregation
FaaS	Function as a Service
FedAvg	Federated Averaging
FedSGD	Federated Stochastic Gradient Decent
FL	Federated Learning
IID	Independent and Identically Distributed
IOT	Internet of Things
MFN	Master Fog Node
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
OCSVM	One Class Support Vector Machine
QoS	Quality of Service
RBF	Radial Basis Function
RR	Round Robin
SFN	Slave Fog Node
SGD	Stochastic Gradient Decent
SL	Split Learning
SLA	Service Level Agreement
SVM	Support Vector Machine
TOSCA	Topology and Orchestration Specification for Cloud Applications
WiFi	Wireless Fidelity
WRR	Weighted Round-Robin

List of Symbols

n	Number of sensors
k	Number of gateways
i	Number of SFN
p	Number of tasks
S_n	nth sensor node
G_k	kth Gateway
CF_k^G	CPU frequency of G_k
SFN_i	ith SFN node in the network
CF_i^{SFN}	CPU frequency of ith SFN node SFN_i
T_p	pth task from task set T
T_p^{in}	Input size of pth task
T_p^{out}	Output size of pth task
$\mathcal{Y}(p,i)$	Decision variable for assigning T_p to ith node
$E_{G_k}^{CPU}$	Computational energy expense of G_k
$E_{G(k,i,p)}^{Trans}$	Transmission energy from G_k to SFN_i
$E_{SFN_i}^{CPU}$	Computational energy expense of SFN_i
$E_{SFN(i,p)}^{Trans}$	Transmission energy from SFN_i to MFN
E_{MFN}^{CPU}	Computation expense of MFN
$E_{S_n}^{Trans}$	Transmission energy expense from S_n and G_k
E_p^{CPU}	Energy consumption of G for processing 1-byte task
$U_{p,k}^{up}$	Uploading rate from G_k to SFN_i
R_{pk}^{in}	Obtainable bandwidth utilization between G_k to SFN_i
PW_i^{up}	Transmission power for uploading T_p from G_k to SFN_i
C_p	Channel gain for offloading T_p
ξ_i^2	Additive white Gaussian noise of the SFN_i
TT_{pi}^{up}	Transmission time to upload T_p from G_k to SFN_i
R_{pi}^{up}	Obtainable bandwidth utilization between SFN_i to MNF
PW^{up}	Transmission power for uploading T_p from SFN_i to MFN
ξ^2	Additive white Gaussian noise of the MFN
TT_p^{up}	Transmission time to upload T_p from SFN_i to MFN
E_i^{CPU}	Energy consumption of SFN_i for processing 1-byte task
E_M^{CPU}	Energy consumption of MFN for processing 1-byte task
E_p^{local}	Total energy consumption in the local processing
E_p^{remote}	Total energy consumption in the remote processing
E_p^{total}	Total energy consumption for T_p
$E_p^{\text{total}}(t)$	Total energy consumption for T_p in time t

1 Introduction

Matthew Evans, the IoT program head at techUK [1], says, "Simply, the IoT is made up of devices– from simple sensors to smartphones and wearables – connected together". These IoT devices connected in conjunction form a network of devices and acts as the medium of capable and intelligent decision-makers for the environment [1]. Depending on the use cases like intelligent health care, smart grids, or smart agriculture, the utilization of IoT devices has become a necessity more than a requirement in recent times [2]. One of the significant sources of data wealth generation is the distributed IoT devices like mobile phones, autonomous vehicles, and wearable devices [3]. In fact, IoT consists of many interconnected devices that enable endless reporting of the variables like temperature, pressure, luminosity, etc., from the physical environment they are deployed in.

As a service-provisioner, cloud computing processes a plethora of applications over the internet and has several advantages. The sense of unlimited resources from the user-end and finely grained services like function as a service(FaaS) makes it very alluring [4]. However, when cloud computing is associated with the IoT framework for processing and analyzing data, offloading tasks from IoT devices to the cloud becomes a significant overhead in such architectures [5]. Furthermore, cloud computing suffers extensively when it comes to delay-sensitive applications like healthcare monitoring services [6]. Hence, the fog and edge architecture in this scenario plays a significant role in mitigating these issues.

The IoT devices can either be deployed as edge computing or fog computing architecture based on the proximity of the sensors and processing devices in the environment. In edge computing, the edge devices are more "physically" close to the sensors or have inbuilt sensors. Fog devices are more like IoT gateways processing data in a cluster or otherwise but indeed detached from the primary sensor devices [7]. In our use case, which involves collecting and analyzing data from the agricultural field, fog computing is a preferable option due to the unreliability of the low-cost edge devices in terms of data consistency and storage.

The network and connections between the things/objects in IoT play a vital role in creating an intelligent environment. Still, the real potential of IoT lies not on the "things" themselves, but the "data" accumulated in this context [8]. In such scenarios, machine learning(ML) is a distinguished approach to process the generated data and induce prospective knowledge to take an automated decision in real-time [9]. The advancement of ML has been dominant in the research fields like robotics, speech recognition, natural language processing, and many more. While in the fog computing domain, the progress of ML has been application-oriented enriching fog services like facial recognition, health care as well as fog computing paradigms, such as efficient resource management and mitigating delay for delay-sensitive applications [3].

[1] https://iotbuzzer.com/what-is-iot-top-10-definitions-overview/

Based on the learning problems, ML can be broadly classified into three domains: 1. supervised learning, 2. unsupervised learning, and 3. reinforcement learning. While supervised learning deals with a class of problems that involves mapping between inputs and target variable where the data is labeled, unsupervised learning is more oriented towards finding or extracting the relationship between unlabeled data. Apart from that, reinforcement learning deals with the problems that can be solved by an agent who learns to operate using feedback from the environment [10].

The applications of ML in distributed systems are different than those in the traditional cloud servers. The traditional cloud servers have greater processing power when involved in the computation of ML algorithms. But in standard ML approaches in the cloud, the data must be stored centrally, preferably in a data center. So when the application involved requires a real-time response from the ML model results, the cloud architecture fails to satisfy the Quality of Service(QoS) parameters [11]. This phenomenon can also have resultant fatal effects in real-time use cases like monitoring health and autonomous driving. Google brought forward one solution to these problems in 2017, known as FL [12]. This collaborative training method of FL can be briefly described in the following steps:

- The end device downloads the current model from the cloud.
- The downloaded model is improved with further training with the local data generated on the device end.
- The updates are sent to the cloud, where the model is averaged with the end devices' updates.

One of the prevalent applications of FL is Gboard on Android by Google. In Gboard, while typing texts, suggestions are visible, and depending on whether or not the user selects the recommendations, the model is further trained. Despite having a ton of benefits, FL still has challenges which the researcher needs to deal with. The major bottleneck for FL is communication energy [13]. The communication between the end devices and central node during the weight updating mechanism (like gradient data update) significantly impacts total energy expenses.

Inspired by the challenges mentioned above, gaps, and opportunities, this thesis proposes a fog architecture with the potential of nullifying the above-discussed issues without taking a hit on the performance of the architecture. The work done in this thesis reveals the advantages of using a gateway between the fog devices and sensors that deals with the distribution of the data and optimized resource provisioning for FL, which results in a significant energy reduction in the framework. An efficient data sampling algorithm is proposed, which executes at the gateway and reduces data redundancy, removes outliers, and aggregates data, the combination of which reduces the energy consumption in the framework in practice.

1.1 Motivation

The rapid development of real-time IoT applications, including green infrastructure, smart grids, smart city, intelligent transport networks, etc., enables green communication between tens of billions of end devices such as wearable devices and sensors. As a result, a tremendous amount of data is generated from massively distributed sources, which require computational intelligence techniques to fulfill high computing and communication demand that frequently exceeds the energy consumption. One of such use cases is smart irrigation in a smart agricultural ecosystem. Agriculture is a domain that impacts the present and the future of the existence of the civilization as food is an essential commodity in our life. Considering the increase of population in recent times, smart agriculture for maximum crop production has become imminent. The importance of efficient irrigation for maximum yield of crops and soil sustainability has become a focus in the smart agriculture ecosystem with the help of IoT and ML. Simple cloud-based solutions cannot fulfill such demands of data processing with data collected from IoT devices. Cloud being a centralized architecture, is restricted by offloading overhead, high latency, and network congestion for real-time applications. The need for a distributed architecture has been imminent and hence, motivates an efficient distributed framework design for real-time data analysis. These challenges can be addressed in depth using FL. But one of the disadvantages of FL is high energy expenses due to the intercommunication between the nodes or devices. Hence, from a practical and analytical approach, the following research question needs to be addressed to design an energy-efficient smart irrigation framework using FL:

1. How to design an intelligent hierarchical fog framework for analyzing real-time environmental parameters using advanced artificial intelligence techniques such as FL while reducing the energy consumption of the network?

2. How to distribute real-time sensory data to the nearby fog devices with an efficient resource provisioning mechanism while utilizing the computing resources (i.e., CPU utilization)?

3. How to apply FL on the resource-constrained fog devices for analyzing and predicting the real-time sensory data with higher accuracy?

Therefore, the goal of the thesis is to deduce an intelligent fog framework for real-time data analysis with modified FL technique and solving the challenges mentioned above in the network.

1.2 Contributions of the Thesis

To enhance the efficiency of the FL technique and reduce the overall energy consumption of the network, in this thesis, a novel energy-efficient FL technique is introduced for analyzing the real-time monitoring data efficiently with minimum energy usage and higher accuracy.

- Develop a hierarchical fog framework with an intelligent gateway between the sensor devices and fog devices to reduce data redundancy, detect and remove outliers using a one-class support vector machine and data aggregation algorithm for minimizing redundant and inaccurate data transmission through the network.
- Develop an energy-efficient resource provisioning algorithm based on CPU utilization of the fog devices using monitoring tool Prometheus and node exporters.
- Design a modified FL-enabled data analytics technique in the distributed fog framework for analyzing real-time monitoring parameters efficiently with higher accuracy.
- Extensive simulation results are conducted to demonstrate the efficiency of the proposed algorithm over advanced machine learning techniques such as standard FL and SL techniques using various performance metrics such as energy consumption and accuracy on a smart agriculture use case for irrigation control.

1.3 Organization of the Thesis

The thesis is organized as follows:

- Chapter 2 deals with the contributions and limitations of the existing state-of-the-art strategies in fog networks.
- Chapter 3 describes the system model followed by the problem formulation of the proposed hierarchical fog framework in the network.
- Chapter 4 reveals the proposed algorithm that has been used to enhance the energy efficiency and performance of the fog framework using data sampling, resource provisioning, and data analysis using the FL technique.
- Chapter 5 describes the experimental setup and environment used for simulation. The dataset generation and the data injection policies are also discussed here. Furthermore, the findings and performance analysis of the proposed architecture is shown by comparative analysis with the existing state of the art.
- Chapter 6 presents the conclusion and the future scopes of the proposed work.

2 State-of-the-art Strategies

In this chapter, the existing literature and the ongoing advancement in fog computing have been discussed. The methodologies aligned with the thesis are examined to understand the trends in this field of research and their disadvantages in practice. This section aims to review the current research scenarios and explore specific gaps or limitations.

2.1 Data Aggregation in Fog Networks

The researchers in the fog network have inclined towards the conclusion that energy saving in fog networks is essential rather than just a possibility [14]. Data aggregation is way by which the total energy consumption of can be minimized. But the problems in aggregation is the aggregation delay. Aggregation delay is the time required for aggregating data and to determine the hops of transferring data. In their research [14], the authors proposed a technique for delay-energy tradeoff which reduces the aggregation delay. But their proposed work is only reactive and application-specific.

The authors of [15] devised a lightweight privacy-preserving data aggregation technique that enables the filter of false data injection and supports fault tolerance. Although this method significantly reduces the communication energy, its differential privacy analyzer makes it computation heavy. The authors of [16] proposed a distributed form of data aggregation based on local minimization of data. This method reduces the network traffic and network congestion in fog networks due to compression sampling. In the research [17], the authors proposed a multilevel data aggregation with a dynamic data aggregation protocol. Here, only the active nodes are selected on each level for data aggregation. The disadvantage is the performance of the model takes a hit when data is not IID data. In the work [18], the authors proposed a clustered data aggregation policy using spatial correlation and lifting wavelet compression, which results in energy saving of the fog network. The simulation in this work suggests that the methodology is energy efficient but is constrained by the compression ratio. In the work [3], Principal Component Analysis is used for data aggregation. The goal of this method is low latency and data redundancy reduction, which results in energy-efficient in execution. But the disadvantage is that the architecture in this work is nonscalable. In the research [19], the authors have proposed the data aggregation methodology on the principle of the tradeoff between data aggregation and topology control. The participating nodes involved in data aggregation would wait a specific time interval to perform the operations. Although this method shows improved energy efficiency, the latency is significantly high when the whole network is considered. Another work [20], suggests performing snapshot data aggregation and standard data aggregation after a specific time interval. The performance of snapshot-based aggregation takes a hit when the architecture is hierarchical. Adaptive-based data aggregation has been proposed in another research [21], where both lossless and lossy data aggregation is performed but depends on the architecture requirement.

There are disadvantages like high aggregation delay or energy expensive or nonscalable in these states of the art for data aggregation in fog network.

In this thesis, the proposed data aggregation algorithm is based on demand from the resource provisioner. When the framework is resource deficit for data offloading, data aggregation is performed on the data by averaging it with the following data. Hence, the proposed method does not increase the data aggregation delay, and the degree of aggregation is also maintained to a certain threshold to avoid losing the originality of the data. Furthermore, as the data is aggregated, the size of data decreases, and the data inaccuracies also reduce. Due to these advances in the proposed method, the accuracy of the model increases, and so does the energy efficiency.

2.2 Resource Provisioning Strategy in Fog Networks

One of the most crucial problems in fog computing is proper resource provisioning as the goals are balancing the load efficiently and utilizing the fog network effectively. Different studies have been conducted in the past to optimize the use of resources in the fog domain. Here, some of those recent contributions are discussed. Furthermore, their merits and demerits have been identified and how the proposed work solves those demerits. Resource provisioning can be broadly divided into two types: Static resource provisioning and dynamic resource provisioning.

