FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTERS THESIS

A Novel Anomaly Detection Mechanism for Open Radio Access Networks with Peer-to-Peer Federated Learning

Author	Samarathunga mapa Attanayakage Dinaj Rukshan Attanayaka
Supervisor	Dr. Pawani Porambage
Second examiner	Prof. Mika Ylianttila
Technical supervisor	Prof. Madhushanka Liyanage

November 2022

Attanayaka, Dinaj (2022) A Novel Anomaly Detection Mechanism for Open Radio Access Networks with Peer-to-Peer Federated Learning Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering, 78 pages.

ABSTRACT

Open radio access network (O-RAN) has been recognized as a revolutionary architecture to support the different classes of wireless services needed in fifth-generation (5G) and beyond 5G networks, which have various reliability, bandwidth, and latency requirements. It provides significant advantages based on the disaggregation and cloudification of the components, the standardized open interfaces, and the introduction of intelligence. However, these new features including the openness and the distributed nature of the O-RAN architecture have created new forms of threat surfaces than the conventional RAN architecture and require complex anomaly detection mechanisms. With the introduction of RAN intelligent controllers (RICs) in the O-RAN architecture, it is possible to utilize advanced artificial intelligence (AI) and machine learning (ML) algorithms based on closed control loops to perform automated security management in a data-driven manner, including detecting anomalies. In this thesis, the use of Federated Learning (FL) for anomaly detection in the O-RAN architecture is investigated, which can further preserve data privacy in a sensitive data processing system such as RAN. A Peer-to-Peer (P2P) FL-based anomaly detection mechanism is proposed for the O-RAN architecture and provides comprehensive analysis of four variants of P2P FL techniques. Three of the models are based on secure multiparty average computing, and the other is a homomorphic averaging-based model that provide protection against semi-honest local trainers. Moreover, the proposed models are simulated using the UNSW-NB15 dataset in a Python environment and the performance is tested using the same dataset. The simulation results indicated that all the proposed models have improved accuracy and F1-score values.

Keywords: 5G, beyond 5G, Network automation, Security, Privacy, Anomaly detection, O-RAN, RAN Intelligent controllers, Federated learning, Peer-to-Peer Federate Learning

CONTENTS

PREFACE

This thesis, titled as 'A Novel Anomaly Detection Mechanism for Open Radio Access Networks with Peer-to-Peer Federated Learning' is an original study performed by myself and submitted to the Masters Degree in Wireless Communication Engineering at the University of Oulu. The research described here was carried out at the University of Oulu's Centre for Wireless Communication Engineering, primarily under the supervision of Dr. Pawani Porambage.

I would like to express my gratitude to my main supervisor, Dr. Pawani Porambage, for the support and guidance given throughout this research. Moreover, I am grateful to Prof. Mika Ylianttila for providing funding as well as resources to proceed with this research and Prof. Madushanka Liyanage for the continuous guidance. Finally, I would like to thank all of the lecturers at the University of Oulu, as well as my parents and colleagues, for their encouragement and support throughout my whole academic period, including the research.

Oulu, November, 2022

Dinaj Attanayaka

LIST OF SYMBOLS AND APPREVIATIONS

O-RAN	Open Radio Access Network
5G	Fifth Generation
RAN	Radio Access Network
AI	Artificial Intelligence
ML	Machine Learning
FL	Federated Learning
P2P	Peer-to-Peer
RIC	RAN Intelligent Controller
1G	First Generation
2G	Second Generation
RNC	Radio Network Controller
3G	Third Generation
IP	Internet Protocol
4G	Forth Generation
eMBB	enhanced Mobile Broadband
mMTC	massive Machine Type Communication
URLLC	Ultra Reliability and Low Latency Communication
QoS	Quality of Service
SDN	Software Defined Networking
NFV	Network Function Virtualization
B5G	Beyond 5G
CAPEX	Capital Expenditure
OPEX	Operational Expenditure
C-RAN	Cloud RAN
vRAN	Virtualized RAN
IoT	Internet of Things
DoS	Denial of Service
SAC	Secure Average Computation
FHE	Fully Homomorphic Encryption
MLP	Multilayer Perceptron
IID	Independent and Identically Distributed
ZSM	Zero-touch Network and Service Management
Non-RT RIC	Non real-time RAN Intelligent Controller
Near-RT RIC	Near real-time RAN Intelligent Controller
BBU	Baseband units
COTS	Commercial Off the Shelf
RRU	Remote Radio Unit
3GPP	3rd Generation Partnership Project
gNB	Next Generation NodeB
CU	Central Unit
DU	Distributed Unit
RU	Radio Unit
RRC	Radio Resource Control
SDAP	Service Data Adaptation Protocol
PDCP	Packet Data Convergence Protocol
RLC	Radio Link Control
MAC	Medium Access Control
PHY	Physical

FCAPS	Fault, Configuration, Accounting, Performance and Security
SMO	Service Management and Orchestration
O-eNBs	O-RAN-compliant evolved Node Base
UE	User Equipment
LLS	Lower Layer Split
CP	Control Plane
UP	User Plane
VNF	Virtual Network Functions
CNF	Cloud-native Network Functions
VM	Virtual Machine
CN	Container
SW	Software
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
DP	Deferential Privacy
DRL	Deep reinforcement learning
DQN	deep Q-network
RAT	Radio Access Technology
SMC	Secure Multi-Party Computation
HE	Homomorphic Encryption
SL	Split Learning
RF	Radio Frequency
DDoS	Distributed Denial of Service
DBN	Deep Belief Networks
SAE	Stacked Autoencoders
KPI	Key Performance Indicator
KQI	Key Quality Indicator
DAD	Deep Anomaly Detection
E2E	End-to-end
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
XAI	Explainable AI
POC	Proof of concept

1 INTRODUCTION

Radio access network (RAN) is part of the mobile telecommunication system, which is between the core network and the user devices such as mobile phones, smart wearables, and computers. Starting with the first generation (1G) in the 1980s, the mobile telecommunication networks have evolved rapidly, as has the RAN [1]. Digital telecommunication was introduced in the second generation (2G), and it included a static functional architecture with geographically localized network functionalities [1]. The base station hosted all the RAN functionalities. However, in the next generation, which is 3G, these radio functionalities were split, with NodeB providing transmission and reception functionalities while the Radio Network Controller (RNC) provides management of the radio resources and user processing [1]. Due to the fast resource allocation, the overall latency was reduced in 3G, although the control process latency between the two parts is higher due to the split. Also, Internet Protocol (IP) based communication was utilized. In forth generation (4G) the data transfer rate and security management was vastly improved with extending the use of the IP for voice data with improved bandwidth and features. The need for a new generation of mobile telecommunications has arisen due to the rapidly increasing applications, requirements, as well as wireless devices.

Hence, the fifth generation (5G) is introduced, which can handle various types of traffic with different service requirements [2]. The primary use cases are enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) and Ultra Reliability and Low Latency Communication (URLLC). To cater to these cases with varying Quality of Service (QoS) requirements from a single network system, new technological concepts have been introduced, which include cloud computing, Software Defined Networking (SDN), and Network Function Virtualization (NFV) [3].

With the advent of 5G and beyond 5G (B5G) wireless systems, RAN has been required to handle an increasing number of new demands in terms of data rate, latency, dependability, mobility, architecture, and protocol complexity. [4]. As a result, mobile network operators should continuously upgrade their RAN infrastructure to adhere to new technologies and varying customer requirements, which will increase capital expenditure (CAPEX) and operational expenditure (OPEX) costs. Since the most significant percentage of the total network cost is accounted for RAN deployments and operations, the motivation to reduce CAPEX and OPEX has been significantly increased [5]. Over the years, various approaches have been introduced; two well-known approaches are Cloud RAN (C-RAN) and Virtualized RAN (vRAN). These approaches reduced the costs and simplified the maintenance, but the dependency on vendor lock-in was still a drawback. In order to overcome the limitations of C-RAN and vRAN, a standard open RAN solution has been proposed by the O-RAN alliance [6]. This new O-RAN architecture offers multi-vendor interoperability and is based on dis-aggregated, virtualized, and software-based components that are connected together using open, standardized interfaces. [5].

With the advancement of mobile communication in each generation, the security challenges have also increased and become more diverse. From the absence of privacy in 1G to spamming and radio link security issues in 2G, the security issues increased in 3G due to the introduction of IP-based communication to mobile networks, which imported the Internet's security vulnerabilities to the mobile networks [3]. Those issues were magnified in 4G due to the new features introduced. Also, the Internet of Things (IoT) was emerging in 4G, which opened several privacy and security concerns.

To avoid security concerns in previous generations, there are new security solutions in 5G with the additional architectures and design to support diverse 5G architecture. However, there are new security issues arisen in 5G due to new technologies introduced like NFV, SDN as well as due to the the diverse ecosystem [3].This will also continue with B5G networks, where connected intelligence is expected. As the entry point to the 5G and B5G networks, RAN is one

of the main targets for attacks. Furthermore, with the increased threat surface due to its inherently open and modular architecture, the risks are significantly higher in O-RAN [4]. Because of its openness, it is vulnerable to intrusions, and cyber attacks on the Open-RAN network can result in denial-of-service (DoS). Although the threat surface is increased in O-RAN, the surfaces are more clear due to the secure-by-design approach than in previous RAN implementations. Hence, defining secure methodologies for the critical assets and applying them will improve the overall security of the deployments and operations of O-RAN as compared with traditional RAN [4].

1.1 Background and Motivation

In 5G and B5G RAN, more proactive security measures have to be considered since conventional offline security measures that analyze a large amount of data would not be sufficient due to the high data rates as well as low latency demands. These vast amounts of data need to be processed in order to get insight into the network. When conventional offline methods are used, the probability of preventing an attack is quite low in 5G and future networks. Furthermore, manual RAN management, including security, is becoming increasingly difficult as technology advances since the number of components to be managed has increased exponentially and the system has become significantly complex. Therefore, intelligent, self-adaptive security mechanisms are required, and these should not affect the performance of the RAN [3]. As a result, near-real-time detection and prevention are preferred, and proactive measures are encouraged.

Artificial intelligence (AI) and machine learning (ML) methods have been used in many areas to implement proactive actions by processing large amounts of data. This approach can also be used for network management in mobile communication systems. AI and ML can fully automate network operations, lowering CAPEX and OPEX while improving performance. Although there are several advantages to ML, new threats have also arisen related to it, including privacy issues. Federated Learning (FL) which is a novel field of ML, has become a popular choice when there is sensitive data involved.

As mentioned previously, the O-RAN architecture would be the preferred RAN architecture in future networks. Until O-RAN, there was no inherent intelligence at the RAN [1]. Due to the introduced intelligence in O-RAN with the RAN intelligent controller (RIC), it is possible to incorporate AI and ML techniques to provide closed-loop operations for security analytics, attack detection, mitigation, or prevention, and security policy updates [4].

1.2 Research Problem

Anomaly detection is an important security measure in any network system. It is significant for O-RAN in 5G, which is a large-scale heterogeneous system with varying latency and privacy requirements. There can be several factors causing anomalies in a RAN, such as attacks, internal errors. Although there are several ML based research studies about anomaly detection on the RAN, a few focused on O-RAN [7].

When performing AI/ML-based operations, the privacy of the data being processed is a major concern. FL is a new form of machine learning that preserves privacy and improves communication efficiency by training models locally and communicating only the parameters for aggregation [8] which is preferred when performing AI based processes in large-scale complex environments with sensitive data such as RAN.

FL and its variation, P2P FL, are suitable for detecting anomalies in a complex O-RAN

environment [9]. Although there is little research on P2P FL, there are specific applications of P2P FL for anomaly detection in O-RAN architecture and automated security management. By using FL at the RAN to identify anomalies, attacks can be prevented before propagating to the core network. The use of FL also significantly protects data privacy and maximizes communication efficiency. In addition to that, it is possible to use FL-based anomaly detection for providing relevant control actions such as UE handover or resource management.

For this study, anomaly detection using P2P FL in O-RAN (which has a higher potential to be the next commercial RAN architecture) is considered a part of security automation in future wireless communication networks.

1.3 Selected scope

Anomalies in O-RAN can be caused by many factors, such as malicious UEs, attacks on O-RAN itself, and O-RAN misconfigurations [10]. In this study, only malicious network traffic, which includes seven types of attacks, is considered [11].

In this study, P2P FL with secure average computation (SAC) [12] is used to design a distributed anomaly detection technique that is compatible with the O-RAN architecture. P2P FL removes the single point of failure, and the parameters are not required to be transmitted to a centralized cloud [13]. In addition to that, we consider two variations of the P2P FL, one being a clustered P2P FL model where each cluster maintains a separate FL model, which is desired when the data is localized, as in RAN. And the other is a hierarchical version of the mentioned clustered P2P FL model, where almost the same model can be derived in each cluster while maintaining a relatively smaller number of communications in training than the normal P2P FL method. In addition to that, a P2P FL method based on multiparty communication via thresholded fully homomorphic encryption (FHE) is suggested to achieve a higher level of security both in communication as well as parameter average calculation [14].

A common Multilayer Perceptron (MLP) model was used for all the methods, which consist of four layers, including two hidden layers. Since the systems achieved sufficient accuracy, the structure of the model has remained unchanged. Three types of training data distributions, namely, random, independent and identically distributed (IID), and non-IID, were considered. Moreover, a total of 100 local trainers are considered.