Static resource provisioning is where the load balancing unit does not have prior knowledge of the state of the resources. The static resource provisioning strategies are more useful when the load or tasks does not vary over time. Round Robin(RR) is one of the primate resource provisioning techniques. In RR, the tasks are provisioned in a circular order among the resources. In this method, the balance of task distribution is not guaranteed, and as a result, the time of completion of the tasks can increase significantly in worst-case scenarios [22]. Unlike RR, the Weighted Round-Robin (WRR) algorithm [23] prioritizes the tasks to be deployed in resources. WRR performs better than RR but does not consider the current load of the resources, and the resource utilization is not optimum. Another algorithm, Max-Min algorithm, operates to assigns larger tasks to the fog or resource with more capacity. The disadvantage of this algorithm is that the smaller tasks are remained unassigned in the worst-case [24]. Like the Max-Min algorithm, the Min-Min algorithm [25] assigns the most priority to the smaller tasks to be deployed on the fogs with high computational power. The result of this algorithm is again similar to the Max-Min as the resources are not used efficiently.

The dynamic resource provisioning strategies deploy the tasks on the fog devices in real-time and utilize several dynamic and heuristic algorithms for optimization in resource provisioning. In the research [26], the authors proposed a heuristic approach for dynamic resource provisioning using the Bayes algorithm and clustering. In another heuristic approach proposed by the authors of [27], the resource provisioning is done by assigning the tasks using a scheduling policy-based stochastic method. The implementation of this method results in a better task throughput, yet fails in efficient resource provisioning. In the work [28], the authors devised an algorithm based on Gibbs sampling. Using Gibbs sampling, the algorithm executes optimal resource provisioning but fails when the resources are heterogeneous. An algorithm by the authors of [29] has been developed based on honeybee optimization. Although the algorithm reduces the waiting time of the tasks, it fails to optimize the resource provisioning in practice. In [30], the authors proposed an efficient resource provisioning policy using Lagrange Multiplier in euclidean form. But this resource provisioning algorithm is not efficient if the resources are non-homogeneous. Another significant contribution is by the authors of [31], where they use a queuing model on multiple resource cores. But this algorithm fails in the optimization of resource provisioning and task deployment.

Most resource provisioning strategies either fail in optimized resource provisioning or have the disadvantage that tasks are not properly provisioned. In this thesis, the proposed method optimizes the resource provisioning by computing CPU usage in real-time and deploys the tasks to the fog with the least CPU usage. In cases where the tasks cannot be provisioned, the tasks are directed for data aggregation. Hence, the proposed algorithm does not increase the delay of task completion in the framework.

2.3 AI-enabled Techniques for Data Analysis in Fog

One of the recent trends in fog computing has been towards integrating AI for data analysis. ML techniques are often applied to analyze a large amount of data for prediction, classification, or detection of events in fog computing [32]. The domain of ML is large, and the algorithms range from simple linear regression [33] to complex deep learning(DL) models [34]. Researchers and organizations have been using the ML techniques like SVM [35], KMeans [36], Random Forest [37], Decision Tree [38] in enhancing their systems by analyzing the data generated in the fog network. According to the authors of [39], it is crucial to include AI in the fog computing domain so that the end devices can perform data analytics. To do so, the algorithm and the framework of AI needs to be energy and memory-efficient. Microsoft developed an ML models [40] that can be deployed in Arduino Uno board. In the work of [41], a pathological speech detection use case is solved by K-Means deployed in Raspberry Pi and Intel Edison. The authors of [42] proposed a system for analyzing sensory data using fog networks. Furthermore, in [43], the authors developed a distributed deep neural network that can autoscale the neural network based on geographic span.

One of the factors to consider in analyzing data in fog networks is that the privacy of the end devices is protected. Data collection from the user end to a centralized location is not sensible and incurs a high cost in data offloading. In such scenarios, some of the distributed frameworks where analyzing data does not need to centralized are: FL [12] and SL [44]. Introduced by Google, FL [12] is a collaborative learning process with data stakeholders like mobile and tab users. The advancement of these 5G [45] technologies with higher performance makes it easy to embed ML in them. A number of existing studies like [12], [46], [47], [48] have revealed the important problems related to the implementation of FL. The authors of [12] and [49] have stated the possibilities concerning the design of FL frameworks and discussed the limitations, problems, and solutions. The authors of [50] have devised two different types of updating the weight methods in FL to reduce the communication cost. The authors of [47] have suggested an update method in five different FL models and have given a detailed empirical analysis using four different datasets. The authors of [51] have proposed an echo state model for FL to predict the orientation and location of virtual reality users. In the work [52], the authors have devised an algorithm to reduced communication cost in FL. The authors of [53] have proposed the resource provisioning problems related to FL in low latency vehicular networks. In the work [54], the authors proposed a way to minimize the transmission delay in FL, and in [55], the authors used FL to estimate network traffic to enhance the user's data rate.

Most of the prior works done on FL have assumed that FL can be readily integrated into the fog network without considering the bandwidth and power. But there is a high chance that FL will face errors when on the training face with the vast amount of data in the unreliable network. Moreover, due to the resource-constrained devices as the participating clients in FL, the number of FL users will be significantly low. So it is essential to have a resource provisioning algorithm to expand the horizon of FL. Here in this proposed framework for FL using a gateway between the sensors and the client devices participating in FL, a robust resource provisioning algorithm has been implementing, which maximizes the CPU utilization and minimizes the energy of the overall fog network.

3 System Model and Problem Formulation

This chapter highlights the system model of the proposed hierarchical fog networks for agricultural data analysis followed by the problem formulation of the work.

3.1 System Model

The hierarchical fog framework for agricultural data analysis is shown in Figure 1. The framework consists of a set of sensors or IoT devices, denoted as $S = \{S_1, S_2, S_3, ..., S_n\}$ that monitor the environmental parameters and forward to the local gateway device G for further decision making. The local gateway devices, represented as $G = \{G_1, G_2, G_3, ..., G_k\}$ with CPU frequency $CF^G = \{CF_1^G, CF_2^G, CF_3^G, ..., CF_k^G\}$ are responsible for performing data sampling and aggregation at edge of the network to reduce the redundancy of the monitoring parameters. Further, each gateway device G distributed the aggregated data to the set of local fog devices for further analysis and prediction. The local fog devices are classified into two categories, namely Master Fog Node(MFN) and Slave Fog Node(SFN). The set of local SFNs, denoted as $SFN = \{SFN_1, SFN_2, SFN_3, ..., SFN_i\}$ with CPU frequency $CF^{SNF} = \{CF_1^{SFN}, CF_2^{SFN}, CF_3^{SFN}, ..., CF_i^{SFN}\}$ are responsible for analyzing the aggregated parameters locally and the MFN takes a global decision using the weights from the analyzed data, received from the SFN with a FL technique. Finally, the weights from the analyzed data are stored in the centralized cloud servers for future reference during data prediction and recommendation.

Besides that, it is considered the environment parameters, represented as a task set $T = \{T_1, T_2, T_3, ..., T_p\}$ are distributed to the local SFNs through a gateway device G using suitable data sampling and offloading mechanism. Each of these tasks contains a tuple including (T_p^{in}, T_p^{out}), where T_p^{in} is the input size and T_p^{out} is the output size of the task T_p. In this context, we assume a binary offloading strategy for the task set T between the G and SFN, where each non-partitioned task T_p is sent to one of the SFN in the network. Here, a decision variable $\mathcal{Y}(p, i)$ is introduced for taking an assignment decision between the task T_p to a suitable ith node in the network, represented as follows.

$$\mathcal{Y}(p,i) = \begin{cases} 1 & \textit{if } T_p \text{ task is assigned to } ith \text{ node} \\ 0 & \text{otherwise.} \end{cases}$$

3.2 Energy Consumption Model

In the proposed fog framework, the overall energy consumption during data analysis in the computing nodes and data transmission through a reliable network depends on five parameters: computational energy expense $E_{G_k}^{CPU}$ in the gateway device for data aggregation and sampling, transmission energy $E_{G(k,i,p)}^{Trans}$ from the gateway devices to the suitable SFN, local computation energy $E_{SFN_i}^{CPU}$ in the SFN_i for data analysis,

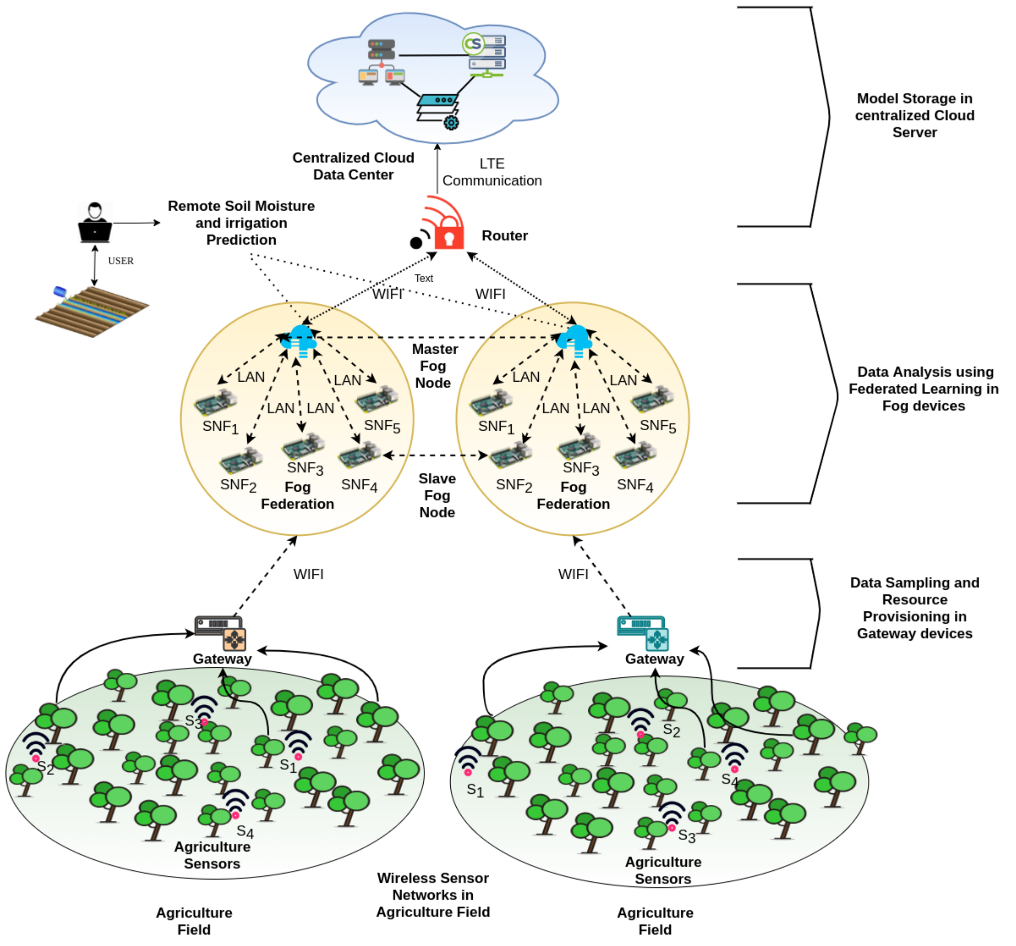

Model Storage in centralized Cloud Server
Centralized Cloud Data Center
LTE Communication
Router
Remote Soil Moisture and irrigation Prediction
USER
Text
WIFI
WIFI
Master Fog Node
LAN
SNF1
SNF2
SNF3
SNF4
SNF5
Fog Federation
Slave Fog Node
Data Analysis using Federated Learning in Fog devices
WIFI
WIFI
Data Sampling and Resource Provisioning in Gateway devices
Gateway
S1
S2
S3
S4
Agriculture Sensors
Wireless Sensor Networks in Agriculture Field
Agriculture Field

Figure 1. Hierarchical fog framework

transmission energy $E^{Trans}_{SFN(i,p)}$ during local training data transmission from each SFN to the MFN of the network for global training and computation expense E^{CPU}_{MFN} during the result aggregation in the MFN. It can be assumed that the sensor devices S_n and gateway devices G_k are in close proximity, hence, the transmission energy expense $E^{Trans}_{S_n}$ between S_n and G_k is significantly low, i.e., $E^{Trans}_{S_n} \simeq 0$.

In the proposed fog networks, the energy consumption depends on mainly two factors: (i) *Computational Energy:* depends on the time required to process and analyze the tasks in the local gateway device G_k and distributed SFN and MFN, respectively; and (ii) *Transmission Energy:* depends on the time required to the transmission of the monitoring task set from the gateway G_k to the distributed SFN and analyzed results from the MFN to the centralized cloud storage and recommendation systems. The following subsections describe the mathematical model of energy consumption in the fog networks during local processing (i.e., in the gateway device) and remote processing (i.e., distributed SFNs and MFN).

3.2.1 Local Processing Model

The set of local gateway device G, located nearby to the sensors or IoT devices is responsible to preprocess the monitoring data such as data sampling and aggregation before distributing it to the SFNs of the network for further processing with advanced machine learning techniques. In the proposed fog network, it is assumed that the local gateway devices are distributed nearby to the sensors or IoT devices, so the uploading and downloading time of the monitoring data is 0. Thus, the overall energy consumption during data preprocessing at the local gateway device depends on the CPU frequency of the gateway device. Considering the energy consumption of the gateway devices G for processing 1-byte task is E_p^{CPU}. Thus, the overall energy consumption for preprocessing a task T_p in the gateway G_k is defined as follows.

$$E_{G_k}^{CPU} = E_p^{CPU} \times \frac{\mathcal{Y}(p,i) \times T_p^{in}}{CF_k^G} \tag{1}$$

3.2.2 Remote Processing Model

In the proposed fog network, the preprocessed data are further offloaded to the distributed SFNs for further analysis with advanced machine learning techniques. Besides that, the MFN of the network is responsible for monitoring the distributed SFNs in the network and takes a global optimization decision based on the analyzed data, received from the SFNs. Thus, the overall energy consumption of the network during remote processing depends on the computation energy usage on the SFNs or MFN and the communication energy usage including the gateway device to SFN, SFN to MFN, MFN to cloud servers, and vice versa.