A single network dataset called UNSW-NB15 is used for all the model training and testing. Furthermore, a ideal environment is considered where all the trainers are available and no communication link failures.

1.4 Methodology

Since this study is based on FL model deployment, first, data pre-processing for the model training was performed. UNSW-NB15 data set contain some feature which has non numerical values such as protocol, service. These were converted to numerical values. Also, some of the feature columns were removed during pre-processing using the TargetEncoder function in the categorical_encoders library, which had no effect on whether a particular flow was an anomaly or not. The ”Attack Type” column was also removed since it was enough to detect an anomaly. After that, all the data was normalized for efficient training. After the pre-processing, the dataset was partitioned into training and testing datasets. Furthermore, the ”Label” column was separated as a separate dataframe, and all the remaining columns were considered a single dataframe. The training dataset is partitioned into local datasets to be allocated to each local trainer, and this

partition step depends on the selected data distribution method for the simulation.

Then the model training was performed. In P2P FL methods local model training, followed by Secure average computation (SAC) or averaging based on HE with the parameter weights sharing was performed. Then local models are trained again using the averaged parameter weights, which completes a global round. In the centralized FL method used for comparison, the local model parameters were sent to a central aggregation server, and the aggregated values were sent back to the local training models. For the general ML method, each local trainer trained their models separately.

Trained model testing was the final phase of this research, and a pre-allocated testing dataset of 10,000 network flows was used for that, and accuracy, precision, recall, F1 score, and communication cost were recorded for each simulation as a Pandas dataframe.Multiprocessing was employed for model training and testing to take advantage of FL parallelism. Finally, in a separate Jupyter notebook, the recorded dataframes were loaded and the results were plotted for comparison.

1.5 Contribution

This study's major contribution is a P2P FL-based anomaly detector with secure average computation and its variations: Clustered P2P FL, Hierarchical P2P FL, and also a P2P FL detector with homomorphic encrypted averaging. O-RAN is relatively new, and there are only a few applications available. Utilizing ML or FL for security in O-RAN is currently in its early phases of research. Using FL in anomaly detection is studied for the systems such as the IoT [15], [16], and Zero-touch Network and Service Management (ZSM) [17] systems but not for the O-RAN. Therefore, this study provides an initial step for research into using FL for anomaly detection in O-RAN.

FL-based solutions can be easily deployed in O-RAN because of the hierarchical positions of Near real-time RIC (Near-RT RIC) and Non real-time RIC (Non-RT RIC) and their different closed loops. Hence the proposed methods, which are based on P2P FL, can be readily deployed in an O-RAN architecture. A reference architecture for deploying is also proposed for further study.

In all of the methods, performance was evaluated based on accuracy and the F1 score. Furthermore, the transmission cost of the training was considered. After 80 rounds of P2P FL methods, it was possible to find anomalies in the test dataset with an accuracy of about 90% and a F1-score value of 92%. Therefore, it can be concluded that the proposed models are effective against network attacks in the O-RAN architecture.

1.6 Organization of the Thesis

There are a total of six chapters included in this thesis, with Chapter 1 providing a descriptive introduction about the research, which includes the motivation, problem, scope of the study, methodology, and contribution. The Chapter 2 contains a comprehensive literature evaluation of the O-RAN architecture and security threats in O-RAN. In addition, a thorough review of FL, P2P FL, and FL-based anomaly detection is given as part of the current state-of-the-art. Chapter 3 describes the proposed solution, which consists of four models, and also provides how these models can be positioned within the O-RAN architecture. Chapter 4 contains information about the dataset used and how the simulations are set up and carried out. Different types of simulations and their results are given in Chapter 5. Moreover, these results are compared and

critically analyzed in this chapter. Chapter 6 consists of a discussion of the thesis, which includes a comparison with the current state-of-the-art, an evaluation of the thesis objectives, and finally, potential future work. Finally, a detailed conclusion to the thesis is provided in Chapter 7.

2 LITERATURE REVIEW

Network automation is becoming a necessary part of current network architectures due to their extremely high complexity and network demands. Closed-loop control based on RICs is a key feature in the O-RAN architecture, which can be utilized for the automation of network operations and management, including security functions. Security automation can be achieved by the combination of AI/ML methods and the closed-loop control in the O-RAN architecture.

2.1 O-RAN architecture

The proprietary RAN consisted of monolithic network components, where a limited number of vendors provided all the components in the RAN and operators had limited access to the internals of these components. This black-box approach had significantly effected the advancement of RAN for years with problems like configurable limitations of the RAN, Joint optimization of the RAN components due to the limited coordination among different network components and also the vendor lock-in [18].

As mentioned before, several new technologies have been introduced to tackle these limitations. C-RAN, which was introduced about 10 years ago, allows the functional digital processing functional part of the typical base station to be moved to a regional cloud or an edge cloud location. However, there is a significantly high communication overhead in the fronthaul link between the data center and radio units for maintaining the required low latency. Virtualized RAN (vRAN) is another approach that virtualizes RAN functions by replacing Baseband units (BBUs) with Commercial Off the Shelf (COTS) hardware. However, the Remote Radion Unit (RRU) is still on proprietary hardware with proprietary interfaces, and there was additional overhead due to virtualization. Hence, these two approaches did not solve the major drawback of the vendor lock-in [4].

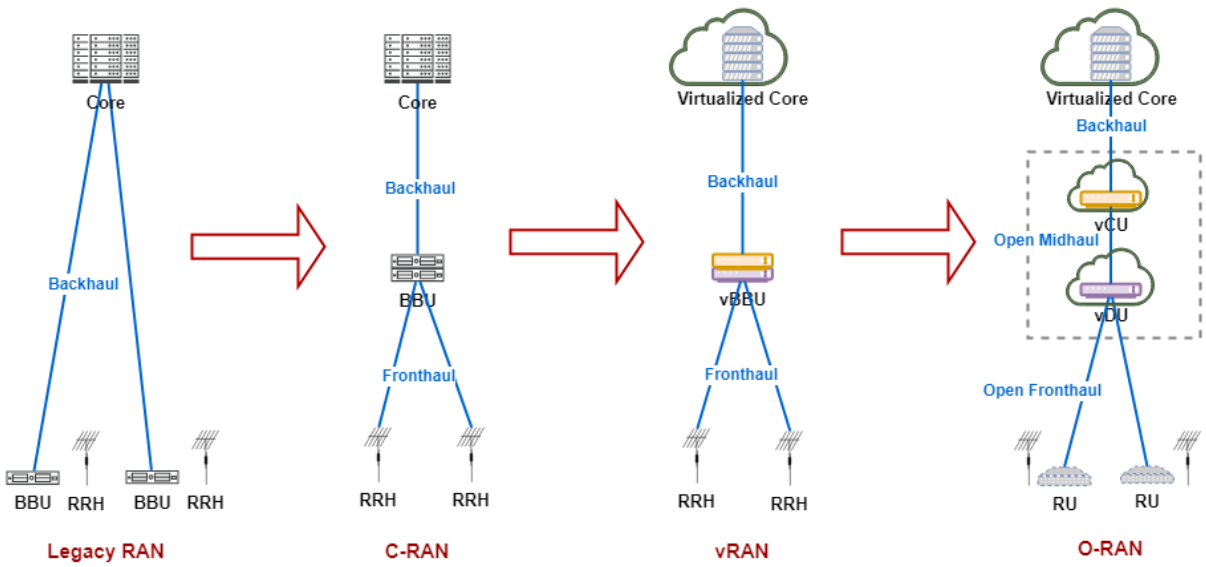


Figure 2.1: RAN architecture evolution [19].

The O-RAN alliance has taken the capabilities of both the C-RAN and vRAN and provided a general-purpose and vendor-independent solution called O-RAN that includes open interfaces between the components [6]. In the O-RAN architecture, cloud-native RAN functions can

be utilized via decoupled hardware and software components with the use of cloudification. This will allow to utilize many benefits of the cloud computing as opposed to the traditional RAN. Introduced intelligence and automation is for using advanced AI/ML capabilities to enable automated network management and orchestration in RAN with data-driven closed loops. Hence it will be possible to deploy non-conventional functions and techniques which were not possible before. Moreover, introduced open internal RAN interfaces support multi-vendor interoperability, allowing operators to optimize their infrastructure. Then the network operators would be able to jointly optimize their infrastructure with mixing and matching components from different vendors which will improve overall efficiency and cost reduction. This standardization of the interfaces is a crucial factor in eliminating the RAN vendor lock-in. Furthermore, flexibility and openness accelerate the delivery of new features and services [1].

The current 3rd Generation Partnership Project (3GPP) defined Next Generation NodeB (gNB) in the RAN is separated into two logical components called the Central Unit (CU) and the Distributed Unit (DU). In the O-RAN, these are further disaggregated according to 3GPP definitions, which result in three main logical units: CU, DU, and Radio Unit (RU) [20].

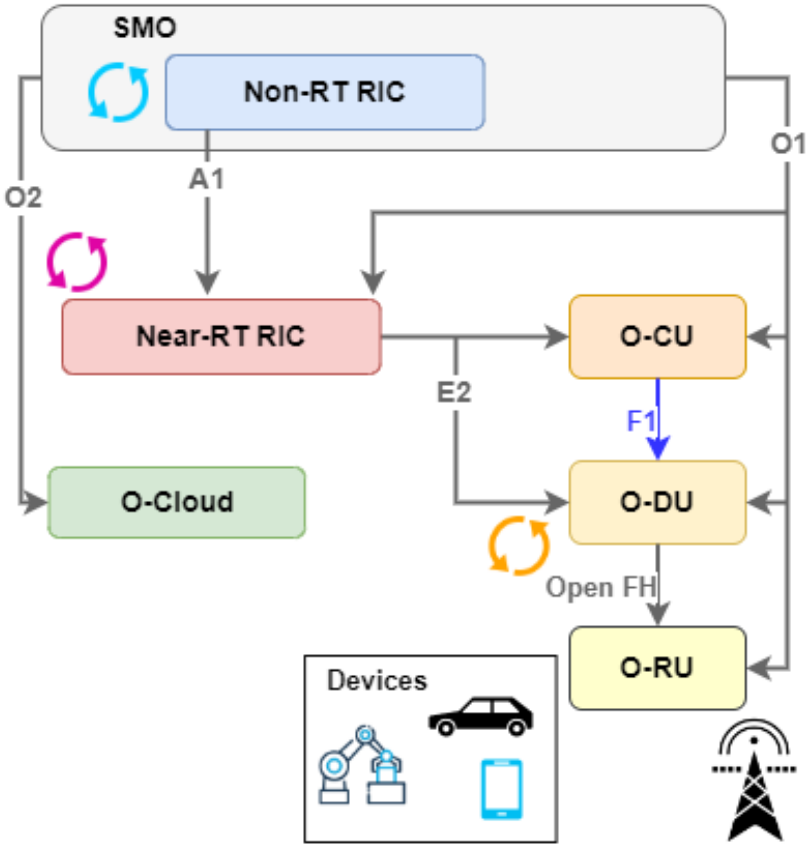


Figure 2.2: Logical Architecture of O-RAN with control loops.

- **CU:** This is the centralized unit, which can support multiple DUs. It includes the Radio Resource Control (RRC), Service Data Adaptation Protocol (SDAP), and Packet Data Convergence Protocol (PDCP) layers. This logical unit provides functionalities like session management, mobility control, transferring user data, etc. The midhaul interface is used for communication between CUs and DUs.

- **DU:** This logical unit, which is controlled by the CU, provides Radio Link Control (RLC), Medium Access Control (MAC), and some parts of the Physical (PHY) layer (High-PHY). Usually this distributed unit is located near the RU and communicates with RU via a front-haul interface.
- **RU:** This logical unit provides the digital front end and remaining parts of the PHY layer (Low-PHY).

The introduced split between DU and RU can be varied according to the use case; however, the split 7.2x is defined by the O-RAN alliance. This DU and RU separation has several benefits, such as cheaper RUs due to their fewer functionalities, efficient resource pooling by the DU, and the ability to manage and control a set of RUs at once [21].

The logical architecture of O-RAN is illustrated in Figure 2.2. O-cloud provides a pool of computing resources including COTS servers, hardware accelerators which are brokered by an abstraction layer to host logical network function mentioned by the introduced disaggregation of O-RAN architecture [22]. The deployment scenarios defined by the O-RAN alliance are illustrated in Figure 2.3. With this cloudification, the O-RAN architecture can take advantage of all the advantages of mature cloud computing, including the ability to share hardware among various tenants, standardise hardware characteristics for the O-RAN deployments, and automate the deployment and initialization of the RAN features.