Uploading Energy The uploading energy of the pth task (T_p) depends on the uploading time of the task to the ith SFN node (SFN_i) and uploading rate $U_{p,k}^{up}$ of the network. The uploading rate $U_{p,k}^{up}$ of the network is defined as follows.

$$U_{p,k}^{\text{up}} = R_{pk}^{in} \times log_2 \left(1 + \frac{PW_i^{up} \times C_p}{\xi_i^2}\right) \tag{2}$$

Where R_{pk}^{in} is the obtainable bandwidth utilization between the gateway and the ith SFN node SFN_i, PW_i^{up} is the transmission power for uploading the task to the selected SFN_i, C_p is the channel gain for offloading pth task and ξ_i^2 represents the additive white Gaussian noise of the SFN_i, which is considered as a constant. The transmission time TT_{pi}^{up} to upload the pth task to the the ith SFN node is defined as follows.

$$TT_{pi}^{up} = \frac{\mathcal{Y}(p,k) \times T_p^{out}}{U_{pk}^{\text{up}}} \tag{3}$$

Thus, the total uploading energy consumption $E_{G(k,i,p)}^{Trans}$ from gateway G_k to the ith SFN node SFN_i for task T_p is calculated as follows.

$$\begin{aligned} E_{G(k,i,p)}^{Trans} &= TT_{pi}^{up} \times PW_i^{up} \qquad : \forall p \in T, i \in (SFN_i) \\ &= \frac{\mathcal{Y}(p,k) \times T_p^{out} \times PW_i^{up}}{R_{pk}^{in} \times log_2 \left(1 + \frac{PW_i^{up} \times C_p}{\xi_i^2}\right)} \end{aligned} \tag{4}$$

Similarly, the total uploading energy consumption $E_{G(k,i,p)}^{Trans}$ from SFN_i to the MFN for task T_p is calculated as follows.

$$\begin{aligned} E_{SFN(i,p)}^{Trans} &= TT_p^{up} \times PW^{up} \qquad : \forall p \in T, i \in (SFN_i) \\ &= \frac{\mathcal{Y}(p,i) \times T_p^{out} \times PW^{up}}{R_{pi}^{up} \times log_2 \left(1 + \frac{PW^{up} \times C_p}{\xi^2}\right)} \end{aligned} \tag{5}$$

Processing Energy The remote SFNs are represented as the computing nodes for analyzing the sensed parameters by the local sensors or IoT devices. The energy consumption of the SFNs depends on the input data size of the tasks and the CPU frequency of the selected SFN. Let us considered that the energy consumption for processing 1 byte task of the SFN_i is E_i^{CPU}. Thus, the required energy consumption during processing pth task on the ith SFN is represented as follows.

$$E_{SFN_i}^{CPU} = E_i^{CPU} \times \frac{\mathcal{Y}(p,i) \times T_p^{out}}{CF_i^{SFN}} \tag{6}$$

Consequently for the processing of the models aggregation leads to the computation expense E_{MFN}^{CPU} in the MFN can defined as:

$$E_{MFN}^{CPU} = E_M^{CPU} \times \frac{\mathcal{Y}(p, MNF) \times T_p^{out}}{CF^{MNF}} \tag{7}$$

Where the energy consumption of the MNF for 1-byte task ss E_M^{CPU} due to the task T_p in the MNF.

The total energy consumption in the local processing model for task T_p in the architecture in 1 iteration can be defined as:

$$E_p^{\text{local}} = E_{G_k}^{CPU} \tag{8}$$

The total energy consumption in the remote processing model for the task T_p in the architecture in 1 iteration can be defined as:

$$E_p^{\text{remote}} = E_{G(k,i,p)}^{Trans} + E_{SFN_i}^{CPU} + E_{SFN(i,p)}^{Trans} + E_{MFN}^{CPU} \tag{9}$$

So the total energy consumption for the task T_p in the architecture in 1 iteration can be defined as:

$$\begin{aligned} E_p^{\text{total}} &= E_p^{\text{local}} + E_p^{\text{remote}} \\ &= E_{G_k}^{CPU} + E_{G(k,i,p)}^{Trans} + E_{SFN_i}^{CPU} + E_{SFN(i,p)}^{Trans} + E_{MFN}^{CPU} \end{aligned} \tag{10}$$

The total energy consumption for the task T_p in the architecture at time t can be formulated as follows:

$$E_p^{\text{total}}(t) = E_{G_k}^{CPU}(t) + E_{G(k,i,p)}^{Trans}(t) + E_{SFN_i}^{CPU}(t) + E_{SFN(i,p)}^{Trans}(t) + E_{MFN}^{CPU}(t) \tag{11}$$

3.3 Problem Formulation

In this section, the problem of energy efficiency in FL is formulated. The processing energy and transmission energy are the prime disadvantages for applications of FL in resource constraint devices and delay sensitive use cases. Hence, the aim of this solution is directed towards the optimization of the time-average energy consumption in this architecture. The objective function with the constraints can be formulated as:

$$\text{minimize} \quad \lim_{t \to \infty} \sum_{t=1}^{\mathcal{T}} E_p^{\text{total}}(t) \tag{12}$$

$$\text{subject to} \quad E_p^{\text{total}}(t) \leq \mathcal{E}_q^{max}, \quad q \in (G \cup SFN \cup MFN) \tag{13}$$

$$CP^p(t) \leq CP_q^{max}, \quad q \in (G \cup SFN \cup MFN) \tag{14}$$

$$\sum_{p=1}^{|T|} \sum_{i=1}^{|\mathcal{F}|} \mathcal{Y}(p,i) \leq |\mathcal{F}| \tag{15}$$

$$\sum_{p=1}^{|T|} \mathcal{Y}(p,k) = 1 \tag{16}$$

$$\mathcal{Y}(p,k) \in \{0,1\} \tag{17}$$

$$TT_{pi}^{\text{up}} \geq 0 \tag{18}$$

The objective function aims to minimize the total energy consumption at t time-frame as shown in Equation 12. The constraints Equation 13 indicates that the energy consumption required for the task p should be less or equal to the qth computing device. Similarly, the constraint Equation 14 indicates that the CPU frequency required for the task p should be less or equal than that of the device q. The constraint Equation 15 indicates the overload prevention in the active computing devices represented as $\mathcal{F}$. The constraint Equation 16 shows that one computing device will work on only one task at a time. The Equation 17 indicates that binary offloading constraint and the constraint Equation 18 represents that the uploading time the task should be zero or more than zero.

4 Proposed Algorithm

This chapter states the contribution of the thesis in depth which results in the energy efficiency of FL. The problem formulated in the previous section is solved aligned to the research question identified in the motivation of this thesis. Proper irrigation or water management in agriculture is a significant factor in developing an intelligent agriculture ecosystem. Both the production of crops in the current year and the sustainability of the soil for future crop production depend on the optimum irrigation methodologies. Considering these factors, a fog based FL framework is trained with features like soil moisture and certain weather conditions. The end-users(farmers and agricultural farms) can make decisions based on this model. But one of the disadvantages of FL is the higher energy expenditure due to the intercommunication between the devices. Hence, a real-time solution has been proposed here to minimize the energy consumption in the architecture. Furthermore, the solution ensures that the model's accuracy is persistent, as it would be in a simplified architecture but with more energy consumption. The proposed solution can be framed in three steps that ensure the effective energy expenses in the fog architecture: (1) data sampling at the gateways (2) resource provisioning for the tasks at a particular instance, and (3) data analysis using FL.

4.1 Data Sampling

The sensors in the agricultural field collect and emit soil moisture and weather parameters like air temperature, air humidity, pressure, wind gust, wind speed, and wind direction. These raw data are transmitted to the gateway devices corresponding to the particular agricultural field (here, the assumption is that one unique gateway is used for each unique agricultural field). The data generated by the sensors from the field can have several typical raw data problems like data redundancy, data anomaly, and data inaccuracy. A three-step method of data sampling is used in this work to mitigate this issue.

4.1.1 Redundant Data Filtering

The moisture of the soil and weather are such features in an agricultural ecosystem that do not vary widely over a short period. For example, without sudden weather change, the moisture of the soil remains constant over a short period as the data are generated from the sensors continuously. The redundancy of data increases if all data are processed further for processing or learning. A method is devised here to discard such redundant data from the sensors. When the sensors emit data to the Message Queuing Telemetry Transport(MQTT) broker, the MQTT client subscribed to the particular broker only takes data as input when there is a change in the soil moisture in the agricultural field. This step of eliminating redundant data is done as in Algorithm 1 by considering the feature soil moisture as the parameter of detecting changes in the values transmitted from the sensors.

In Algorithm 1, it can be observed that the input is a set of tasks T and an empty queue Q. The first task is assigned to the first position in the Queue as $Q[0]$, and the $feature$ for comparison is denoted as $SoilMoisture$. Now if the value of the $feature$ of ith tasks $T[i]$ is not same as the previous task $T[i-1]$ then the Task $T[i]$ is appended to the Queue Q. A Queue of nonredundant values are extracted at the end of this process.

Algorithm 1: Redundant Data Filtering

```
1 INPUT: Incoming tasks/data: Set $T$, Queue :$Q \leftarrow Null$
2 OUTPUT: Generate a Queue with non redundant Task/Data
   1: Assign $feature : SoilMoisture$
   2: The First row of Data is Stored in the Queue
   3: $Q[0] == T[0]$
   4: for $i$:1 to $n$ do
   5:    if The current Task $T[i].feature \neq T[i-1].feature$ then
   6:       Append the Task $T[i]$ to Queue $Q$
   7:    end if
   8: end for
   9: Return a non redundant Queue $Q$ of Tasks/Data.
```

4.1.2 Outliers Detection using One Class Support Vector Machine

As the redundancy of the data is reduced, another significant aspect of bulk data is addressed here. When data is generated in bulk, it is a rule rather than an exception that the data will have anomalies or outliers. The outliers in data are caused by errors in the instruments in taking measurements and sometimes actual extremities in data. One cannot ignore the use of low-cost sensors in the agricultural field as a cause of error in data collection in this case. Furthermore, in an agricultural scenario, the error in the physical setup of the sensors can also produce anomalous data significantly.

We mitigate the above problem of outliers detection in data using a one class support vector machine (OCSVM) based mechanism. This detection is done by taking advantage of the "Novelty Detection" using OCSVM. "Novelty Detection" is the process of identification of the "unfamiliar" or "new" data that the model has not encountered before. If we let these "unfamiliar" data in the training phase in the model, it creates a negative impact during the prediction. For example, it is considered that the need for irrigation is the prediction parameter. In such case, if these anomalous data types are present, it causes inaccuracy in prediction, and the model suffers despite training at large.

The support vector machine's(SVM) general concept is to plot the data in N (where N is the number of features) dimensional plane. Then the SVM plots the hyperplanes, which separates the classes and chooses the best one based on the data. The mechanism is executed using the support vectors, which are nothing but the points on the plane for selecting the optimal hyperplane. But for SVM, the hyperplane separates the different classes based on the data. In OCSVM, the concept is similar to SVM but with the modification that the one class is the origin and the other is the whole training data as in the Figure 2.

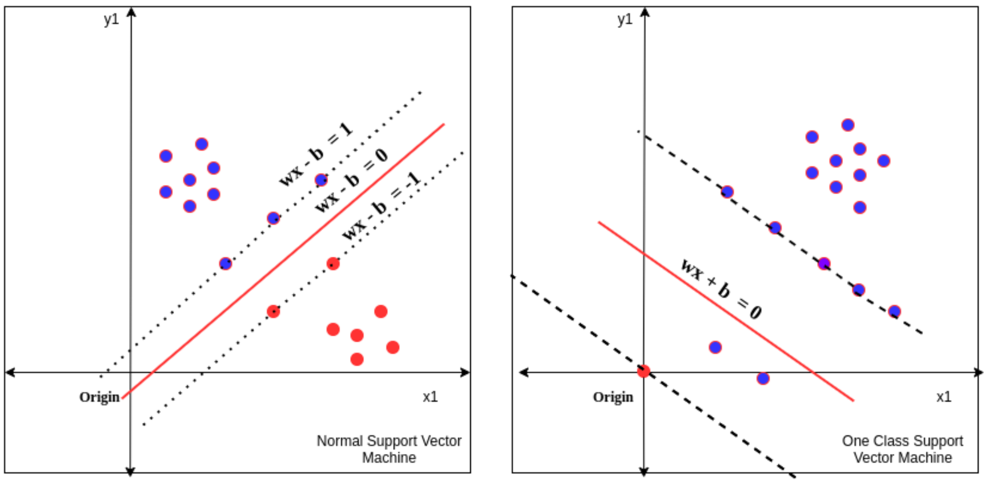

y1
wx - b = 1
wx - b = 0
wx - b = -1
Origin
x1
Normal Support Vector Machine
y1
wx + b = 0
Origin
x1
One Class Support Vector Machine

Figure 2. Difference between SVM(left) and OCSVM(right)

When the number of features is more than two, i.e., the SVM plots in $N > 2$ dimensional plane, the computation in the SVM takes a hit to form the optimum hyperplane. In this case, kernel ticks are used to provide non-linear classification. Kernel ticks transform the training data so that a non-linear decision surface is converted to a linear equation in a higher number of dimension spaces. In simple words, kernel ticks provide a computationally effective way to transform data into higher dimensions. So, while using OCSVM, the data $X_1, X_2, \cdots X_l \in X$ are mapped to higher dimensional feature space, and a hyperplane is formed which separates the origin and the training data by maximum margin. OCSVM can solve the solution of the hyperplane formation by solving the quadratic programming equations given as:

$$argmin_{\mathbf{w},\xi,\rho} \quad \frac{1}{2} \parallel \mathbf{w} \parallel^2 + \frac{1}{\upsilon n} \sum_{\mathrm{i}=1}^{n} (\xi_i - \rho) \tag{19}$$

$$\text{subject to} \qquad \langle \mathbf{w}, \phi(\mathbf{x_i}) \rangle \geq \rho - \xi_i \tag{20}$$
$$\xi_i \geq 0 \tag{21}$$

Where $\mathbf{w}$ is the normal vector, n denotes the number of training instances, υ is the trade-off parameter, ρ is the offset of desired hyperspace, and ξ is the slack variable. In Equation 20, $\phi(.)$ is the kernel function which is usually a nonlinear feature map to compute the inner product of the training data. The use of ρ is to set the upper bound of the fraction of training samples outside the decision boundary and a lower bound on the support vectors.

The kernel function $K = (\phi(x_i), \phi(y_i))$ is used to calculate the inner product and hence, reduce the explicit mapping of $\phi(.)$. There are a wide choice of kernel functions like Polynomial, Neural Network and Radial Basis Function(RBF), for the OCSVM, the Gaussian RBF $exp[- \parallel x_i - y_i] \parallel /2\sigma^2]$ is chosen based on the previous studies [56] where σ is the width parameter. The υ is set to a small value to minimize the misclassification rate, and as σ increases, number of support vectors decreases, and the decision boundary loosens. The choice of value σ is one of the main concerns in OCSVM and can be determined by an iterative algorithm using Grid Search. Hence, the quality of OCSVM is dependent on the choice of the parameters υ and σ. To do so, Grid Search is done on the OCSVM algorithm to determine the υ and σ as shown in Algorithm 2.