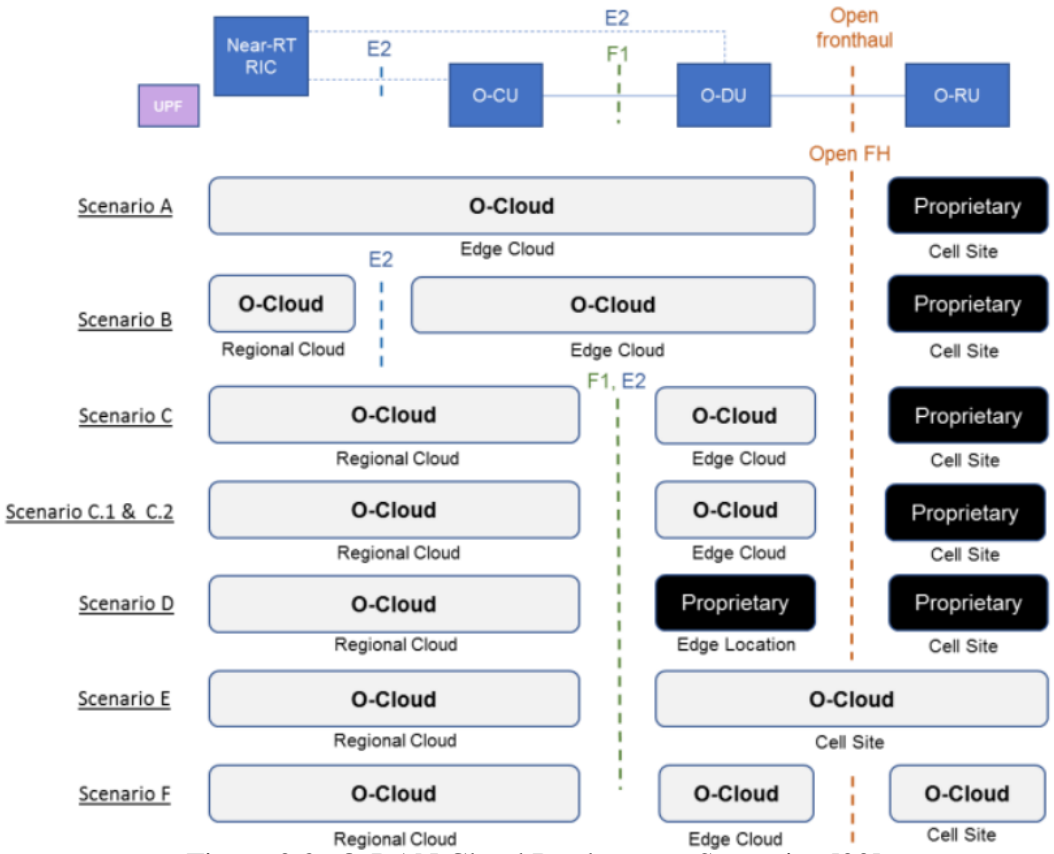


Figure 2.3: O-RAN Cloud Deployment Scenarios [22].

The main focus of the O-RAN architecture is open interfaces, and the O-RAN alliance is currently developing technical specifications for these interfaces between different components

of the O-RAN architecture. The standardization of these open interfaces will eventually eliminate vendor lock-in in the RAN. The new open interfaces in the O-RAN architecture are given in Table 2.1.

Table 2.1: Open interfaces in O-RAN

Interface	**Location**	**Task**
E2	Between Near-RT RIC and the E2 nodes	Enables the RIC to manage the operational processes of the E2 nodes.
O1	Between SMO and the Near-RT RIC, RAN nodes	Management and maintenance according to the FCAPS model.
A1	Between Non-RT RIC and Near-RT RIC	Allows the management of ML models and the deployment of policy-based guidance for the near-RT RIC by the non-RT RIC.
Open Fronthaul	Between DU and RUs inside the same gNB	Enables the control of RU operations from the DU and the distribution of physical layer functions between the RU and the DU.
O2	Between SMO and O-cloud	Enables programmatic management and provisioning of network operations.

2.1.1 RAN Intelligent Controllers

The O-RAN architecture consists of two logical controllers called RAN intelligent controllers (RICs). These components, which are based on software-defined networks (SDN), perform particular radio resource management tasks. Due to the network infrastructure's data stream, RICs have a centralized and abstract perspective on the network. In order to choose and implement control rules and actions on the RAN, these two can process data and make use of AI and ML methods [20].

The Non-RT RIC is a component of the Service Management and Orchestration (SMO) framework and operates on control loops longer than 1s. Near-RT RIC utilizes third-party micro-services called rApps to provide non-conventional services to facilitate the RAN optimizations and operations, such as providing policy-based guidance and enrichment information to the Near-RT RIC, intelligent orchestration of xApps and rApps, and ML model management. Moreover, it can influence SMO operation using policies [18].

The Near-RT RIC is at the centre of the RAN's control and optimization, operating the control loops with periodicities ranging from 10 ms to 1 s. It communicates with O-RAN-compliant evolved Node Bases, CUs, DUs, and E2 nodes (O-eNBs). The key element of Near-RT RIC is xApps, which are also third-party micro-services that can perform certain control or management tasks in the RAN [20]. For the functioning of the xApps, Near-RT RIC consists of a database containing RAN information, an internal messaging infrastructure, a subscription manager, and a conflict resolution mechanism [18].

RICs can be deployed in any of the cloud locations, namely the core cloud, regional cloud, and edge cloud. The RIC platform can be used to deploy external RAN control applications created by outside vendors. Compared to earlier proprietary RAN systems, the O-RAN architecture has a substantial benefit because these third-party apps can incorporate many types of cutting-edge RAN control algorithms [23].

The performance of the RAN can be affected by RICs in three main areas [5]:

- **Network intelligence:** The performance of the RAN is evaluated and reported, and the data that is generated in a standard format can be analyzed to develop new policies and algorithms, for instance, using AI/ML approaches.
- **Resource assurance:** The objective is to ensure that the required performance is delivered for the devices/services.
- **Resource control:** The aim is to ensure that the RAN system operates effectively when several user groups compete for available resources.

The main factor in O-RAN when using ML for RAN optimization is RICs. Previously, RAN analysis software conducted offline analysis on a large amount of acquired RAN data. However, O-RAN RICs can be used to analyze data, find improvements, and design and deploy new policies and actions based on the insights much more quickly [5].

In the future, there will be real-time control loops that operate below 10 ms and perform node level radio resource management tasks such as scheduling and beam-forming management.

2.2 Security Threats in O-RAN architecture

Although using the O-RAN architecture has several benefits, those architectural changes significantly alter the attack surface of the RAN. These are thoroughly discussed in this O-RAN Security Threat Modeling and Remediation Analysis document [10]. Additional functions and interfaces, decoupling and virtualization, and the usage of open-source codes have expanded the RAN threat surface. Furthermore, the effect of the diversity of User Equipment (UE), the diversity of third-party applications and the integration of AI/ML must be analyzed in terms of security threats [4].

Table 2.2: Threat surface groups [10]

Group	**Factors**
Additional functions	SMO, Non-Real-Time RIC, Near-Real-Time RIC
Additional open interfaces	A1, E2, O1, O2, Open Fronthaul
Modified architecture	Lower Layer Split 7-2x
Trust Chain	Decoupling, use of third-party xApp and rApps
Containerization and Virtualization	Disaggregation of software and hardware
Use of Open-Source Code	Exposure to public exploits may be increased

In O-RAN architecture, three main types of entry points are defined [10]. There are threats that are coming from inside the O-RAN architecture and threats that are coming from the outside. Furthermore, the API between different planes would also cause the propagation of threats.

Furthermore, six major threat agents who would deploy an attack were mentioned in the O-RAN security analysis documentation [10]. There can be insiders who have authorized access to the system who are malicious. Cybercriminals use their technical knowledge to commit crimes by exposing vulnerabilities via networks or devices with the purpose of gaining financial benefits. Cyber-terrorists are another type of threat agents who have the purpose of spread violence against particular country or group. Hacktivists are somewhat different; they would

attack O-RAN architecture to achieve some political or social gain. There are some attackers who would not have deep experience, knowledge, or resources, and they are called script kiddies. The final type of threat agent is a Nation-state, which has financial and technical support from a country to carry out attacks in the public and private sectors to steal information or compromise important processes.

There are various potential vulnerabilities, either particular to O-RAN or more generally, that might be used to jeopardize the confidentiality, integrity, and availability [10] as given in Table 2.3.

Table 2.3: Potential Vulnerabilities [10]

Uniqueness	**Vulnerability**	**Effects**
O-RAN specific	Illegal access to the O-DU, O-CU-CP, O-CU-UP as well as RU to compromise RAN performance or carry out a network attack.	Availability
O-RAN specific	Control plane traffic on the Open Fronthaul Interface are unprotected.	Integrity and Availability
O-RAN specific	Disable over-the-air ciphers causing eaves-dropping	Confidentiality
O-RAN specific	Conflict between Near-RT RIC and O-gNB	Availability
O-RAN specific	x/rApps conflicts	Availability
O-RAN specific	x/rApps access to network and subscriber data	Confidentiality
O-RAN specific	Management interface not secure	Confidentiality, Integrity and Availability
O-RAN specific	CP UL or DL messages can be injected to launch an attack against UP	Availability
General	Function decoupling without a hardware root of trust and a software trust chain	Integrity
General	Use of Open-Source code exposes system to public exploits.	Confidentiality, Integrity and Availability
General	Misconfiguration, inadequate access management or insufficient isolation in the O-Cloud platform	Confidentiality, Integrity and Availability

Moreover, there are 42 critical assets and 47 security threats in the O-RAN architecture. The threats can be grouped into six main types: threats against the O-RAN system, threats against the O-cloud, threats against the ML system, threats against 5G radio networks, threats to open source code, and physical threats [10]. Some of the threats included in these categories are given in Table 2.4.

The UNSW-NB15 dataset [11] includes the following attacks in addition to the threats mentioned:

- DoS: Attempt to disrupt or suspend the service in order to deny access to legitimate users. ICMP floods and buffer overflow are examples.
- Analysis: Port scanning and spamming are examples of these types of attacks.
- Backdoors: A method where the security mechanism of a system is bypassed using a

Table 2.4: O-RAN Threats [10]

Threat Categories	**Threats**
Threats against O-RAN system	Against the fronthaul interface and M-S-C-U planes, Against RICs, Against RUs, Against xApps and rApps, Against SMO, Common threats including those caused by poor authentication, setup errors, insecure designs, and a lack of access control.
Threats against O-CLOUD	VNF/CNF images and secrets can be compromised, Weak orchestrator configurations, Misuse of VM/CN to attack others, Spoofing and eavesdropping on network traffic, Compromise supporting network services
Threats to open source code	Backdoor attacks resulting from the usage of known-vulnerable SW components and unreliable libraries, Intentionally put backdoor by a developer
Physical Threats	Damage the components or steal sensitive data
Threats against 5G radio networks	Disruption through Radio Jamming, Sniffing and Spoofing, DoS attacks on cognitive radio networks
Threats against ML system	Data Poisoning Attacks, machine learning models can be altered, Transfer learning attack

hidden way.

- Exploits: Exploiting well-known security issues in the system, such as software issues.
- Reconnaissance: Attacks such as port scanning and packet sniffing where the information of a system is collected.
- Generic: A method that attacks block ciphers with an exposed block and key size.
- Shellcode: These are codes added to take advantage of software flaws.
- Fuzzers: These seek to stop a network or software function by feeding it data that is created randomly.
- Worms: A type of computer software that replicates itself and/or spreads to other parts of a system.

2.3 Federated Learning

The confidentiality of the data being processed when using AI/ML is a top priority. Since RAN data contains sensitive information, FL, which is a privacy-preserving distributed technique, is preferred. Developed by Google, FL is used to train models utilizing a large number of devices [14]. At each trainer location in the federation, a local model is trained using the training dataset, which is retained locally. Only the model parameters are sent to the centralized aggregator site after the local model training. The aggregator uses the received parameters to create a common global model and feeds it back to the local trainers. Since sensitive data is not explicitly accessed, each local trainer can benefit from the datasets of other local trainers without affecting privacy and while reducing the communication cost [24].

The problem of optimization is also available in FL and in [14], the model proposed for non-IID, unbalanced, and massively distributed data. And its optimization problem is given as,

$$min_{w \in R(D)} f(w) \tag{2.1}$$
$$f_i(w) = l(x_i, y_i : w) \tag{2.2}$$

where $f_i(w)$ is the loss of the prediction of data (x_i, y_i) for the weight set w. Considering FL, this can be mathematically derived as

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \tag{2.3}$$

where

$$F_k(w) = \frac{1}{n_k} \sum_{k \in p_k} f_i(w) \tag{2.4}$$

K is the total number of clients in the federation, p_k is the available number of data samples for the k^{th} client, and $n_k = |p_k|$. As the results shown in this study, the simple averaging techniques provide adequate accuracy.

The lifecycle of a FL model is explained in [13] where the model engineer identifies the problem first, followed by clients storing data locally for training. Then a model is prototyped and sent for training. The received trained models are evaluated and The best-performing model is distributed across all client devices. The reference lifecycle FL-trained model is shown in Figure 2.4. The amount of clients chosen for a particular round can be random, but there are several studies such as [25] based on optimization to get a optimal value.