Algorithm 2: Choosing Optimal Value of σ and υ

```
1 INPUT: Set $\sigma = [0.01, \infty]$, Set $\upsilon = (0, 1]$ , Training Set $T$
2 OUTPUT: Optimal Value of $\sigma_{opt}$ and $\upsilon_{opt}$
    1: Perform Grid Search with OCSVM on $T$ with $\sigma$ and $\upsilon$
    2: SET cross validation = 10
    3: Calculate Precision and Recall
    4: Return the $\sigma_{opt}$ and $\upsilon_{opt}$ with best Precision and Recall score
```

In the Algorithm 2, a Grid Search is performed with defined parameters of σ and υ over a range of value $\sigma = [0.01, \infty]$ and $\upsilon = (0, 1]$. The cross-validation parameter is set to 10, and the scoring is done based on Recall and Precision. Based on the Precision and recall, the Grid Search gives the optimal value of σ_{opt} and υ_{opt} based on the Training set. Furthermore, outliers are detected and removed from the data using the optimal values σ_{opt} and υ_{opt} as shown in Algorithm 3.

In the Algorithm 3, the OCSVM model is trained with the parameters σ_{opt}, υ_{opt} and Kernel = rbf on T_{train}. The predicted values -1/1 is stored in the $ypred$ and the negative class index is stored as $outlierindex$. Then based on the values of the $outlierindex$, the training data $T_{train}[i]$ with the same index are dropped and stored in the queue Q.

Algorithm 3: Outlier detection and removal from the training data

```
1 INPUT: Incoming tasks: Set $T_{train}$, $\sigma_{opt}$ and $\upsilon_{opt}$, Kernel = $rbf$
2 OUTPUT: Generate a Queue $Q$ without outliers
  1: Fit and Train the OCSVM with $\sigma_{opt}$ and $\upsilon_{opt}$, Kernel = $rbf$ on $T_{train}$
  2: $ypred$ = $model.predict(df)$
  3: $outlierindex$ = $where(ypred == -1)$
  4: for $i$:1 in $outlierindex$ do
  5:     DROP $T_{train}[i]$
  6: end for
  7: Q = T
  8: Return a queue $Q$ without outliers .
```

4.1.3 Data Aggregation

The conventional methods of data communication are different from that of distributed devices. Some of the significant issues in distributed data communication include that the sensors are low powered, many to one data flow, data redundancy and packet loss. Hence, data aggregation is a way to solve this issue of data redundancy and packet loss when the data sources are plural. There are two significant classifications of data aggregation when dealing with the sensors or, in a more broad sense, IoT devices. One of the approaches of aggregating the data from many sensor devices in a singular packet is known as lossless aggregation. On the other hand, a statistical data aggregation (namely maximum, minimum and average) can be performed on data from a single source to base station; this is known as lossy aggregation. Lossy aggregation implies that the data cannot to transformed back to its original state.

After the redundant data and outliers are filtered form the data, the next phase requires the data to be communicated to the SNF. At this instance, it becomes a matter of decision for the gateway to choose the appropriate device to send the data. This is known as resource provisioning as discussed in the next section and Algorithm 5. But there is a problem to be addressed here. In dynamic resource provisioning where the number of computation components are static and there is need of resource provisioning, it becomes a bottleneck for the architecture. In terms of business level conversation, the Service Level Agreements(SLA) are violated when resource cannot be provisioned to tasks and there are packet losses. In our fog hierarchical architecture, it is to be noted that the computational devices are physical and are non scalable. Hence, a situational and local data aggregation methodology is developed to mitigate this loophole without compromising the integrity of the architecture. Here a parameter is introduced for measuring the number of times the data is aggregated known as Degree of Aggregation(DoA). DoA is used to stop repetitive aggregation on the data. In lossy aggregation, the DoA is required to be minimized as the originality of the data decreases when the DoA increases. The DoA can be, formulated as :

$$DoA = m(X) \quad (22)$$

where $m(X)$ is the number of times a data set X has undergone lossy aggregation.An algorithm is deduced to execute the lossy Data Aggregation algorithm as shown in Algorithm 4.

Algorithm 4: Data aggregation

```
1 INPUT: Queue $DA$ and m(DA)
2 OUTPUT: Data aggregation or resource provisioning based in DoA
  1: while $len(DA) \geq 1$ do
  2:    if $m(DA[0]) \leq 3$ then
  3:       Perform data aggregation by averaging the corresponding rows of DA[0] and
           DA[1] with same label value.
  4:       m(DA[0]) += 1
  5:    else
  6:       POP DA[0] from the queue and try resource provisioning in Algorithm 5
  7:    end if
  8: end while
```

In the Algorithm 4, when the Queue DA has more than one data set, data aggregation is performed based on averaging the two contiguous datasets and same label. As the data are aggregated, a value of DoA, $m(DA[0])$ is assigned to the data. If the $m(DA[0])$ is greater than 3 , which means, the data are already aggregated 3 times, it is popped from the Queue DA and is sent for resource provisioning using Algorithm 5.

4.2 Resource Provisioning

Resource provisioning is the mechanism by which the tasks and resources are mapped to one another. Here, the resources are the $SFNs$, and the tasks are set of data to be used for FL at the $SFNs$. After the agricultural data are processed at the gateway, it needs to be sent for further processing at the SFN for learning purposes. The gateway devices are required to choose the suitable fog device to send the data. If the data are sent to all the SFN, it defeats the purpose of distributed processing. Another way of sending the data to the fog nodes is slicing the data equally into all Fog devices. This process has the disadvantage that it requires more transmission and computational energy, not an energy-efficient approach. In this work, a load-based resource provisioning algorithm is designed for effective resource provisioning to the incoming tasks at the gateway. With problems and opportunities discussed above, a method of resource provisioning is formulated considering the computational capabilities of the Gateways. The gateway acts

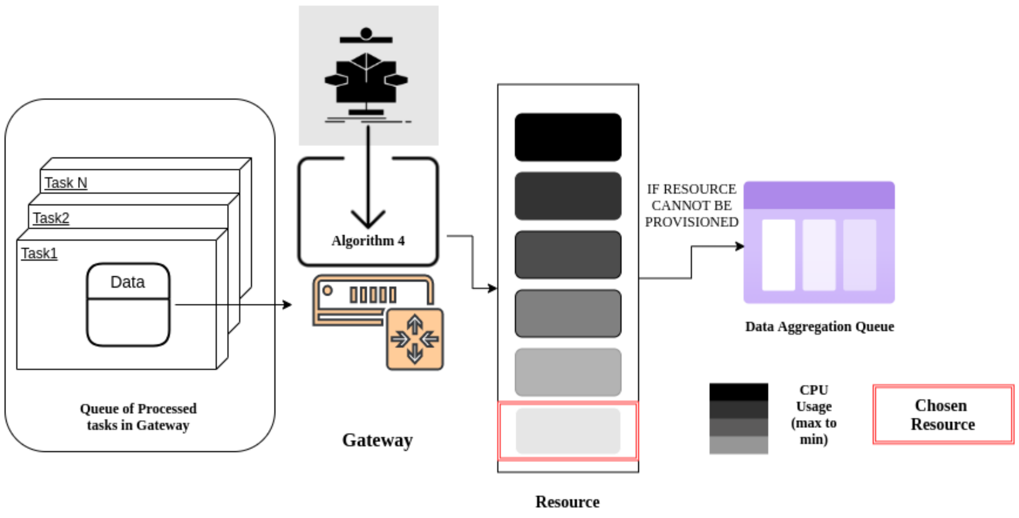


Figure 3. Resource provisioning by the gateway device

as a load-balancer to assign tasks to the $SFNs$ based on the model's design. The decision for the tasks to be provisioned to the fog devices can also be termed as data offloading. To perform this data offloading and, consequently, to optimize the fog device's computation, one of the significant factors is CPU Usage of the Fog devices. The CPU usage parameter in the context of a physical device allows measuring how much a processor is used at a specific time instance. The CPU usage of a device can be monitored using software like Prometheus. Therefore, the server-side of the monitoring tool is deployed at the gateway, and SFNs are the client-side for the software. Hence, the resource provisioning algorithm is devised based on Algorithm 5.

In Algorithm 5, the task $T_{train(j)}$is the incoming task, and it is being provisioned to the SFN assigned as $CPU(min)$ with minimum CPU usage. This is done by calculating the $CPU_{usage(i)}$ of each SFN and finding the SFN with minimum CPU usage $CPU_{(min)}$ with capacity $CPU_{provisioning(min)}$. But there is a condition that the $CPU_{provisioning(min)}$ is greater than a certain $CPU_{provisioning(thresh)}$ that is, the tasks cannot be provisioned to any resources (SFN). In such situations, the task $T_{train(j)}$ is stored in a queue to perform Data Aggregation using Algorithm 4.

4.3 Data Analysis using Federated Learning

As the tasks are now provisioned to the SFNs, the aim is to run a distributed learning mechanism to analyze the data for prediction. The distributed machine learning framework that is being used here is FL. FL offers a mechanism that considers the participating devices (here SFNs and MFN) and conducts a collaborative learning approach to result

Algorithm 5: Resource Provisioning Algorithm

```
1 INPUT: Incoming task: $T_{train(j)}$
2 OUTPUT: Find the best fit $SFN_i$ for task $T_{train(j)}$
 1: for $i = 0$ to $n$ do
 2:     Calculate the $CPU_{usage(i)}$ of the $SFN_i$ for last $t$ minutes
 3: end for
 4: $CPU_{provisioning(min)} = CPU_{usage(0)}$
 5: for $i = 1$ to $n$ do
 6:     if $CPU_{usage(i)} \leq CPU_{provisioning(min)}$ then
 7:         $CPU_{provisioning(min)} \leftarrow CPU_{usage(i)}$
 8:         $CPU_{(min)} \leftarrow SFN_i$
 9:     end if
10: end for
11: if $CPU_{provisioning(min)} \geq CPU_{provisioning(thresh)}$ then
12:     Assign $T_{train(j)}$ to $CPU_{(min)}$
13: else
14:     Store $T_{train(j)}$ in $DA$ queue
15: end if
```

into a global model with parameters update between the MFN and the SFN. Initially, during the training phase, the MFN initializes the global weight w_t and transfers it to the SFNs. As the SFNs receive the global model with initialized weights, it trains the model on the processed data received from the gateway devices. After the local training is done at the SFNs suppose at SFN_i, it returns the trained and updated model w_t^i to the MFN. The global model is then formed by aggregation of the local models w_t^i and results into a global model w_{t+1}. This one iteration can be termed as a $round$ in terms of FL. These rounds are repeated in our model as the tasks arrive at the gateways and stop when the model converges. Here, the FedAvg algorithm is used to aggregate the local weights to produce the global model, and the process is repeated.

Hence, with this process in view, the objective is considered as a finite sum problem which can be represented as:

$$min_{w \in \mathbb{R}} f(w) \quad where \quad f(w) = \frac{1}{n} \sum_{i=1}^{N} f_i(w) \tag{23}$$

In case of ML problems, it is considered that $f_i(w) = \ell(x_i, y_i, w)$, which is the loss function on prediction over the example (x_i, y_i) with model parameter w. In FL, the $SFNs$ are considered as clients and if the number of SFN clients are k and distribution of data is done is such a way that the set of indexes of data points are P_k, then $n_k = \parallel P_k \parallel$. Hence, the objective function in Equation 24 can be expressed as:

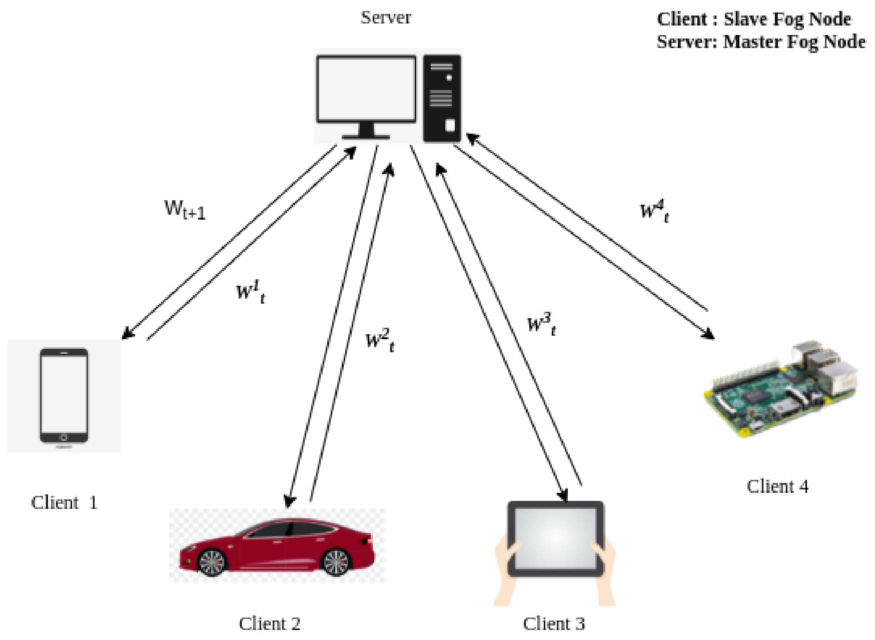


Figure 4. Federated learning architecture

$$f(w) = \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} F_k(w) \quad where \quad F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \tag{24}$$

Now if the data were uniformly distributed across the k SFN clients, then the expectation of $F_k(w)$ could be $\mathbb{E}_{P_k}[F_k(w)] = f_(w)$. This is true with the assumption that data is IID, but this case does not hold in this setup. The Expectation $\mathbb{E}_{P_k}$ is a bad approximation where the data is non-IID.

When the setup of the ML infrastructure is considered in data centers, the computation cost is more dominant than the communication cost. The use of GPUs are recently optimizing this computational cost in data centers. But in fog FL architecture, the communication energy expense acts as a bottleneck as the upload bandwidth is usually ranging around 1MB/s or less. Other factors to be considered here are that the $SFNs$ are to remain plugged in all the time, which can sometimes be not the case during certain instances due to failure and the usage of these $SFNs$ are to be assumed at few rounds daily. Hence, the goal of the FL algorithm design becomes minimization of rounds of communication to train the model. This is done here by using increasing independent parallelism, which means using more clients between communication rounds.

The usage of different versions of Stochastic Gradient Decent(SGD) has been one of the most prominent approaches in optimizing DL models. Thus, the application of SGD in FL can be one of the methods of optimization. But from previous studies [57], it is observed that the optimization of the Federated DL model can be very energy-intensive(especially communication energy). The number of rounds required for producing a good model is significantly large (for example, SGD takes 50000 rounds to build a good model even using Batch Normalization on the MNIST dataset).

In the study conducted by Chen et al. [58], the authors devised a large-batch normalization approach in SGD, which performs much better than asynchronous optimization approaches. When mimicked on the FL paradigm, the C-fractions of clients are selected for each round. When the value of this C is 1, then the FL model can perform full batch gradient descent. The implementation of this is termed as Federated Stochastic Gradient Decent(FedSGD), where C=1, learning rate = η and w_t is the average gradient on the local data of k, each client k computes as :

$$g_k = \nabla F_k(w_t) \tag{25}$$

and the update done by the central server averaging this gradients is :

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} g_k \quad as \quad \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} g_k = \nabla f(w_t) \tag{26}$$

As we considered k number of client in this case, for all k clients, the update can be formulated as:

$$w_{t+1}^k \leftarrow w_t - \eta g_k \tag{27}$$

and at the central server, the update will be:

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} w_{t+1}^k \tag{28}$$

Technically, the local k client devices update the gradient ones on the current model at each round. The weighted average of this k client's devices is being performed at the central server. But considering the overhead in the communication of using FedSGD, Federated Averaging(FedAvg) is implemented. The prime difference between FedSGD and FedAvg is that instead of calculating the local wight update just one time, $w_{t+1} \leftarrow w_t - \eta \nabla F_k(w^k)$ is applied multiple times in the local clients before it is averaged by central server. Apart from the multiple number of computations in each local clients, FedAvg intends to control the computation using the three parameters such as (i) Fraction of Clients per round $\rightarrow C$, (ii) Number of times the local data is trained at each client $\rightarrow E$, and (iii) The local mini-batch size on client $\rightarrow B$. Algorithm 6 and Algorithm 7 implies the execution of the FL (FedAvg Algorithm) in the server side and the client side respectively.

Algorithm 6: FedAvg algorithm ($ServerExecute$)

```
1 INPUT: K clients indexed by: $k$
2 OUTPUT: Global weight $w_{t+1}$
  1: initialize $w_0$
  2: for each round $t = 1$ to $\infty$ do
  3:    $m \leftarrow max(C * K, 1)$
  4:    $S_t \leftarrow$ set of m clients
  5:    for each client $k \in S_t$ in parallel do
  6:       $W^k_{t+1} \leftarrow ClientUpdate(k, w_t)$
  7:    end for
  8:    $w_{t+1} \leftarrow \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} w^k_{t+1}$
  9: end for
```

In the server side algorithm which runs in the MFN, the initial weight w_0 is initialized. Then in each round the maximum number of clients or MFN are chosen for training and S_t is the set of m clients. With each client k in S_t, the clients train on the data and updates the weight to sent to the server MNF. MNF averages this weight and form a global model as in Algorithm 7.

Algorithm 7: FedAvg Algorithm ($ClientUpdate$)

```
1 INPUT: $K$ clients indexed by: $k$, Local minibatch Size: $B$, Local epochs $E$,
    Learning rate $\eta$
2 OUTPUT: local weight $w$
  1: $\mathbb{B} \leftarrow$ Split $P_k$ in batches of size $B$
  2: for each local epoch $i = 1$ to $E$ do
  3:    for each batch $b \in \mathbb{B}$ do
  4:       $w \leftarrow w - \eta \nabla \ell(w; b)$
  5:    end for
  6: end for
  7: return $w$ to server
```

In the Client side as in Algorithm 7, the client (SNF) trains the on the local data by splitting the data in batch size B for E number of epochs. where the weight is updated for each batch b as $w \leftarrow w - \eta \nabla \ell(w; b)$ using the loss function $\ell(w; b)$. A simple neural network model is used for implementation of the FedAvg in practice here. The global weight is generated from the MFN and the training is done on the $SFNs$.

5 Empirical Evaluation

This chapter deals with the simulation analysis of the architecture based on smart irrigation and the algorithms proposed in this thesis. The proposed work is validated and compared with other existing state-of-the-art(SL and standard FL) by simulation and empirical analysis.

5.1 Experimental Setup and Environment

The experimental setup consists of the tools and devices where the simulations are performed. In the use case of smart irrigation based on agricultural data, an architecture has been designed for an energy efficient FL. The environment replaces the theoretical architecture that is expected to be completed within the experimental design. This work consists of a hierarchical fog architecture, and it is deployed using several physical devices like raspberry pi, tinker board, laptop, and cloud server. The specification of this setup is an essential factor to be considered here as the computational power depends on the specification of the setup. The connection between the devices is also significant as the communication energy depends on them. The communication protocol used in the experiment is MQTT for data transfer, and the parameters/weights are communicated using web sockets. Hence, a detailed list of devices used and their specifications are given in the Table 1:

Table 1. **Configuration of equipment's used in the experiment**

Equipment Purpose	Equipment Name	RAM	HDD	Processor	Operating System
Gateway	Tinker Board S	2GB	16GB	Cortex-A17 Quad-Core	TinkerOS
Slave Fog Node 1	Raspberry Pi-3	1GB	32GB	64bit Quad Core Processor	Raspberry Pi OS
Slave Fog Node 2	Raspberry Pi-3	1GB	32GB	64bit Quad Core Processor	Raspberry Pi OS
Slave Fog Node 3	Raspberry Pi-3	1GB	32GB	64bit Quad Core Processor	Raspberry Pi OS
Slave Fog Node 4	Raspberry Pi-3	1GB	32GB	64bit Quad Core Processor	Raspberry Pi OS
Master Fog Node	HP EliteBook 840 G4	16GB	256GB	i5-8250U CPU @ 1.60GHz	Lubuntu 20.04
Sensor Data Generator	Openstack Instance	4GB	20GB	2VCPU	Ubuntu18.04
Multi-meter	PeakTech	NA	NA	NA	NA
Cloud Storage	Openstack Instance	4GB	20GB	2VCPU	Ubuntu18.04

Based on the location of the devices and their proximity, the devices are connected to each other by either in Local Area Network or WiFi. The detail setup and implementation of the physical devices for real-time implementation are described as follows.

5.1.1 Gateway Device

The gateway devices act as a function unit of the architecture where the data the preposessing steps: reducing the data redundancy and removing the outliers from the data are done. Furthermore, the gateway is responsible for the resource provisioning as well as data aggregation when there is a scarcity of resource(when all the $SFNs$ are busy). The data are transferred from the data generator to the gateway device using MQTT based communication and the broker is installed at the data generator. The mosquitto client on the gateway device subscribes to the data using a unique topic and prepares a CSV file. The preparation of the CSV file is done using the Algorithm 1. Hence, the resultant data only has values when there is change in value in the data.

The next step in the gateway device is the outlier detection and outlier removal from the non-redundant data. The OCSVM algorithm is applied on the data for this reason based on Algorithm 3. But the value of σ and υ are to be set before the execution of the Algorithm 3 by taking advantage of Grid Search on the data as mentioned in Algorithm 2. The application of Grid Search on the data based on recall and precision results into the optimized values of $\sigma = 0.001$ and $\upsilon = 1 \times 10^{-7}$ when the criterion are precision and recall. Now, this optimal values derived from Algorithm 2 is applied over two subset of the data as an experiment i.e. from Agriculture Field 1 and Agriculture Field 2. The parameters for visualizations are taken as learned forntier which is the decision boundary and the classification of the normal values and outliers as regular observation and abnormal observation respectively in the Figure 5 and Figure 6.

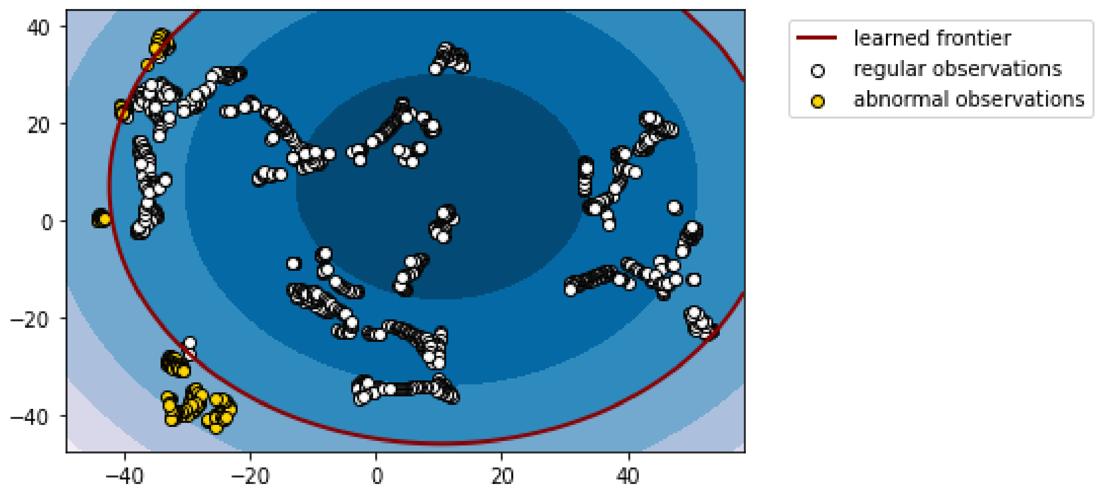


Figure 5. Outlier detection on the data from agriculture field 1

It is observable in Figure 5 and Figure 6, the outliers are reasonably detected, and the regular observation converges almost inside the learning frontier with minimum errors. The data is non-redundant and non-anomalous at this point; the data offloading is done from the gateway device to the $SNFs$ using the resource provisioning Algorithm 5. The resource provisioning algorithm uses Prometheus [2](a monitoring tool) to monitor the

[2]https://prometheus.io/

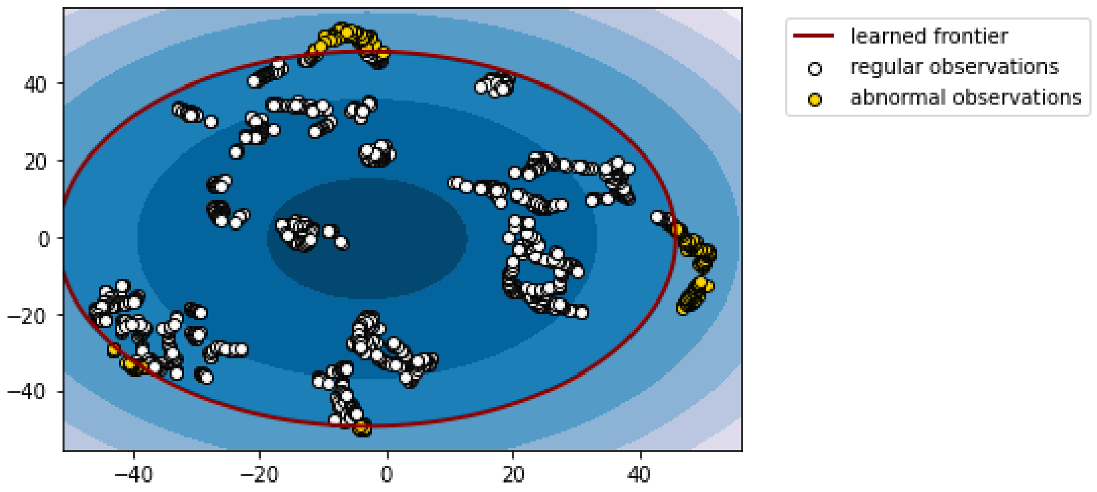


Figure 6. Outlier detection on the data from agriculture field 2

$SFNs$ and queries the Prometheus server installed on the gateway in order to find the SFN with minimum CPU Usage. The metrics from the $SNFs$ are exported using the installed node exporters[3] on the $SNFs$ when the prometheus query is performed. A threshold is set to 50 percent, and the SNF with minimum CPU usage is chosen as the resource for the data to be sent using scp with sshpass.

There is a possibility here that the gateway cannot allocate the resource (when the threshold criteria are not satisfied), as per the Algorithm 5. In such cases, the batch of data are stored for data aggregation in data aggregation queue DA. When this scenario of failed resource provisioning occurs again, aggregation is performed by averaging the values of current and the previous batch of data. But it is important to note here that the degree of aggregation DoA of the data stored in the DA queue cannot be more than 3. If the DoA is more than three, the data is sent for resource provisioning(allocating a SNF) without further aggregation. The flowchart of the processes within the Gateway can be represented as in the Figure 7 .

In the Figure 7, it can be observed that the incoming sensor data is processed in gateway with the steps of reducing the data redundancy, removing data outliers and the decision making process of resource provisioning. The resource provisioning also consists of a nested decision parameter of degree of aggregation. Based on this degree of aggregation the data is stored and aggregated by averaging as in Algorithm 4.

[3]https://github.com/prometheus/node_exporter

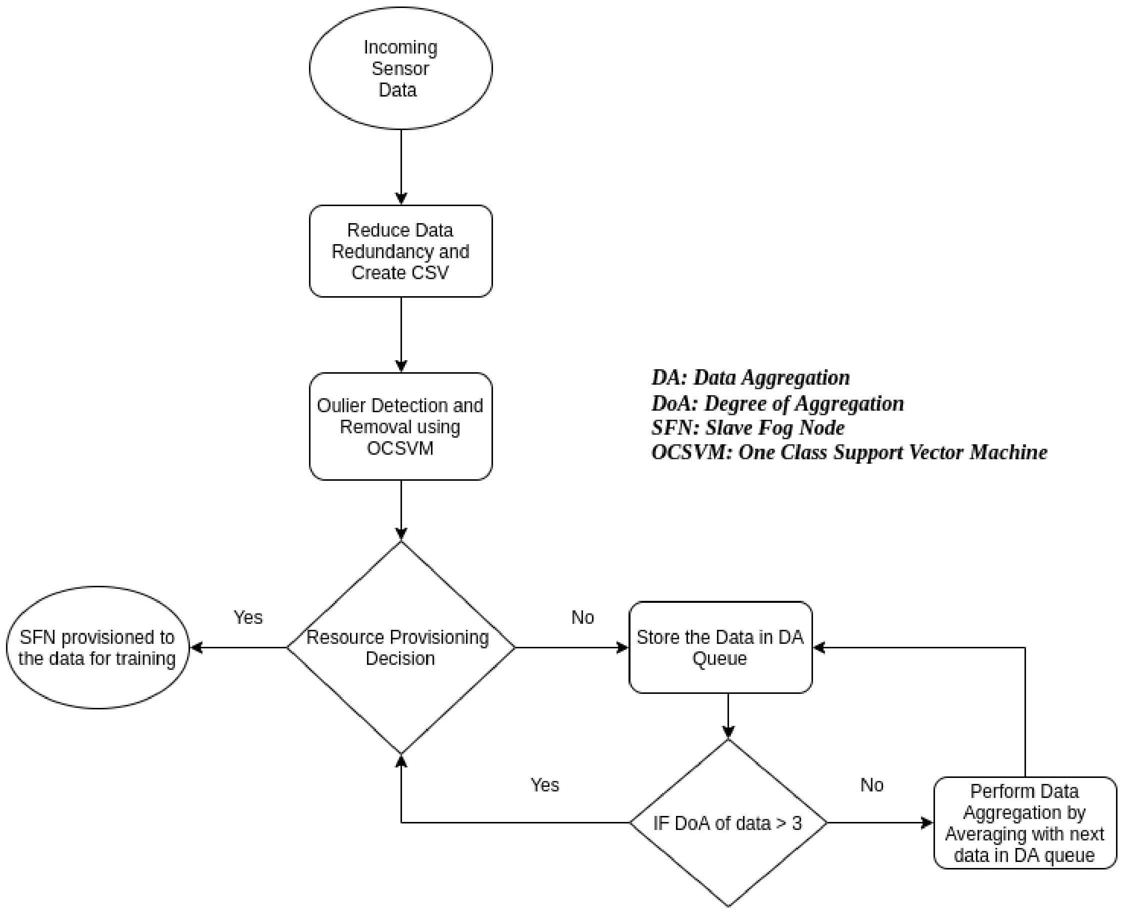

Incoming Sensor Data
Reduce Data Redundancy and Create CSV
Oulier Detection and Removal using OCSVM
DA: Data Aggregation
DoA: Degree of Aggregation
SFN: Slave Fog Node
OCSVM: One Class Support Vector Machine
SFN provisioned to the data for training
Yes
Resource Provisioning Decision
No
Store the Data in DA Queue
Yes
IF DoA of data > 3
No
Perform Data Aggregation by Averaging with next data in DA queue

Figure 7. Data flow in the gateway device

5.1.2 Master and Slave Fog Nodes

The computations in MFN and the $SFNs$ are executed in coordination with each other as FL is a collaborative learning process. The data is transferred from the gateway device to the $SNFs$, and the MNF initializes the global weight in the first iteration as mentioned in the Algorithm 6. The number of rounds in training is determined by the MNF and is ready to be sent to the participating $SNFs$ through a webSocket. The number of participating devices is also fixed by the MNF by maximizing the number of active MNF ready to process the training locally. The MNF is not accustomed to the data or the type of data in the SFN and sends the initial weights based on the neural network model as the $SFNs$ connects through the websocket.