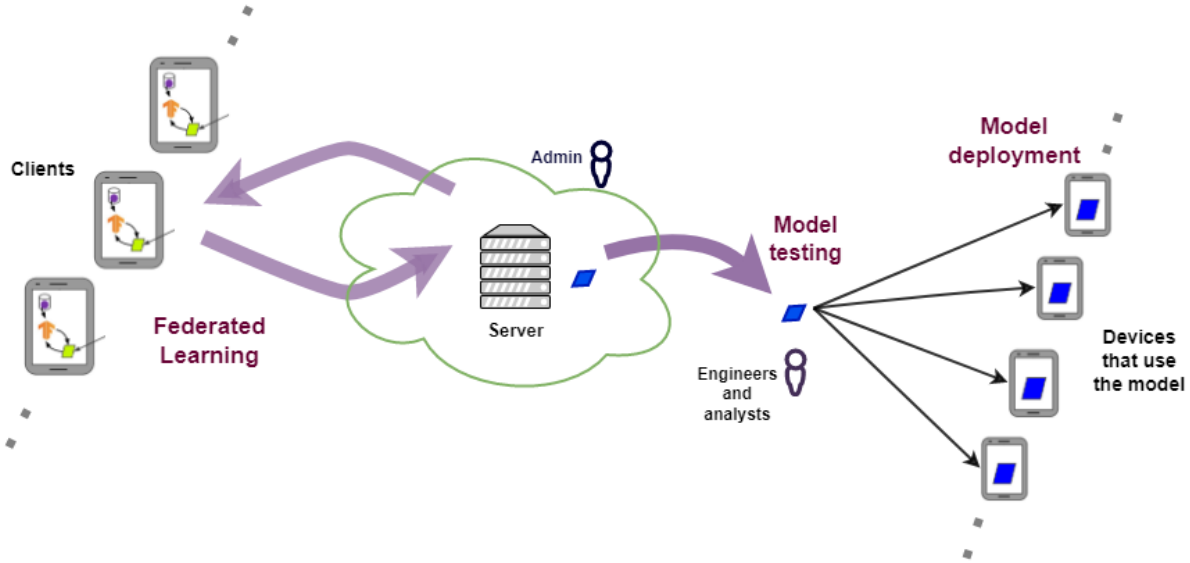


Figure 2.4: The lifecycle FL-trained model [13].

There are several possible options for ML models to select, and Artificial Neural Networks (ANN) are one of the more preferred models for complex systems. Being a subclass of ML, the neural networks imitate the human brain [26]. There are several variations of ANNs, including Multilayer perceptron (MLP), convolutional neural networks (CNN), and recurrent neural networks (RNN). As mentioned before, the MLP model was used for this study. Generally, MLP is a feed-forward ANN that is fully connected. An MLP model consists of three major components: an input layer, an output layer, and a hidden layer or layers. In addition to those, there can be activation and optimization functions.

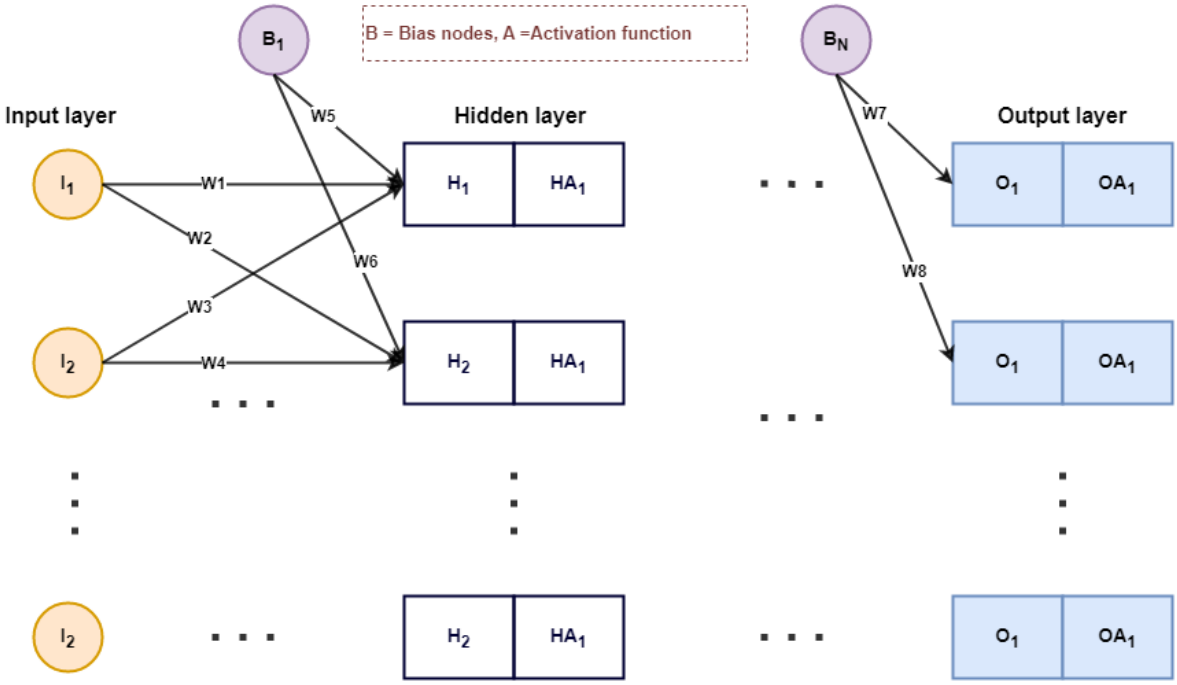


Figure 2.5: General MLP architecture [26].

There are several wireless communication applications that can utilize FL, such as spectrum management, edge computing, autonomous driving, and 5G core network functioning [24]. When there are a lot of components in a system that are essential for a required process, as in most scenarios in wireless communications systems, FL tends to be the preferred approach to use for AI-based applications.

The ability to use FL for the O-RAN architecture is described in detail in ”AI/ML workflow description and requirements” technical report [27]. Both Near-RT RIC and Non-RT RIC in the O-RAN architecture can host the AI/ML training by utilizing xApps and rApps, respectively. Since RICs are already organized in a hierarchical manner, FL can be directly integrated. The Near-RT RIC, which resides in the regional or central cloud, can act as the central aggregator. At the same time, Non-RT RICs, which are located in edge or regional clouds, can serve as distributed local trainers [27] as shown in Figure 2.6.

Although there are several benefits to FL, it also has considerable drawbacks. Even though FL improves data privacy, there is a possibility of analyzing global data and exposing the clients participated [24]. FL is also vulnerable to poisoning attacks by attackers who have access since there is model re-training with new available data [28]. As an example, the training data could be altered to get the results preferred by a particular attacker. Moreover, FL models could be exposed to membership inference attacks, where a malicious client affects the model’s

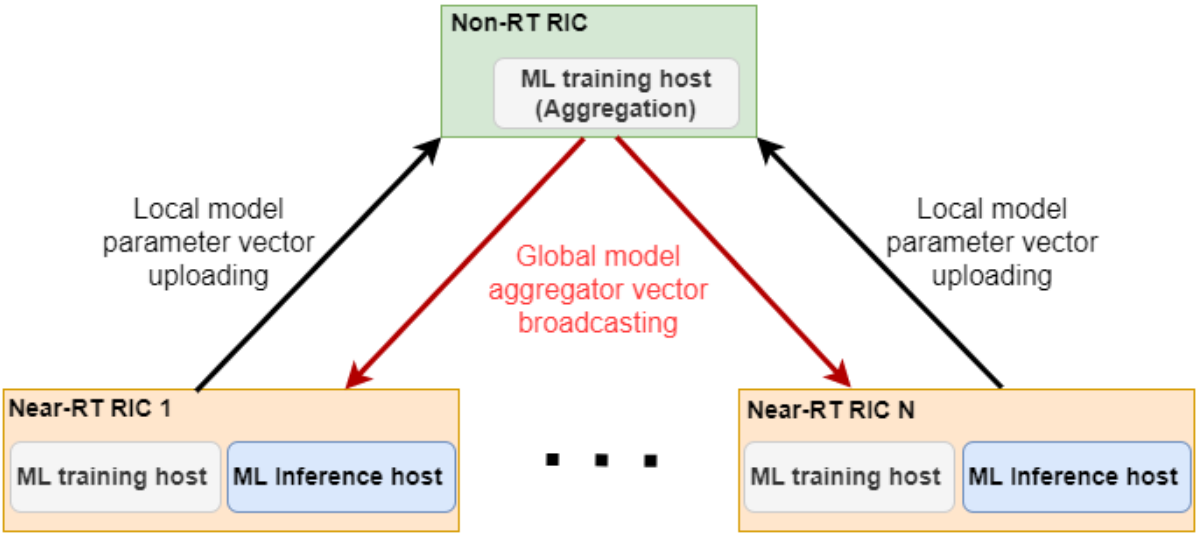


Figure 2.6: Federated learning among Non-RT RIC and Near-RT RICs.

training by masquerading as an honest client [29]. There are different kinds of inference attacks, including parameter inference and input inference.

Furthermore, there are challenges that are relevant to the algorithms; Some may be related to concerns of convergence and optimization, like the selection of optimal number of local clients [24]. Also, there are problems specific to wireless communication, including parameter quantization due to limited network capacity, which will cause quantization errors [24]. In [8] the applicability of FL in 6G is explained. There are some security challenges that can be overcome by using FL.

Several remediations for the aforementioned issues have been proposed, including the communication-efficient FL algorithm [8]. Asynchronous FL systems such as the ones proposed in [30], [31], [32] improve communication efficiency by asynchronous aggregation. Some other methods to improve communication efficiency are gradient compression using stochastic gradient descent (SGD) algorithms [33], gradient quantization, and sparsification [34].

To prevent malicious attacks in FL model systems, robust aggregation and detection algorithms as well as management of reliable reputations are proposed in [35]. Blockchain-based smart contracts and deferential privacy (DP) techniques can also be used. Furthermore, techniques such as federated parallelization and distillation can be used to increase the effectiveness of FL [36].

Considering the usability of FL in O-RAN, an FL set-up for O-RAN is proposed in [37] where the system model is defined as Near-RT RICs in edge clouds hosting local models and Non-RT RICs in regional clouds hosting the aggregator with update vectors maintained in a synchronized manner in global rounds. In this research, A combined optimization model for selecting local trainers and allocating resources for FL is derived. An FL-based deep reinforcement learning (DRL)-based solution for O-RAN user access control is given in [38] where each UE acts as a local trainer of the local deep Q-network (DQN), and RIC is responsible for averaging the DQN parameters received from chosen UEs in order to update the global DQN model. A federated meta-learning-based traffic steering in O-RAN is proposed in [39] which utilizes RL for allocating radio access technology (RAT) services between UEs.

A practical solution of deploying ML in O-RAN is given in [40]. A closed-loop control is deployed using a RIC platform called ”ColO-RAN” and E2 nodes running in a framework called ”SCOPE” with network data generated by an emulator called ”Colosseum.” Furthermore, three testing xApps were built that provide DRL-based control over slicing of RAN resources as well

as scheduling.

2.3.1 Peer-to-Peer FL

As mentioned before, the centralized FL method has some considerable drawbacks, including a single point of failure due to the centralized aggregator and an imbalance of data distributions in different local trainers [13]. Also, only the central aggregator has bi-directional communication with the trainers, and the local trainers have limited knowledge about other trainers connected, which can be a vulnerability [12]. Hence, privacy-preserving FL methods have gained significant interest. Mainly three approaches are considered in privacy-preserving FL, namely Secure Multi-Party Computation (SMC), Homomorphic Encryption (HE) and Differential Privacy (DP) [41]. The SMC approach aims on calculating a required function without each participant knowing the other's exact inputs. A privacy-preserving FL framework based on SMC is proposed in [42] where users' gradients are securely aggregated. HE enable performing computations on encrypted values by third-parties without having to perform encryption. In [43], a privacy-preserving FL model that is based on HE is given, which takes advantage of the additive property of HE. DP can also be used to protect privacy of the participant in FL-based methods [44] and based on that an adaptive privacy-preserving FL framework is proposed in [41].

P2P FL is a novel technique that eliminates the need for a centralized system or service [12]. There are several proposed techniques for P2P FL model training that can use some of the aforementioned approaches for privacy-preserving FL. Split learning (SL), mentioned in [45] splits the model into two parts, the server and the node, for training. The training process is considerably slower when multiple nodes are involved since SL methods are difficult to parallelize and must run sequentially by design. BrainTorrent is a P2P FL environment that is decentralized [46]. At any given training round, just one participant updates their local model weights by taking into account their prior weights and model weights received from peer models that are apparently newer, and there is a risk of a malicious or semi-honest participant being undetectable. The peer-to-peer FL method [12] based on secure average computation (SAC) utilizes an n-out-of-n secret partitioning method and average calculation technique, which mitigates the effects of semi-honest participants. An example of calculating the average with having three peers is shown in Figure 2.7. However, this model is not optimal when there is a larger number of trainers since the communication cost is significantly higher, which reduces the overall performance.

Multiparty computation based on HE is another approach that can be used for parameter averaging in P2P FL. There different solutions have being proposed to achieve privacy while multiparty computation. A secure multiparty computation based on FHE in a environment where a untrusted cloud server with high resources is proposed in [47]. Another general multiparty communication protocol which uses somewhat homomorphic encryption is provided in [48] which can provide security against even $n - 1$ pout of n participants are corrupted. Another multiparty communication solution based on threshold homomorphic encryption is given in [49] which can be used against active and static adversaries. There is a significantly higher computation cost when using HE for multiparty computation due to encryption and decryption.