The $SFNs$ execute the local training based on the local number of epochs parameter on the data transferred to them from gateway devices. The data at the SFN are divided into equal batches. A certain number of epochs of training are done on each batch of local training data, and the local weight is obtained as a result. These resultant local weights are communicated to the MNF for averaging and forming a global weight. This is what happens in each round of training as shown in Algorithm 7. As the subsequent batches of data are communicated from the gateway to the SNF, the next round of training begins initialized by the averaged weight from the previous round. The process of iterative continues depending on the number of rounds. The server-side code and the client-side codes for FL are executed on the MNF and $SNFs$, respectively. But to start the training, it is essential to create the MNF parameter initialization before the $SFNs$ are joined by websocket.

In the use case of predicting the need for irrigating, the MNF becomes the point of a query for the users to predict irrigation requirements based on the test data they provide. But it to be noted here if the data set is limited and the number of rounds is significantly high, the $SNFs$ can also offer the same functionality as the MNF in prediction. The reason for the occurrence of such a phenomenon is that the global weights and the local weights become the same. The MFN has some parameters to initialize before the training starts in the architecture.

An example of such parameters can be the criterion: CrossEntropyLoss, epochs: 10, learning rate: 0.001, batch size: 5 and optimizer: Adam. These parameters should remain the same for the $SFNs$ and MFN as well as the neural network used in the scenario. In practice, we manipulate the batch size of the data on the number of training rounds. In real-life scenarios, the training rounds in FL models are set to infinite as data are streamed from the sensors as long as the sensors are actively receiving data. Furthermore, another method used to deal with these endless data streams is storing the global weights to persistent storage after every the rounds. This is discussed in the next section. Considering the collaborative approach of $SFNs$ and MFN in FL execution, the activity sequence diagram can be represented as in the Figure 8.

5.1.3 Cloud Storage

The cloud storage in this architecture is considered persistent storage as shown in the Figure 9. The importance of persistent storage is that the data stored is retained even after the system is turned off. The MFN in the architecture is responsible for the global weight retention, but it acts like a volatile memory. In volatile memories, the data cannot be retrieved if the device is turned off or some failures occur. After all training rounds are completed, the MNF sends the global weights to the cloud server to save as the last checkpoint. Hence, in case of failure in the MFN, the MNF does not lose the trained weights done over time. It begins the training from the last saved checkpoint in the cloud server.

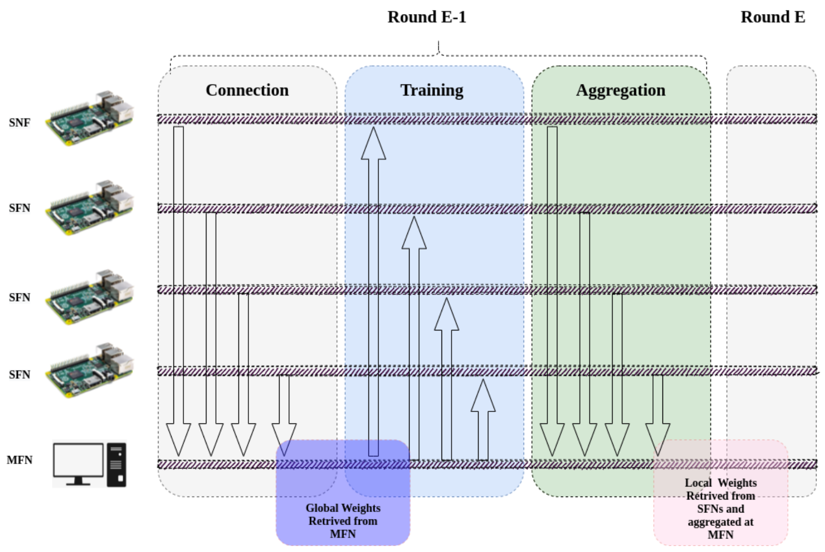

Round E-1
Round E
Connection
Training
Aggregation
SNF
SFN
SFN
SFN
MFN
Global Weights Retrived from MFN
Local Weights Retrived from SFNs and aggregated at MFN

Figure 8. Activity sequence diagram in one round of FL

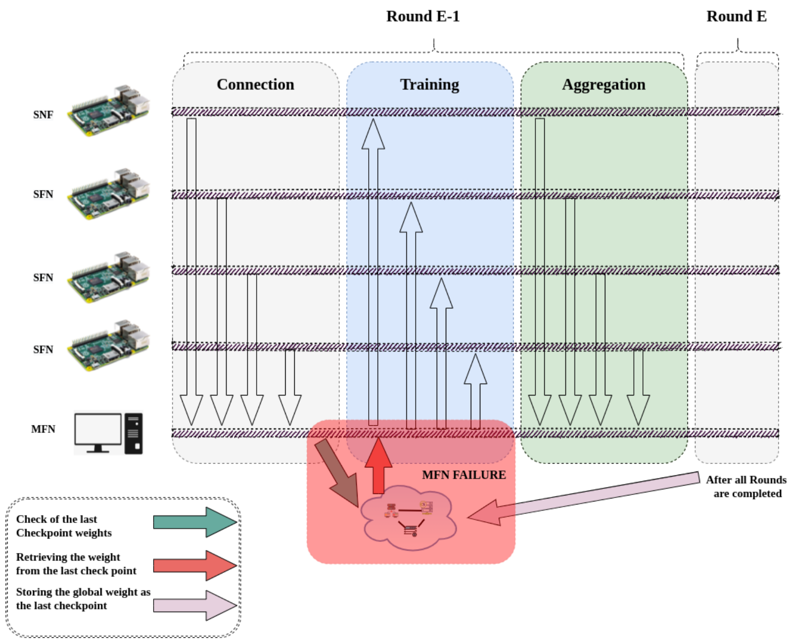
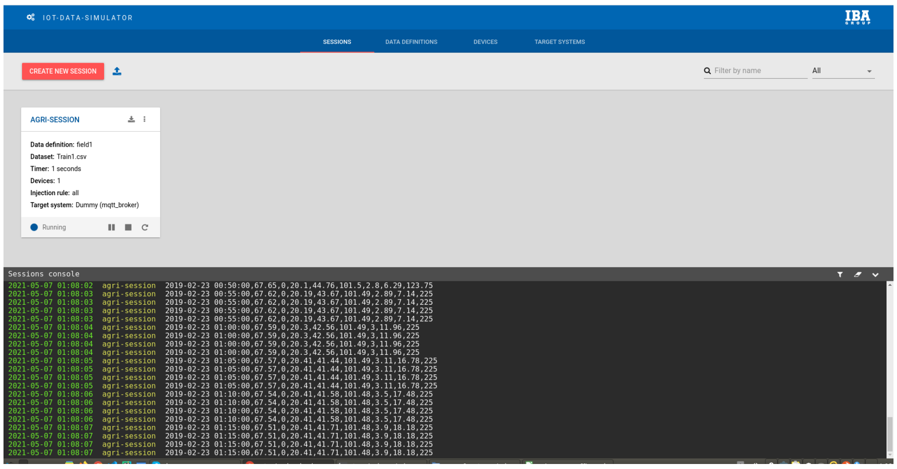

Round E-1
Round E
Connection
Training
Aggregation
SNF
SFN
SFN
SFN
MFN
MFN FAILURE
After all Rounds are completed
Check of the last Checkpoint weights
Retrieving the weight from the last check point
Storing the global weight as the last checkpoint

Figure 9. Activity sequence diagram with cloud in case of error in the MFN

There are two sequences of action by the MFN as observed from Figure 9 when the MFN fails or restarts the training. The MFN performs a check on the checkpoint weights and retrieves the weight from the last checkpoint stored in the cloud storage. Apart from failure handling, the cloud server has the advantage that the architecture can also be scaled if needed with a new set of MFN and $SFNs$. The scaling can be done by simply replicating the global weights stored in the cloud server to the new cluster of MFN and $SFNs$.

5.2 Dataset Generation and Injection Policy

The use case study in the thesis is smart agriculture to predict irrigation based on the data from irrigation fields. To simulate this data on the proposed setup, the data must be generated at a regular time interval. In real life scenario, data are generated from sensors in the field and are processed through the gateway devices to the cloud or other computational devices. In the model, a server is used to mimic these sensors to send the data to the gateway device.



Data definition: field1
Dataset: Train1.csv
Timer: 1 seconds
Devices: 1
Injection rule: all
Target system: Dummy (mqtt_broker)

```
Sessions console
2021-05-07 01:08:02  agri-session  2019-02-23 00:50:00,67.65,0,20.1,44.76,101.5,2.8,6.29,123.75
2021-05-07 01:08:03  agri-session  2019-02-23 00:55:00,67.62,0,20.19,43.67,101.49,2.89,7.14,225
2021-05-07 01:08:03  agri-session  2019-02-23 00:55:00,67.62,0,20.19,43.67,101.49,2.89,7.14,225
2021-05-07 01:08:03  agri-session  2019-02-23 00:55:00,67.62,0,20.19,43.67,101.49,2.89,7.14,225
2021-05-07 01:08:03  agri-session  2019-02-23 00:55:00,67.62,0,20.19,43.67,101.49,2.89,7.14,225
2021-05-07 01:08:04  agri-session  2019-02-23 01:00:00,67.59,0,20.3,42.56,101.49,3,11.96,225
2021-05-07 01:08:04  agri-session  2019-02-23 01:00:00,67.59,0,20.3,42.56,101.49,3,11.96,225
2021-05-07 01:08:04  agri-session  2019-02-23 01:00:00,67.59,0,20.3,42.56,101.49,3,11.96,225
2021-05-07 01:08:04  agri-session  2019-02-23 01:00:00,67.59,0,20.3,42.56,101.49,3,11.96,225
2021-05-07 01:08:05  agri-session  2019-02-23 01:05:00,67.57,0,20.41,41.44,101.49,3.11,16.78,225
2021-05-07 01:08:05  agri-session  2019-02-23 01:05:00,67.57,0,20.41,41.44,101.49,3.11,16.78,225
2021-05-07 01:08:05  agri-session  2019-02-23 01:05:00,67.57,0,20.41,41.44,101.49,3.11,16.78,225
2021-05-07 01:08:05  agri-session  2019-02-23 01:05:00,67.57,0,20.41,41.44,101.49,3.11,16.78,225
2021-05-07 01:08:06  agri-session  2019-02-23 01:10:00,67.54,0,20.41,41.58,101.48,3.5,17.48,225
2021-05-07 01:08:06  agri-session  2019-02-23 01:10:00,67.54,0,20.41,41.58,101.48,3.5,17.48,225
2021-05-07 01:08:06  agri-session  2019-02-23 01:10:00,67.54,0,20.41,41.58,101.48,3.5,17.48,225
2021-05-07 01:08:06  agri-session  2019-02-23 01:10:00,67.54,0,20.41,41.58,101.48,3.5,17.48,225
2021-05-07 01:08:07  agri-session  2019-02-23 01:15:00,67.51,0,20.41,41.71,101.48,3.9,18.18,225
2021-05-07 01:08:07  agri-session  2019-02-23 01:15:00,67.51,0,20.41,41.71,101.48,3.9,18.18,225
2021-05-07 01:08:07  agri-session  2019-02-23 01:15:00,67.51,0,20.41,41.71,101.48,3.9,18.18,225
2021-05-07 01:08:07  agri-session  2019-02-23 01:15:00,67.51,0,20.41,41.71,101.48,3.9,18.18,225
```

Figure 10. Data simulator in running session

There are many tools available for generating data like iosynth, CTGAN, SynSyn, etc. The data generator used here is IoT-data-simulator[4]. IoT-data-simulator provides flexibility in the generation of data, either custom based or from a historical dataset. The time stamps and other properties of the data can also be controlled using this tool. It is necessary to clone their repository from git and run the tool using docker commands. The simulator is a web application docker running on port 8090. It supports MQTT and other similar protocols. Here, the data simulator publishes data using an MQTT broker

[4]https://github.com/IBA-Group-IT/IoT-data-simulator

installed in it and this topic is subscribed by the mosquito client on the gateway device to receive and process the data.

Every session configuration of the data generator is done based on the field the data is being generated as seen in the Figure 10. The configuration file of each of these sessions needs to be configured. Each of these CSV files consists of the data from a particular field. The data generated from the data generator is sent to the topic with "data_fieldno" and are subscribed by the respective gateway. The parameters defined in IoT Data simulator can be expressed as in Table 2.

Table 2. **Parameters for data generation session**

Parameter Name	Parameter Value
Interval	5 mins
Data Definition	Created in accordance to the data
Injection Method	Specific to Sensor_no
Data processing rules	Default Value
Target System	IP and Port of the MQTT broker

5.3 Performance Analysis

With the experimental setup and data generation being done, it is essential to measure or quantify the effects of the changes in the performance due to the proposed architecture. This is done by comparing the proposed method with the existing state-of-the-art solutions in the same domain. The generalized problem dealt here is distributed machine learning in low end devices like raspberry pie and other mobile devices. The use case is predicting the irrigation requirement based on agricultural data. The importance of such architecture is data privacy preservation as well as to avoid the usage of cloud servers that leads to major computational overhead. The recent literature review suggests SL and standard FL as two of the state-of-the-art in this domain. Hence, we have compared the proposed modifed FL Approach with these two algorithms. Before the performance of the proposed work is compared with SL and standard FL, it is important to discuss what they are and how they differ from the proposed architecture.

Split Learning: SL is a collaborative learning process which uses DL to train on data in a client-server architecture similar to FL. But unlike FL, where the training on the data is executed only on the client side, SL uses both the server and the clients to train the model. In this collaborative learning process, the layers of the neural network model is divided in atleast two sub splits between the server and the clients. The data on the client side remains unknown to the server and this ensures privacy protection to the clients. The clients train the data on a sub network which consists of a few layers while most of the layers are on the server side. Using this concept, the processing at the clients can be reduced and the data privacy is also preserved. The SL architecture can be presented as the Figure 11. The difference between the SL and the proposed modified FL architecture are at two points. The modified FL uses the FedAvg algorithm where the learning part of the computation occurs at the client side (SFN) and the averaging of the local weights are done at the server side (MFN). Whereas in SL, the computation involved in the learning process takes place both at the server side MNF and $SFNs$ by slicing the neural network into atleast two parts. The other major difference is the gateway processing unit that has been introduced in the proposed work, this is not present in SL by default.

Federated Learning: To demonstrate performance analysis of the modified FL with the additional functional unit of gateway processing, it is essential to compare existing standard FL with the modified FL. In this scenario standard FL model is also based on FedAvg algorithm with the same architecture as shown in the Figure 4. The difference between the modified version of FL and the standard FL is mostly based on the data sampling part at the gateway. Before the data is offloaded to the modified FL clients $SFNs$, the data undergoes redundancy deletion, outlier removal and data aggregation on demand at the gateway device. The proposed algorithm is evaluated with the existing advanced

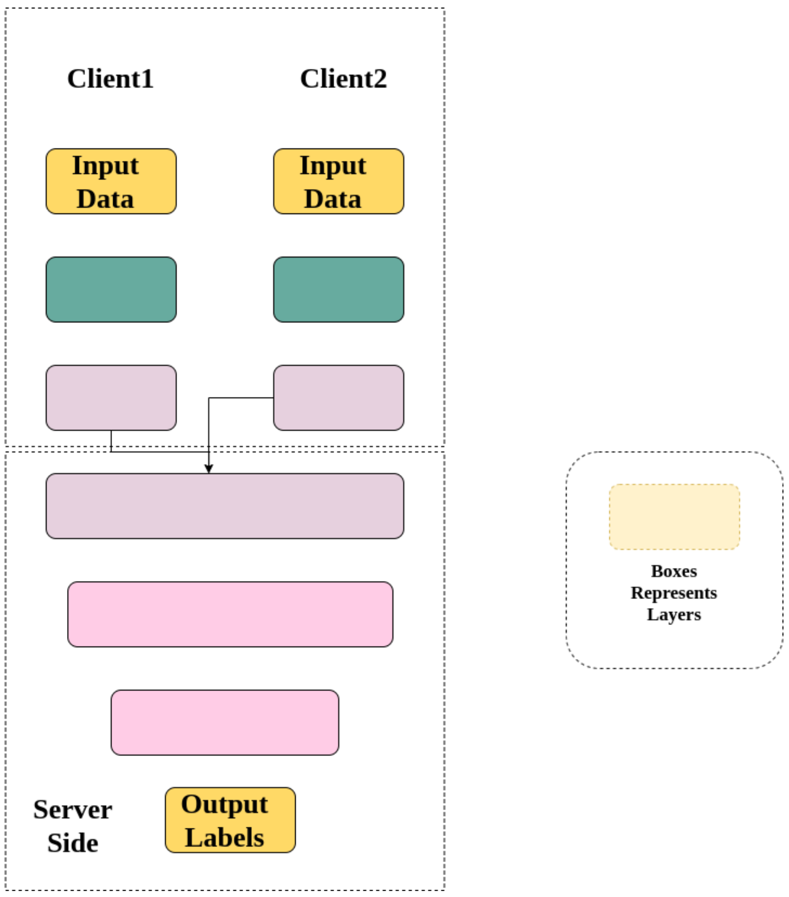


Figure 11. Architecture of split learning

machine learning techniques based on two performance metrics such as computation and communication energy consumption and accuracy.

5.3.1 Energy Consumption (Computational and Communication)

Collaborative learning techniques like FL and SL are helpful in real-world applications due to their coordinated nature of training. The distributed and parallel nature of such architecture results in significantly less time in execution and good accuracy scores. But these techniques are not energy efficient as the communication between the devices accounts for significantly more energy. In low-powered devices like mobiles and tablets, which are battery-powered, energy efficiency is one of the most significant parameters for deploying such technologies.

The computational current is calculated using multi meter when the devices are in the training phase. The computational energy cannot be calculated directly using a multimeter, but a multimeter can calculate each device's current(in Ampere) and voltage(in Volts). This analysis can be seen in the Figure 12.

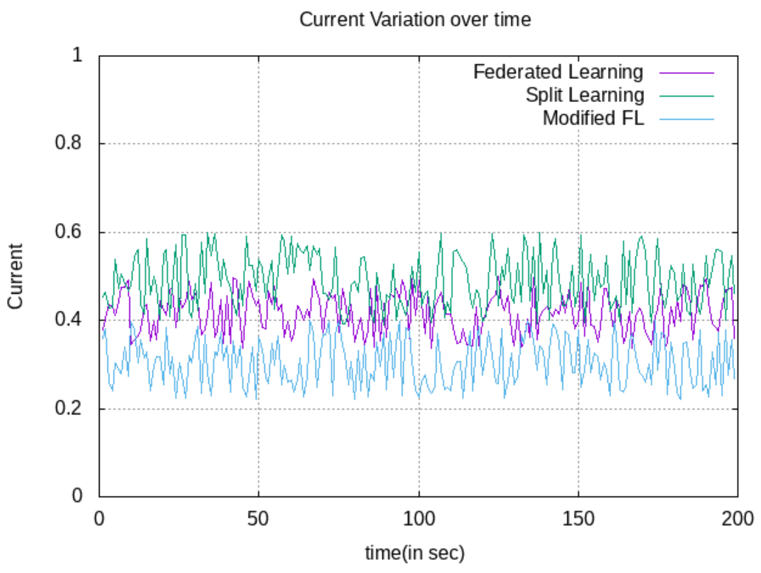
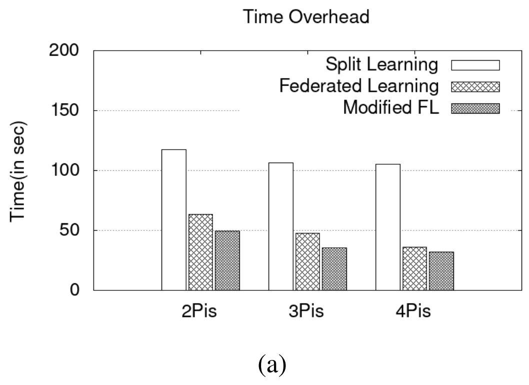


Figure 12. Current(A) Vs time(sec) graph during training

In the Figure 12, it can be seen that the modified FL consumes the least amount of current during training as compared to SL and standard FL. From the current consumption rate over 200 seconds, it is observed that the average current consumption during training for SL is 0.491 A, standard FL is 0.412 Ampere, and modifed FL is 0.305 A. The reduction in current consumption is due to the drop in the task size to T_p^{out} in Equation 5 in modified FL as compared to T_p^{in} in standard FL. The analysis of average current consumption holds true based on time overhead measurements. The time overhead in one of the training rounds is shown in Figure 13(a). The voltage while using the $SFNs$ is considered to a constant value of 5 Volts. Based on the time overhead and average current consumption of the compared algorithms, the total computation energy is calculated for every round, as in Figure 13(b).



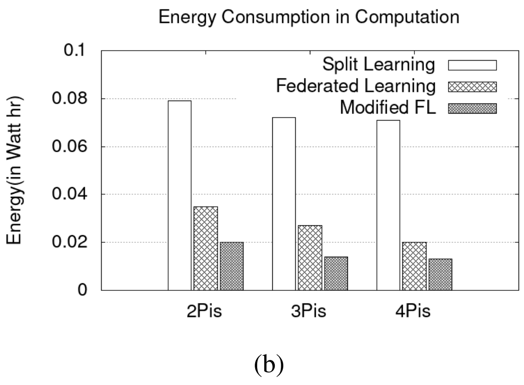
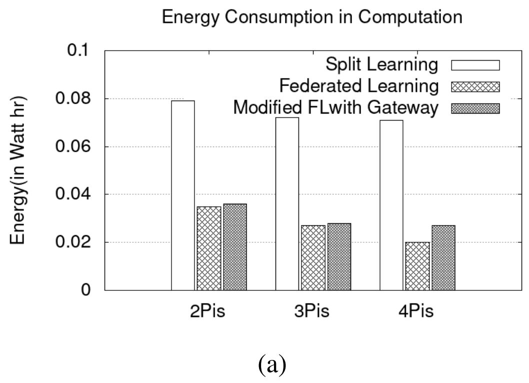


Figure 13. (a) Time overhead comparison in training in one round (b) Computational energy consumption in each round of training at SFNs comparison in training in one round.



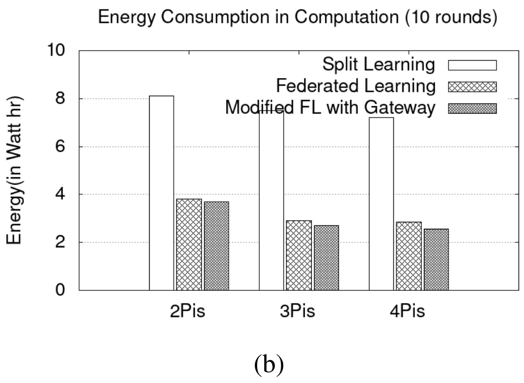


Figure 14. (a) Computational energy with gateway inclusion in 1 round (b) Computational energy with gateway inclusion in 10 rounds.

Apart from the computation in the local client devices $SFNs$, the computation at the gateway needs to be considered. Computations at the gateway are the resultant of the data prepossessing tasks in the modified FL and are included in Figure 14(a). In Figure 14(a), it can be observed for one round of training, when the computation in the gateway is considered, the modified FL is more energy expensive. The reason behind such a phenomenon is that computation in one round of training cannot perform aggregation, and data size does not decrease significantly. In such a scenario, the training is performed over ten rounds, and the result can be observed in Figure 14(b). After ten rounds of training, the computation energy graph 14(b) shows that the modified FL performs better in terms of energy consumption than that of SL and standard FL. The difference between the one round of training as in Figure 14(a) and ten rounds of training as in Figure 14(b) is due to the data aggregation at the gateways. Due to data aggregation, the $SFNs$ have to train with fewer data in the subsequent round in Figure 14(b) as compared to Figure 14(b). Hence, the difference in energy consumption is observed. From this observation, it can be concluded that as the number of rounds increases, the modified FL architecture trends to be more energy efficient.

Transmission of weight data between the devices in the architecture plays a significant role in training phase. The communication energy expense occurs in 2 phases in the proposed architecture. The data is transmitted from the gateway device to the $SFNs$ and bidirectional communication for weight transfer between MNF and $SNFs$. Latency is also one of the key factors in consideration for communication energy expense between the gateway and the $SNFs$. The Latency in transferring the data from the Gateway to the $SFNs$ for the 3 algorithms can be compared as in the Figure 15(a).

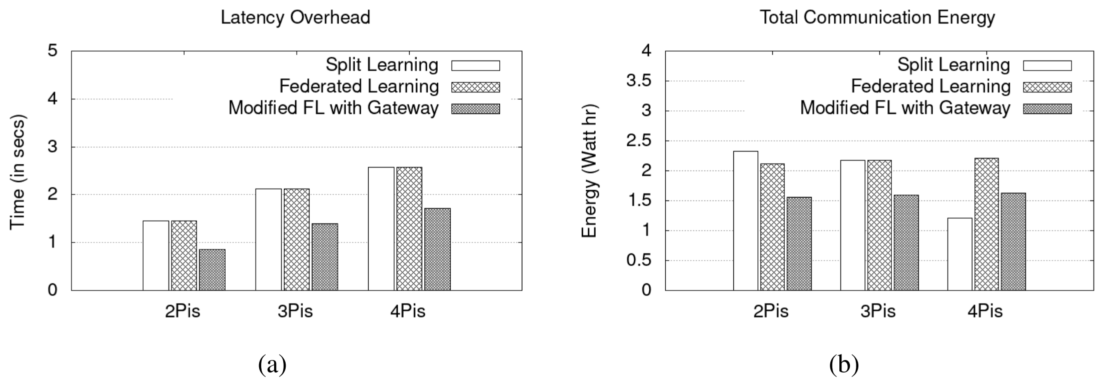


Figure 15. (a) Latency overhead for one round of training (b) Total communication energy in 10 rounds of training.

From Figure 15(a), it can be observed that the latency of the modifed FL is significantly less for each round of training due to the data prepossessing at the gateways unlike that of SL and standard FL. Furthermore, the total communication energy from the gateway to the $SFNs$ and the bidirectional communication between $SNFs$ and MNF can be represented as Figure 15(b). It can be observed in Figure 15(b), the modifed FL total communication energy is less than FL on average when the number of devices are increasing. Finally, the communication energy and computation energy together results in the total transmission energy for ten rounds of training as shown in Figure 16.