	Peer1	Peer2	Peer3
Model weights	$w_1 = 25$	$w_2 = 19$	$w_3 = 37$
1.a) Initialize secret sharing	$rn_{11} = 10$ $prn_{11} = 0.2$ $rn_{12} = 20$ $prn_{12} = 0.4$ $rn_{13} = 20$ $prn_{13} = 0.4$	$rn_{21} = 9$ $prn_{21} = 0.16$ $rn_{22} = 18$ $prn_{22} = 0.32$ $rn_{23} = 30$ $prn_{23} = 0.52$	$rn_{31} = 30$ $prn_{31} = 0.41$ $rn_{32} = 20$ $prn_{32} = 0.27$ $rn_{33} = 24$ $prn_{11} = 0.32$
1.b) Create parts	$w_{11} = 5$ $w_{12} = 10$ $w_{13} = 10$	$w_{21} = 3$ $w_{22} = 6$ $w_{23} = 10$	$w_{31} = 15$ $w_{32} = 10$ $w_{33} = 12$
1.c) Share parts	$w_{21} = 3$ $w_{31} = 15$	$w_{12} = 10$ $w_{32} = 10$	$w_{13} = 10$ $w_{23} = 10$
2. Subtotals	$ps_1 = 23$	$ps_2 = 26$	$ps_3 = 32$
3. Totalization and averaging	$S = 81$ $Avg = 27$	$S = 81$ $Avg = 27$	$S = 81$ $Avg = 27$

Figure 2.7: Secure average computation with three clients [12].

2.4 FL-based anomaly detection

The novel features of 5G and B5G cause the current detection technologies to be obsolete if they are not adapted accordingly [50]. Because the communication link between the UE and the remote radio head unit has remained mostly unchanged, both conventional RAN and O-RAN utilize Radio Frequency (RF) channels that are fairly similar. Therefore, O-RAN is vulnerable to some known RF attacks on conventional RAN. Furthermore, the diversity and pervasiveness of UE types are expanded in O-RAN, increasing the chance that the attack surface will be a target of new threats. Also, there is the risk of DoS and Distributed Denial of Service (DDoS) attacks on cellular services and control planes, as well as DDoS flooding attacks on network and transport layers [4]. Therefore, detecting anomalies is an important security measure to be considered in O-RAN. A data-driven FL technique is ideal when considering a complex system like a RAN in 5G architecture, where there are many ways to cause anomalies.

Considering ML-based anomaly detection in RAN, a two-level deep ML model for 5G network architecture is proposed in [50]. The first layer is for anomaly symptom detection, with models implementing Deep Belief Networks (DBN) and Stacked Autoencoders (SAE) at each RAN. The network anomaly detection is done centrally with the received anomaly symptoms in the second layer using the Long short-term memory (LSTM) model. In [7], an AI-based solution for detection of anomalies and analysing the root causes in RAN is proposed, with a novel anomaly detection algorithm called Soda and two-level root cause analysis based on key performance indicators (KPIs) and key quality indicators (KQIs). Also, in a trial of O-RAN architecture, its closed-loop network automation technique with the given framework for capacity problems has been validated.

There are several FL-based anomaly detection solutions proposed for the IoT. The anomaly detection method presented in [15] is based on calculating the probability associated with device communication patterns. Built for a small office environment, this solution includes a security gateway consisting of anomaly detectors and an IoT security service maintaining a repository of models. The IoT intrusion system, which is based on unsupervised FL, is presented in [51]. It consists of an autoencoder and an ANN model. This system successfully detected various anomalies such as DoS attacks, botnets, and port scans, and a maximum accuracy of 97.75% was achieved. Another federated learning framework for industrial IoT applications is given in [16] where a Deep Anomaly Detection (DAD) model is trained. This framework includes

a cloud aggregation server and local devices for anomaly detection. A gradient compression mechanism was also suggested, which makes communication faster and more efficient.

A reinforcement learning based FL approach for detecting anomalies in 5G heterogeneous networks is proposed in [52]. This includes three main entities: the end devices for model training, the edge hosting the aggregator server for the trained model in the end devices, and also separate anomaly detection models. The models at the edge are aggregated in the cloud.

In addition, a method named DeepFed is presented in [53] and uses a federated deep learning architecture with a Paillier public-key cryptography based secure communication protocol to identify intrusions in industrial cyber-physical systems. DeepFed is composed of three main parts: a trusted authority to generate public keys and private keys, cloud servers for aggregation, and industrial agents who host local models to train available data. This system can fight against several attacks, including DoS, DDoS, and injection attacks.

A hierarchical FL based anomaly detection mechanism which can be used in the ZSM framework is proposed in [17] where the UNSW-NB15 dataset is used. In stage 1, the anomaly detectors (A-type), each consisting of a simple ML model and a database, are in the same security management domain. In stage 2, the anomaly detectors (B-type) are in different security management domains. The aggregation server of stage one is placed in the security management domains, and the end-to-end (E2E) management domain hosts the second stage aggregator server. Once the network flow has been handled in stage 1, it will be handled again in stage 2, and any problems that are found will be fixed right away.

There is an anomaly detection use case [54] presented by the O-RAN Software Community. However, the main objective is to identify the anomalous user equipment (UE) in a certain cell and, if necessary, initiate a handover to a neighboring cell. In this scenario, being anomalous means degradation of one of the considered parameters is below a pre-defined threshold, and three xApps, namely, Anomaly Detection (AD), QoE Predictor (QP) and Traffic Steering (TS) in Near-RT RIC, are used.

3 PROPOSED SOLUTION

The intelligent closed-loop control with RICs enables the direct use of ML in the O-RAN architecture. As mentioned previously, FL is more suitable in RAN due to its increased privacy. In this study, SAC based [12] P2P FL method with two variations based of clustering and HE based P2P FL method is proposed. Training of the designed FL models and anomaly detection are the main components of this study.

In all of these methods, the local trainers are Near-RT RICs which reside in edge cloud or regional cloud. The Near-RT RICs host the training, and P2P communication is via inter-regional or inter-edge cloud connections. The training model and the detector can be separate xApps, or they can be parts of the same xApp, which can include other functions. These deployments can be image-based or file-based [27]. Moreover, when an anomaly is detected, the detector or an additional xApp hosted by Near-RT RIC can transmit control actions to the relevant CUs and DUs via the E2 interface. The general proposed architecture is shown in Figure 3.1.

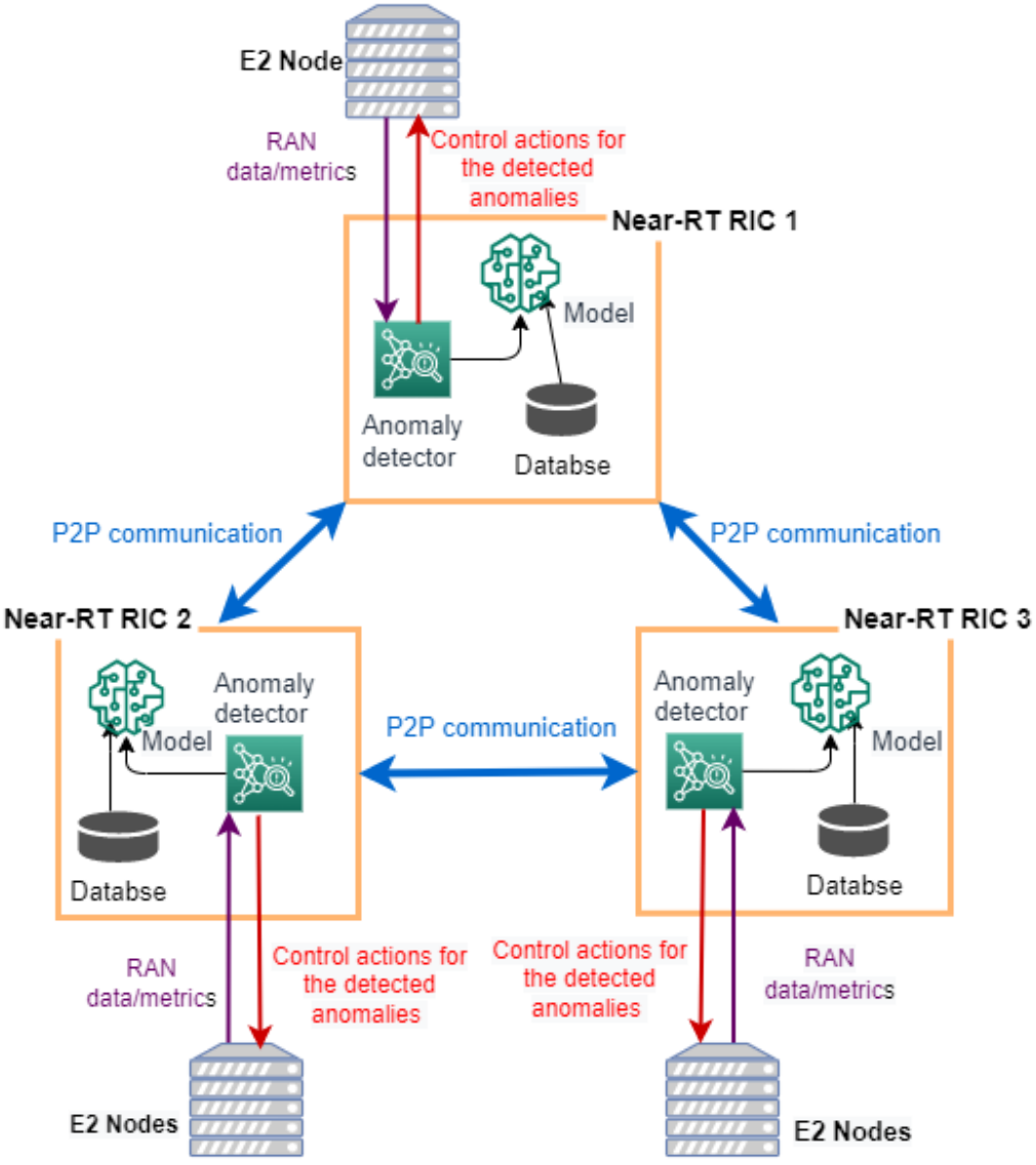


Figure 3.1: Proposed anomaly detection mechanism mapped with O-RAN architecture.

The network flows data that is stored in the database and can be used for the model's training. After model training, it can be utilized by the anomaly detector in the same Near-RT RIC. Control actions for the detected anomalies are communicated to the E2 nodes.

3.1 Model 1: Normal P2P FL method

In the normal P2P FL method, every trainer communicates with every other to calculate average model weights using secure average computation [12] as shown in Figure 3.2.

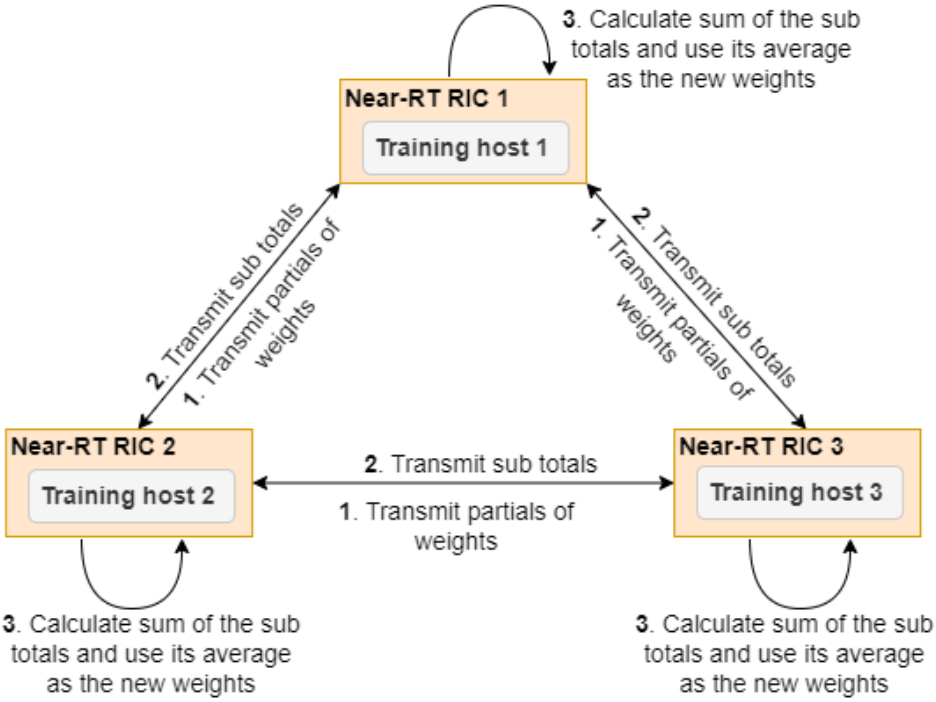


Figure 3.2: Secure weight average computation in P2P FL.

When a local trainer trains its model, the model parameter is randomly partitioned into partial weights that are equivalent to the network's number of local trainers which is shown in Equation 3.1.

$$w_i = \sum_{k \in Trainers} w_{ik} \tag{3.1}$$

Then each trainer keeps one partial weight and transmits the rest of the partial weights to other local trainers, one per trainer. Thereafter, the local subtotal is calculated by each trainer with its own and the received partial weights, as shown in Equation 3.2.

$$t_i^{partial} = \sum_{j \in Trainers} w_{ji} \tag{3.2}$$

One full round is completed after computing the new weight, which is calculated as the

average of all previously calculated weights as shown in Equation 3.3.

$$w_{avg} = \frac{\sum_{i \in Trainers} t_i^{partial}}{\text{total number of trainers}} \tag{3.3}$$

3.2 Model 2: Clustered P2P FL method

In practical RAN deployments, data is unbalanced and localization (in edge or regional clouds) is probable. Hence, parameter averaging between the training models of all the Near-RT RICs in a massive base station deployment can be considered sub-optimal. Therefore, a clustered model is proposed where only the clients in the same cluster share the parameters for averaging, which results in different FL models for each cluster. Local trainers are clustered using location-based K-means clustering using scikit-learn library function [55].

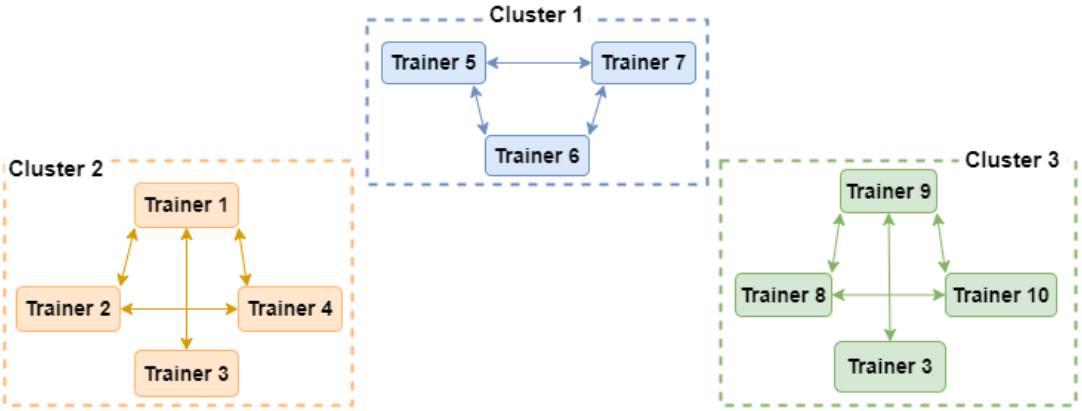


Figure 3.3: Parameter sharing of Clustered P2P FL.

3.3 Model 3: Hierarchical P2P FL method

In the hierarchical P2P FL method, a hierarchical version of the clustered model is considered, where a master trainer is pre-selected for each cluster by considering a pre-calculated resource reference value. After a fixed number of global rounds, parameter values are shared and averaged among cluster masters, with weight values corresponding to the number of clients in the cluster, as shown in Equation 3.4. Then, the values are shared with the other clients in the cluster by the master client. The probability of having the same model in each cluster will increase while maintaining a lower number of communications than the proposed normal P2P FL model.

$$w_{avg} = \sum_{k \in \text{clusters}} \frac{\text{number of trainers in k}}{\text{total number of trainers}} w_k \tag{3.4}$$

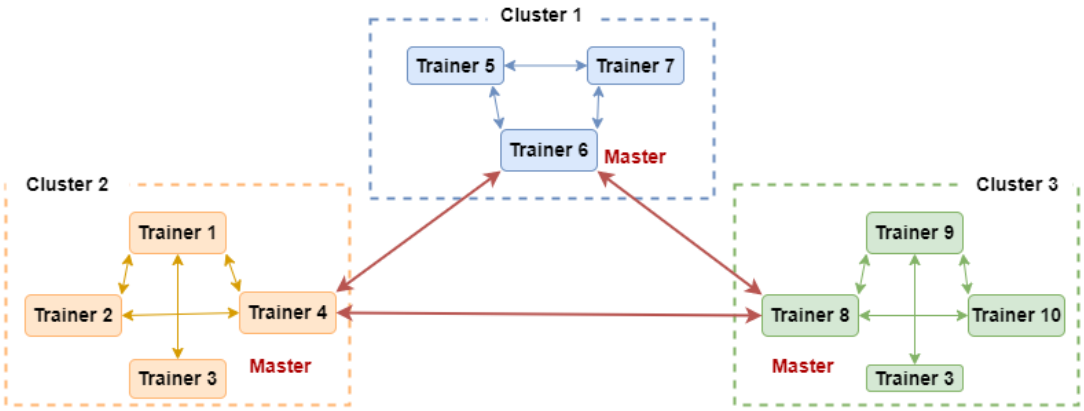


Figure 3.4: Parameter sharing of Hierarchical P2P FL.

3.4 Model 4: Homomorphic P2P FL method

In addition to the above methods, which are based on SAC, a parameter averaging method with secure communication and the threshold HE method [47] was considered. A key generation protocol can be used to generate a common public key (P_k) and secret shares of the private key (S_k), with each trainer receiving the P_k and a share of S_k. It means that each trainer can encrypt, but no one can decrypt without the help of other trainers. Hence, for parameter averaging in P2P FL, each trainer can encrypt parameter values and share them with each other. Then do homomorphic addition to get the encrypted total values, followed by partial decryption using individual secret key share. After that, these partial decryption values are shared with each other, and the final decryption of the total is performed at each trainer, and the average can be calculated. This provides secure communication of parameter values and protection against semi-honest trainers. This procedure is given in Figure 3.5.

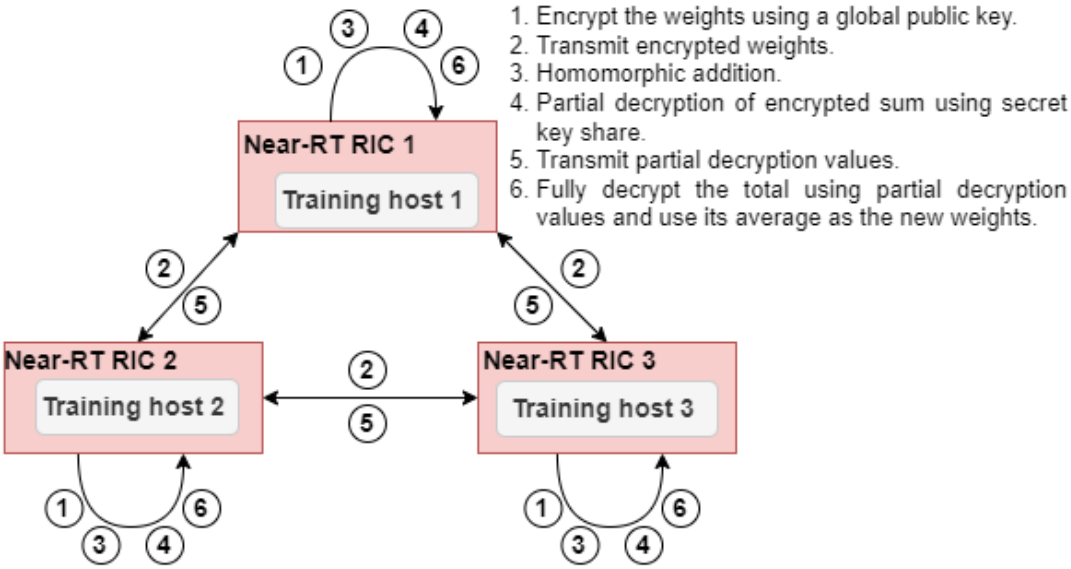


Figure 3.5: Weight average computation using HE.

4 SIMULATIONS

The Python scripting language is used for the simulation. And the FL model's creation, compilation, training, and testing were done using Tensorflow libraries. Initial simulations with 10 local trainers were performed on a personal computer using Jupyter notebooks. However, when the number of users increased to 100, the simulation time increased significantly. Hence, the simulation functions are modified to utilize the multiprocessing library in Python to perform parallel model training and testing of local trainers. Due to the limited processing power the simulations were migrated to the Puhti supercomputer environment [56] provided by "CSC – IT Center for Science Ltd". In that environment, the simulations were run as batch jobs, which included python scripts as opposed to Jupyter notebooks. In Puhti, the already available "tensorflow/2.9" environment module was used as the simulation environment, and some of the packages included are depicted in Figure 4.1.

Package	Version
absl-py	1.1.0
affine	2.3.1
aiosignal	1.2.0
alabaster	0.7.12
anyio	3.6.1
argon2-cffi	21.3.0
argon2-cffi-bindings	21.2.0
arrow	1.2.2
astroid	2.11.6
astunparse	1.6.3
atomicwrites	1.4.0
attrs	21.4.0
autokeras	1.0.19
autopep8	1.6.0
Babel	2.10.3
backcall	0.2.0
bayesian-optimization	1.2.0
beautifulsoup4	4.11.1
binaryornot	0.4.4
black	22.3.0
bleach	5.0.0
cachetools	5.2.0
category-encoders	2.5.1.post0
certifi	2022.6.15
cffi	1.15.0
cftime	1.6.0
chardet	4.0.0
charset-normalizer	2.0.12
click	8.0.4
click-plugins	1.1.1
cligj	0.7.2
cloudpickle	1.6.0
colorama	0.4.4
ConfigSpace	0.5.0
cookiecutter	2.1.1
cryptography	37.0.2
cycler	0.11.0
Cython	0.29.30

Figure 4.1: Part of the packages installed in simulation environment.

4.1 The UNSW-NB15 dataset

This dataset is generated as a combination of real normal network activities and artificial attack behaviors [11]. There are 257673 total network data flows with 49 classified features, including transaction protocol, source and destination IP addresses and ports, and service. All the features are given in Figure 4.2.

SNo.	Name	Type	Description
36	is_sm_ips_ports	Binary	"If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal thenthis variable takes value 1 else 0"
39	is_ftp_login		If the ftp session is accessed by user and password then 1 else 0.
49	Label		0 for normal and 1 for attack records
7	dur	Float	Record total duration
15	Sload		Source bits per second
16	Dload		Destination bits per second
27	Sjit		Source jitter (mSec)
28	Djit		Destination jitter (mSec)
31	Sintpkt		Source interpacket arrival time (mSec)
32	Dintpkt		Destination interpacket arrival time (mSec)
33	tcprtt		"TCP connection setup round-trip time, the sum of synack and ackdat."
34	synack		"TCP connection setup time the time between the SYN and the SYN_ACK packets."
35	ackdat		"TCP connection setup time the time between the SYN_ACK and the ACK packets."
2	sport	Integer	Source port number
4	dsport		Destination port number
8	sbytes		Source to destination transaction bytes
9	dbytes		Destination to source transaction bytes
10	sttl		Source to destination time to live value
11	dttl		Destination to source time to live value
12	sloss		Source packets retransmitted or dropped
13	dloss		Destination packets retransmitted or dropped
17	Spkts		Source to destination packet count
18	Dpkts		Destination to source packet count
19	swin		Source TCP window advertisement value
20	dwin		Destination TCP window advertisement value
21	stcpb		Source TCP base sequence number
22	dtcpb		Destination TCP base sequence number
23	smeansz		Mean of the ?ow packet size transmitted by the src
24	dmeansz		Mean of the ?ow packet size transmitted by the dst
25	trans_depth		Represents the pipelined depth into the connection of http request/response transaction
26	res_bdy_len		Actual uncompressed content size of the data transferred from the servers http service.
37	ct_state_ttl		No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd		No. of flows that has methods such as Get and Post in http service.
40	ct_ftp_cmd		No of flows that has a command in ftp session.
41	ct_srv_src		No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42	ct_srv_dst		No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct_dst_ltm		No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ ltm		No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm		No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct_dst_sport_ltm		No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
47	ct_dst_src_ltm		No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
1	srcip	Nominal	Source IP address
3	dstip		Destination IP address
5	proto		Transaction protocol
6	state		Indicates to the state and its dependent protocol
14	service		"http ftp smtp ssh dns ftp-data
48	attack_cat		"The name of each attack category. In this data set nine categories *eg* Fuzzers Analysis Backdoors DOS exploits Generic Reconnaissance Shellcode and Worms"
29	Stime	Timestamp	record start time
30	Ltime		record last time

Figure 4.2: Part of the packages installed in simulation environment [57].

At the data pre-processing stage, some of the features including source and destination IP addresses and attack categories were removed. And some of the nominal data features, such as

transnational protocol, service, and state, are converted to numerical features using categorical encoder functions. The testing dataset of 10,000 network flows with a 60% anomaly percentage was separated prior to the simulation.

4.2 Simulation process

In this study, ML models are trained to detect whether the data is an anomaly or not. Each method mentioned was created as a function so that it was possible to perform simulations while changing different parameters. For that, a separate Python script was created that included data loading, pre-processing, local trainer clustering, and also calling the methods as functions. Moreover, three types of client data distribution are considered.

- **Random:** Each trainer has a training dataset with a random number of anomalies that form a small percentage of the given system anomaly percentage.
- **IID:** Each trainer has a training dataset with the same number of anomalies (system anomaly percentage).
- **Non-IID:** trainers in the same cluster has a training dataset with an equal number of anomalies.

The ML model used had four layers, including two hidden layers, as depicted in Figure 4.3. Each trainer uses ten epochs for local training. The simulations are performed with different values of anomaly percentages in the training data, the local batch size, as well as the number of rounds.

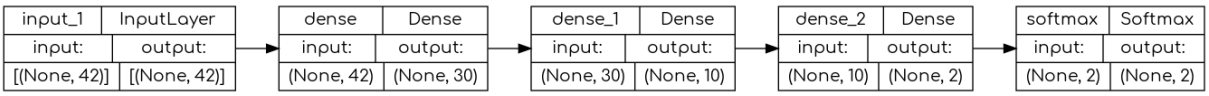


Figure 4.3: MLP model used.

During each simulation, the trained models are tested using the test dataset. And the performance is tested in terms of accuracy, F1-score Also, transmission costs for training are considered separately.