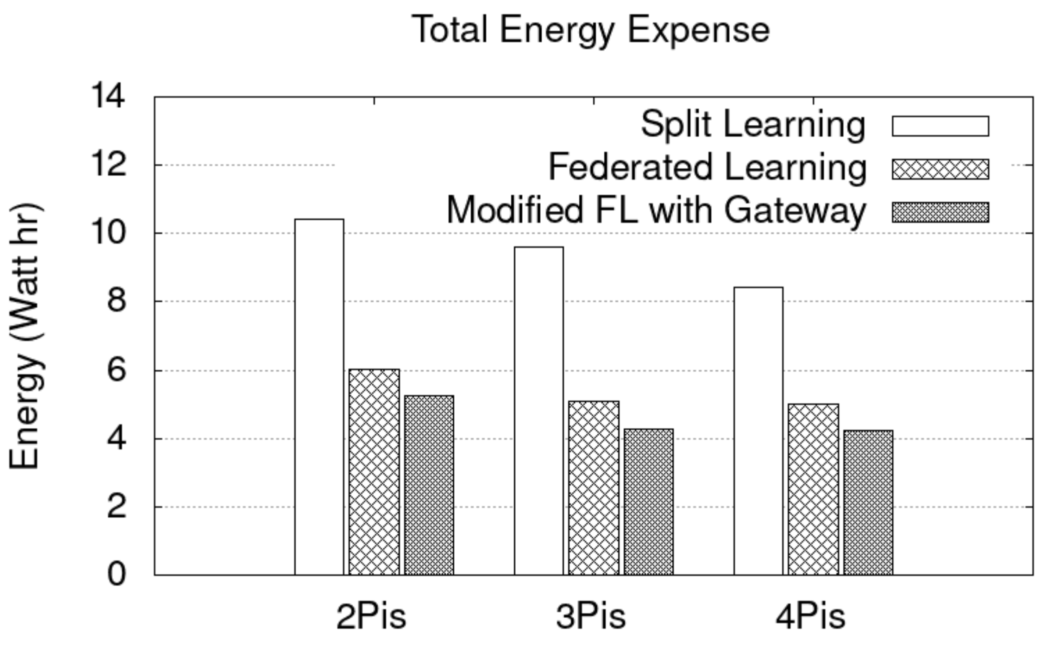


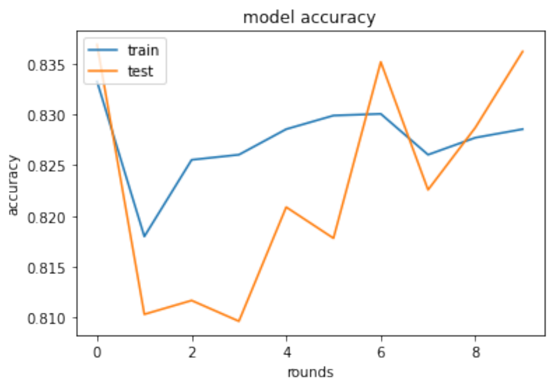
Figure 16. Total energy expense in 10 rounds of training

It can be observed from the total energy consumption graph in the Figure 16 that on an average, the modifed FL saves on total energy consumption by 51.5 percent compared to SL and 15.2 percent compared to standard FL.

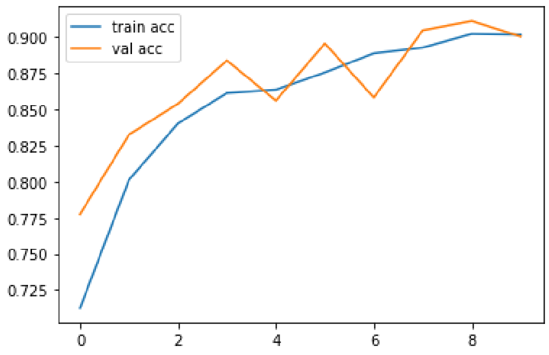
5.3.2 Accuracy

Accuracy is one of the parameters that can be used to measure the performance of any machine learning model. Accuracy can be defined as the percentage of correct prediction after the model is trained. Although it is more important to measure the accuracy of the model on test data, accuracy can also be an indicator for over-fitting when calculated on both test and train data. In the FL model evaluation phase, the accuracy is tested by each round of training in FL, and accuracies are collected for train data and test data. Here, the number of training rounds is fixed to ten, and the number of clients used for training is four. This evaluation is implemented for standard FL and modified FL model and is represented in the Figure 17(a) and Figure 17(b) respectively.

From the Figure 17(a), it can be observed that the standard FL results in an accuracy of 0.83 in test data after ten rounds of training. Whereas in the Figure 17(b), the



(a)



(b)

Figure 17. Model accuracy after 10 rounds of training (a) Standard FL (b) Modified FL.

accuracy score is 0.91 on test data. This proves that the data sampling steps (reducing data redundancy, removing outliers, and data aggregation) at the gateway enhance the performance of the modified FL model than that of the standard FL model.

6 Conclusion and Future Scopes

In this thesis, an intelligent fog framework has been designed for analyzing the real-time monitoring data at the edge of the networks with a modified FL technique. The contributions of this thesis are three folded. Firstly, a novel data sampling technique has been designed at the edge of the network for reducing redundancy, outliers removal, and data aggregation of the sensory data to increase the prediction accuracy. Secondly, a resource provisioning mechanism has been developed to distribute the aggregated data on the nearby fog devices while monitoring their resource availability through an online monitoring tool such as Prometheus. Finally, a modified FL technique is introduced on the distributed fog devices for analyzing the sensory data with higher accuracy. The key outcomes of the proposed strategy are summarized as follows.

- Due to the redundant data filtering, outliers removal, and data aggregation, the data inaccuracy is minimized, and the modified FL technique results in an accuracy of 91.1% as compared to the accuracy of 83% in standard FL.
- Employing the proposed methods, it is observed that the total energy consumption of the modified FL decreases by 51.5 % and 15.2% as the number of rounds increases compared to the existing SL and standard FL techniques, respectively.

However, the proposed algorithm has some limitations that can be considered as the future scope of the work, which are summarized as follows.

- Incorporating an intelligent intrusion detection mechanism in the fog networks for reliable data transmission and protecting the data from intruders is one of the critical research challenges. In the proposed fog-enabled smart agriculture framework, we have employed password-based device authentication to transmit data to the remote computing devices, however, it can be further enhanced by incorporating blockchain-based techniques in the fog networks.
- For automating the deployment of the services in the fog networks with a standard such as Topology and Orchestration Specification for Cloud Applications (TOSCA) and an orchestrator is another research objective. In the proposed scenario, the requested cloud services are deployed at the edge of the network based on their requirements, which can be automated by incorporating TOSCA standard and xOpera orchestrator.
- Introducing a self-adaptive resource provisioning mechanism in the fog networks with different heuristic or advanced AI-enabled technologies for optimizing multiple QoS parameters is one of the important research challenges. Besides that, adapting the proposed FL-enabled fog framework of real-time data analytics for other smart environments such as smart healthcare, intelligent transportation system, smart grid, etc. is the future scope of this work.

References

[1] Ovidiu Vermesan and Peter Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River publishers, 2013.

[2] Larissa R Suzuki. Smart cities iot: Enablers and technology road map. In *Smart City Networks*, pages 167–190. Springer, 2017.

[3] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[4] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.

[5] Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and Weijia Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6(3):4272–4282, 2018.

[6] L Minh Dang, Md Piran, Dongil Han, Kyungbok Min, Hyeonjoon Moon, et al. A survey on internet of things and cloud computing for healthcare. *Electronics*, 8(7):768, 2019.

[7] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan. Harvesting aware power management for sensor networks. In *Proceedings of the 43rd annual design automation conference*, pages 651–656, 2006.

[8] Subhadeep Sarkar, Subarna Chatterjee, and Sudip Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, 6(1):46–59, 2015.

[9] Zaib Ullah, Fadi Al-Turjman, Leonardo Mostarda, and Roberto Gagliardi. Applications of artificial intelligence and machine learning in smart cities. *Computer Communications*, 154:313–323, 2020.

[10] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[11] Mohammad Hossein Ghahramani, MengChu Zhou, and Chi Tin Hon. Toward cloud computing qos architecture: Analysis of cloud systems and cloud services. *IEEE/CAA Journal of Automatica Sinica*, 4(1):6–18, 2017.

[12] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi,

H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[13] Yuxuan Sun, Sheng Zhou, and Deniz Gündüz. Energy-aware analog aggregation for federated learning with redundant data. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.

[14] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. Modelling data-centric routing in wireless sensor networks. In *IEEE infocom*, volume 2, pages 39–44. Citeseer, 2002.

[15] Rongxing Lu, Kevin Heung, Arash Habibi Lashkari, and Ali A. Ghorbani. A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot. *IEEE Access*, 5:3302–3312, 2017.

[16] Tsung-Yi Tsai, Wei-Chi Lan, Chunlei Liu, and Min-Te Sun. Distributed compressive data aggregation in large-scale wireless sensor networks. *J. Adv. Comput. Netw.*, 1(4):295–300, 2013.

[17] Lutful Karim and Mohammed S Al-kahtani. Sensor data aggregation in a multi-layer big data framework. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–7. IEEE, 2016.

[18] Ledan Cheng, Songtao Guo, Ying Wang, and Yuanyuan Yang. Lifting wavelet compression based data aggregation in big data wireless sensor networks. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 561–568. IEEE, 2016.

[19] Vijay Erramilli, I Malta, and Azer Bestavros. On the interaction between data aggregation and topology control in wireless sensor networks. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 557–565. IEEE, 2004.

[20] Athanassios Boulis, Saurabh Ganeriwal, and Mani B Srivastava. Aggregation in sensor networks: an energy–accuracy trade-off. *Ad hoc networks*, 1(2-3):317–331, 2003.

[21] Kumar Padmanabh, Sunil Kumar Vuppala, et al. An adaptive data aggregation algorithm in wireless sensor network with bursty source. *Wireless Sensor Network*, 1(03):222, 2009.

[22] Rodrigo N Calheiros, Rajiv Ranjan, César AF De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[23] R Basker, V Rhymend Uthariaraj, and D Chitra Devi. An enhanced scheduling in weighted round robin for the cloud infrastructure services. *International Journal of Recent Advance in Engineering & Technology*, 2(3):81–86, 2014.

[24] OM Elzeki, MZ Reshad, and M Abu Elsoud. Improved max-min algorithm in cloud computing. *International Journal of Computer Applications*, 50(12), 2012.

[25] S Song, T Lv, and X Chen. A static load balancing algorithm for the future internet. *Journal of Electrical Engineering*, 12(6), 2014.

[26] Jia Zhao, Kun Yang, Xiaohui Wei, Yan Ding, Liang Hu, and Gaochao Xu. A heuristic clustering-based task deployment approach for load balancing using bayes theorem in cloud environment. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):305–316, 2015.

[27] Siva Theja Maguluri and Rayadurgam Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transactions On Networking*, 22(6):1938–1951, 2013.

[28] Chuan Pham, Nguyen H Tran, Cuong T Do, Eui-Nam Huh, and Choong Seon Hong. Joint consolidation and service-aware load balancing for datacenters. *IEEE Communications Letters*, 20(2):292–295, 2015.

[29] Dhinesh Babu LD and P Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied soft computing*, 13(5):2292–2303, 2013.

[30] YF Hu, Robert J Blake, and David R Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency: practice and experience*, 10(6):467–483, 1998.

[31] Junwei Cao, Keqin Li, and Ivan Stojmenovic. Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers. *IEEE Transactions on Computers*, 63(1):45–58, 2013.

[32] Melody Moh and Robinson Raju. Machine learning techniques for security of internet of things (iot) and fog computing systems. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pages 709–715. IEEE, 2018.

[33] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[34] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep

learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):1–36, 2018.

[35] Amira S Ashour, Amira El-Attar, Nilanjan Dey, Hatem Abd El-Kader, and Mostafa M Abd El-Naby. Long short term memory based patient-dependent model for fog detection in parkinson's disease. *Pattern Recognition Letters*, 131:23–29, 2020.

[36] Mareska Pratiwi Maharani, Philip Tobianto Daely, Jae Min Lee, and Dong-Seong Kim. Attack detection in fog layer for iiot based on machine learning approach. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1880–1882. IEEE, 2020.

[37] Irfan Ullah, Basit Raza, Ahmad Kamran Malik, Muhammad Imran, Saif Ul Islam, and Sung Won Kim. A churn prediction model using random forest: analysis of machine learning techniques for churn prediction and factor identification in telecom sector. *IEEE Access*, 7:60134–60149, 2019.

[38] SR Mugunthan. Decision tree based interference recognition for fog enabled iot architecture. *Journal of trends in Computer Science and Smart technology (TCSST)*, 2(01):15–25, 2020.

[39] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. A gap analysis of internet-of-things platforms. *Computer Communications*, 89:5–16, 2016.

[40] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International Conference on Machine Learning*, pages 1935–1944. PMLR, 2017.

[41] Debanjan Borthakur, Harishchandra Dubey, Nicholas Constant, Leslie Mahler, and Kunal Mankodiya. Smart fog: Fog computing framework for unsupervised clustering analytics in wearable internet of things. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 472–476, 2017.

[42] Zhuo Zou, Yi Jin, Paavo Nevalainen, Yuxiang Huan, Jukka Heikkonen, and Tomi Westerlund. Edge and fog computing enabled ai for iot-an overview. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 51–56. IEEE, 2019.

[43] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339. IEEE, 2017.

[44] Maarten G Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. Split learning for collaborative deep learning in healthcare. *arXiv preprint arXiv:1912.12115*, 2019.

[45] Yang Yang, Jing Xu, Guang Shi, and Cheng-Xiang Wang. *5G wireless systems*. Springer, 2018.

[46] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[47] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[48] Qunsong Zeng, Yuqing Du, Kaibin Huang, and Kin K Leung. Energy-efficient radio resource allocation for federated edge learning. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2020.

[49] Deyi Li and Yi Du. *Artificial intelligence with uncertainty*. CRC press, 2007.

[50] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[51] Mingzhe Chen, Omid Semiari, Walid Saad, Xuanlin Liu, and Changchuan Yin. Federated echo state learning for minimizing breaks in presence in wireless virtual reality networks. *IEEE Transactions on Wireless Communications*, 19(1):177–191, 2019.

[52] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[53] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. Distributed federated learning for ultra-reliable low-latency vehicular communications. *IEEE Transactions on Communications*, 68(2):1146–1159, 2019.

[54] Sukjong Ha, Jingjing Zhang, Osvaldo Simeone, and Joonhyuk Kang. Coded federated computing in wireless networks with straggling devices and imperfect csi. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2649–2653. IEEE, 2019.

[55] Oussama Habachi, Mohamed-Ali Adjif, and Jean-Pierre Cances. Fast uplink grant for noma: A federated learning based approach. In *International Symposium on Ubiquitous Networking*, pages 96–109. Springer, 2019.

[56] Yan Qiao, Kui Wu, and Peng Jin. Efficient anomaly detection for high-dimensional sensing data with one-class support vector machine. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[58] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, et al. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*, 2018.

Appendix

I. Licence