The accuracy, which is given by 4.1 is the ratio between the number of correctly classified samples and the total number of samples given.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

F1-score provides a balanced metric of the other two metrics, called precision and recall, where typically one's increase means the other's decrease. The precision is the ratio of true predicted positives to all the predicted positives. Higher precision means the system does not produce significant amount of false positives. And the recall is the ratio between the true predicated positives and all the actual positives. Hence, higher recall means the system does not produce a large number of false negatives. And the F1-score is the weighted average of these two.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1\text{-}Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{4.4}$$

5 RESULTS

5.1 Performance of SAC-based P2P FL models

5.1.1 Varying the batch size

First, the batch size taken for training the local model was changed, and the variation in accuracy was obtained. As shown in Figure 5.1 the accuracy keeps getting reduced as the batch size increases. Hence the training time was also plotted to select an optimum batch size to continue the simulations further. The training time keeps reducing until the batch size is about 500, at which point it starts to converge. As a result, 100 was chosen as the fixed batch size for subsequent simulations as a significantly optimal point.

Table 5.1: Fixed parameters for the simulations with varying training batch size

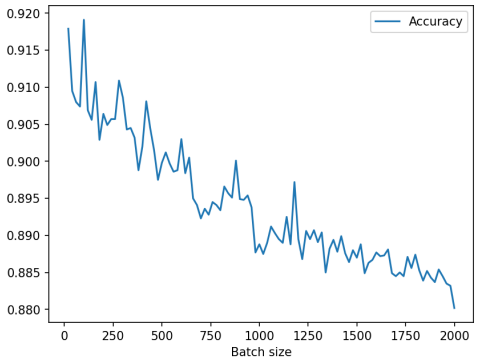
Parameter	**Value**
Number of trainers	100
Number of clusters	5
Number of epochs	10
Number of rounds	40
Batch size	100
Training sample size	200000
Training anomaly percentage	60%
Testing sample size	10000
Testing anomaly percentage	60%

5.1.2 Varying the training anomaly percentage

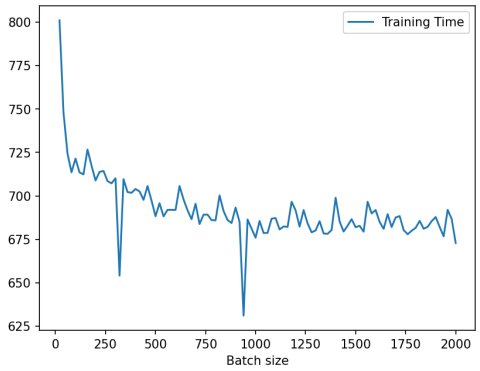
Then the performance of the methods was assessed when the training anomaly percentage was varied. Here only the IID data distribution is considered, and all the other fixed parameters are given in Table 5.2.

Table 5.2: Fixed parameters for the simulations with varying training anomaly percentage

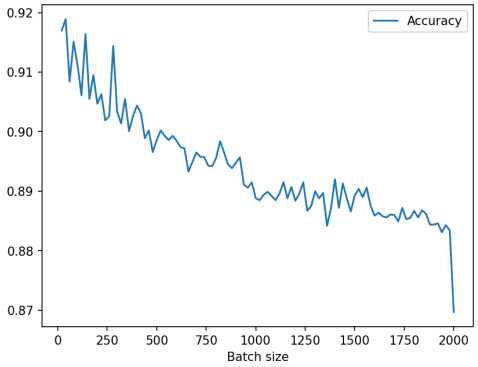
Parameter	**Value**
Number of trainers	100
Number of clusters	5
Number of epochs	10
Number of rounds	50
Batch size	100
Training sample size	100000
Testing sample size	10000
Testing anomaly percentage	60%



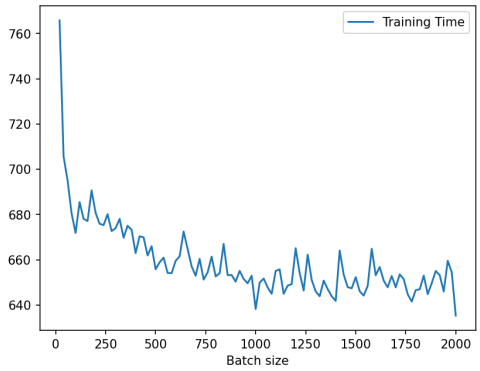
(a) Centralized FL model: Accuracy.



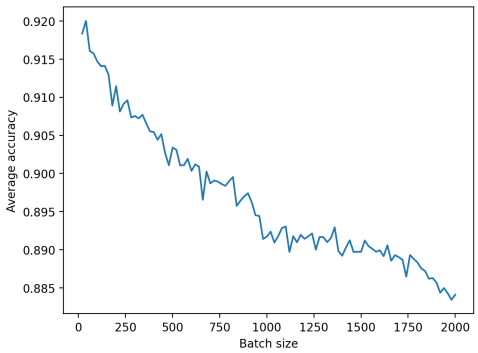
(b) Centralized FL model: Training time.



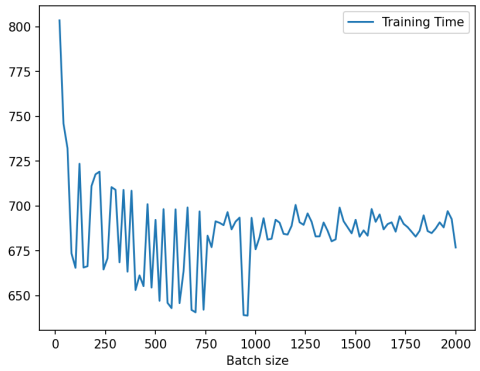
(c) Normal P2P FL model: Accuracy.



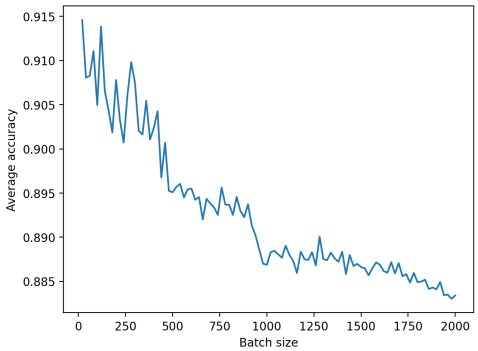
(d) Normal P2P FL model: Training time.



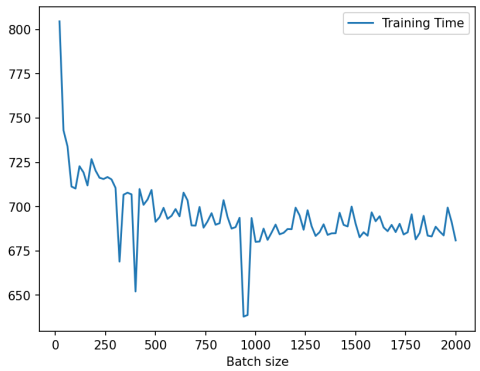
(e) Clustered P2P FL model: Accuracy.



(f) Clustered P2P FL model: Training time.

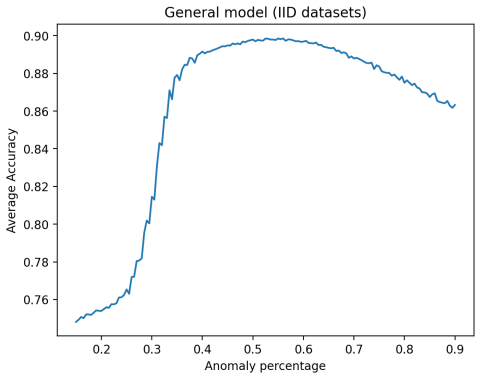


(g) Hierarchical P2P FL model: Accuracy.

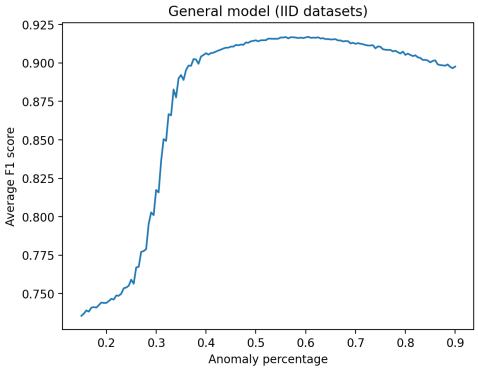


(h) Hierarchical P2P FL model: Training time.

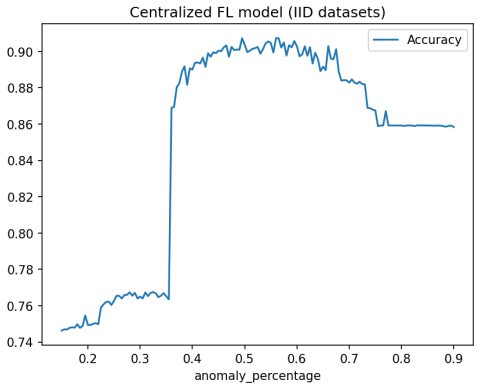
Figure 5.1: Changing the batch size.



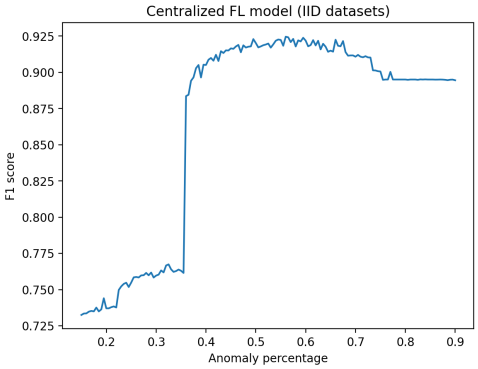
(a) General ML model: Accuracy.



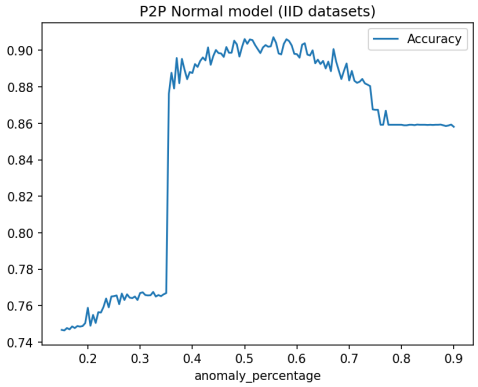
(b) General ML model: F1-score.



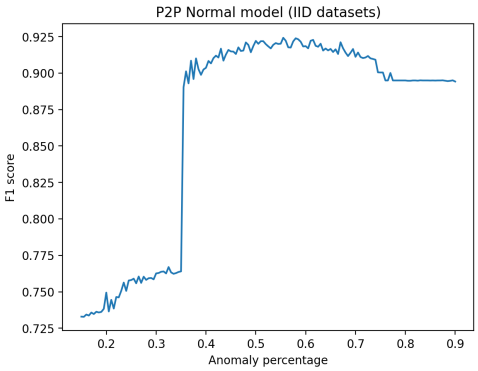
(c) Centralized FL model: Accuracy.



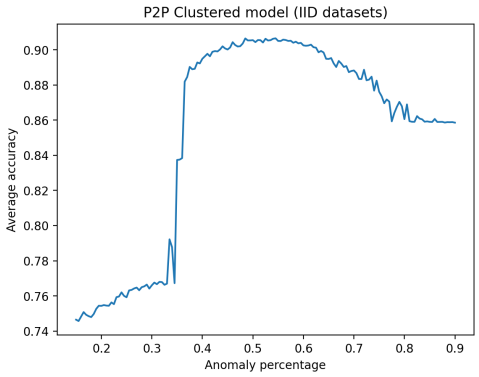
(d) Centralized FL model: F1-score.



(e) Normal P2P FL model: Accuracy.

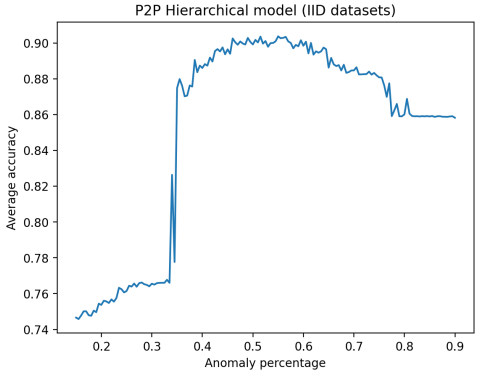


(f) Normal P2P FL model: F1-score.

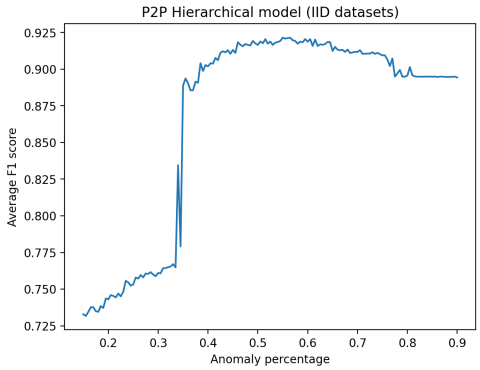


(g) Clustered P2P FL model: Accuracy.

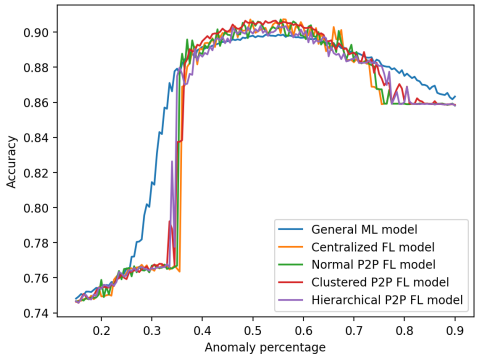
(h) Clustered P2P FL model: F1-score.



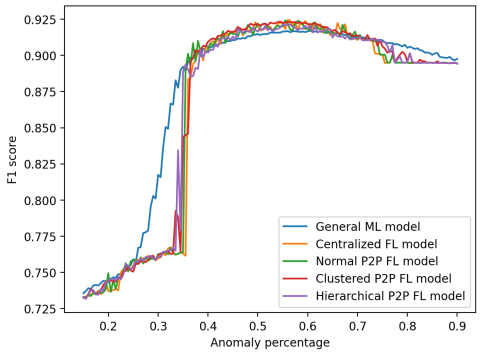
(i) Hierarchical P2P FL model: Accuracy.



(j) Hierarchical P2P FL model: F1-score.



(k) All models: Accuracy.



(l) All model: F1-score.

Figure 5.2: Changing anomaly percentage.

As shown in Figure 5.2, the maximum accuracy and F1 score values are achieved when the training anomaly percentage is around 60% in all methods. When the anomaly percentage is considerably lower, training is not accurate because there are not enough anomalies. When the anomaly percentage is quite high, normal data captured in the training set is not enough. Hence, there can be over-fitting, which leads to decreased performance.

5.1.3 Varying the number of training rounds

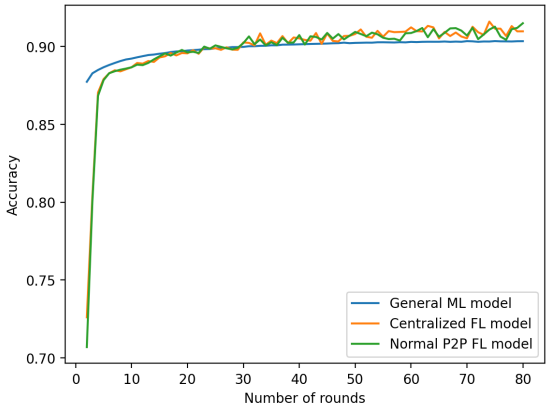
The main parameter considered in this study is the number of training rounds. All three data distributions are considered for the simulations, and all the other fixed parameters are shown in Table 5.3.

For the varying training rounds, the Centralized FL model and the Normal P2P FL model have similar accuracy and F1 score values, which are illustrated in Figure 5.3 and Figure 5.4. This is the expected behavior since averaging method used is the same in both models and the Centralized FL model can be replaced by the Normal P2P FL model. In all three distributions, the accuracy of these two FL models is higher than that of general ML model training after about 45 training rounds.

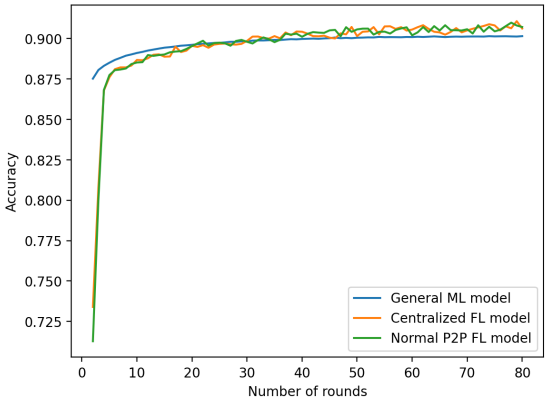
After 80 rounds, the accuracy of the Normal P2P FL model is 91.5% in a random data distribution scenario, which is about 1.1% greater than the General ML model. Similarly, the accuracy is 90.8% in IID and 90.6% in Non-IID, which are about 0.5% and 0.8% more than in General ML model, respectively. Considering the F1-score, in a random data distribution the value is 93% which is about 1% higher than the General ML model. Moreover, the F1-score is

Table 5.3: Fixed parameters for the simulations with varying training rounds

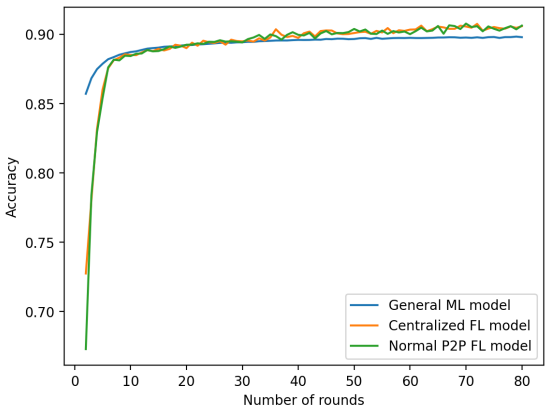
Parameter	**Value**
Number of trainers	100
Number of clusters	5
Number of epochs	10
Batch size	100
Training sample size	150000
Training anomaly percentage	60%
Testing sample size	10000
Testing anomaly percentage	60%
Cluster anomaly proportions	[0.6, 0.5, 0.4, 0.7, 0.6]



(a) Random datasets.

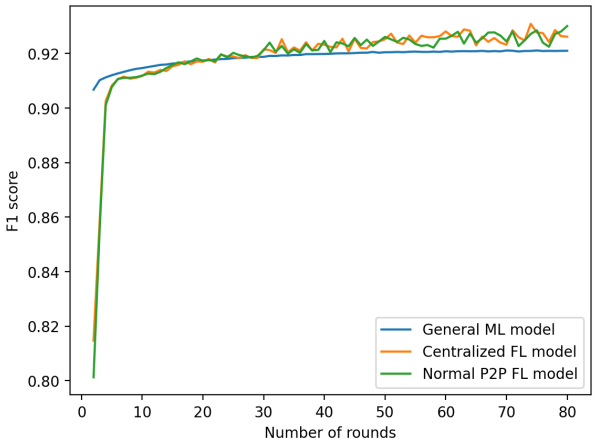


(b) IID datasets.

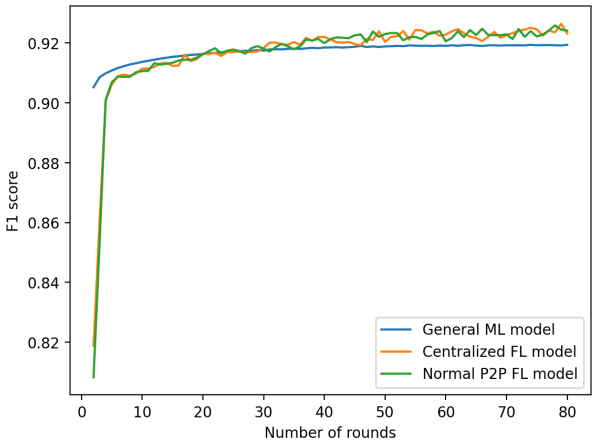


(c) Non-IID datasets.

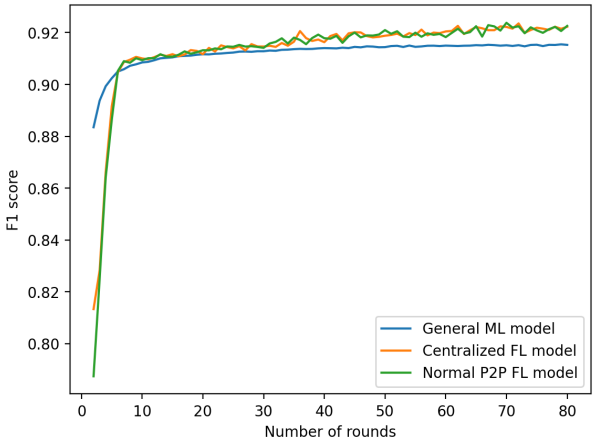
Figure 5.3: Accuracy vs Number of rounds of Normal P2P FL model.



(a) Random datasets.



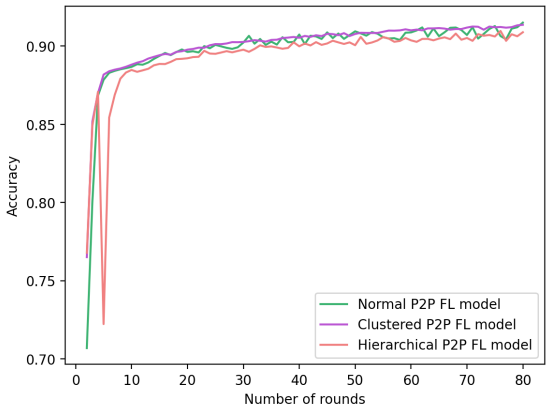
(b) IID datasets.



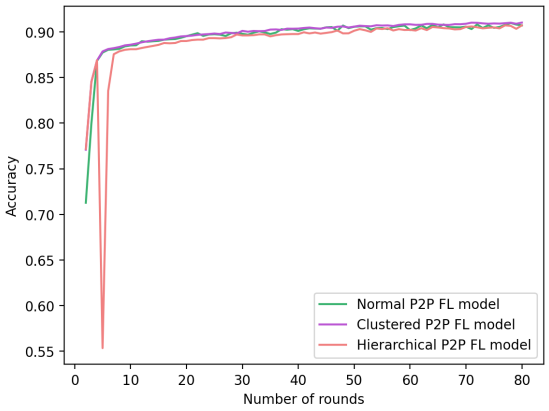
(c) Non-IID datasets/

Figure 5.4: F1-score vs Number of rounds of Normal P2P FL model.

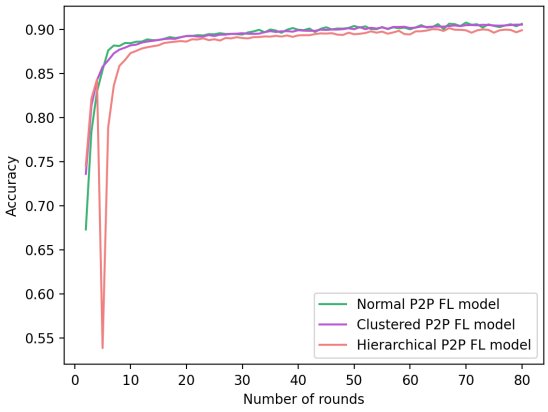
92.4% in IID data distribution, which 0.5% higher than the General model, and in Non-IID data distribution, the gap is about 0.8% with a F1-score of 92.25% in the Normal P2P FL model.



(a) Random datasets.



(b) IID datasets.



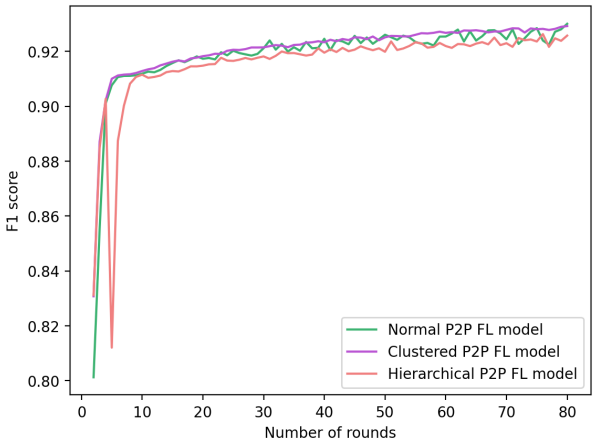
(c) Non-IID datasets.

Figure 5.5: Averaged accuracy vs Number of rounds of SAC based P2P FL methods.

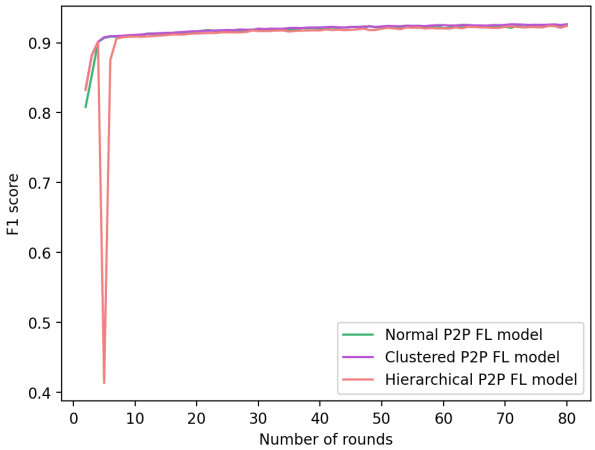
When the three types of SAC-based P2P models are compared, the average accuracy and F1 curves of the Clustered P2P FL model are similar to the Normal P2P model. As illustrated in Figure 5.5, the accuracy of the Clustered P2P FL model after 80 rounds is 91.36% in random data distribution, which is about 0.14% less than the Normal P2P model. In the IID scenario, the accuracy of the Clustered model is about 91% which is 0.2% higher than the normal model, and the value is 90.5% in the non-IID data distribution, which is 0.1% lower than the normal model. Similar behavior can be observed in F1-score as shown in Figure 5.6 where Clustered P2P FL model has a F1-score of 92.9% in random distribution, 92.65% in IID and 92.17% in Non-IID data distributions. These fluctuations are due to the randomness of the training process.

However, the Hierarchical P2P FL model's overall performance is slightly worse than that of the Normal and Clustered P2P FL models. It achieves accuracy values of 90.88%, 90.72%, and 89.9% in random, IID, and non-IID data distributions, respectively. Furthermore, the F1-scores of this model for the three mentioned data distributions, namely, random, IID, and Non-IID, are 92.58%, 92.41%, and 91.73% respectively, and these values are slightly lower than those of the Normal and Clustered P2P FL models. The sub-optimal averaging between the masters could be a reason for this.

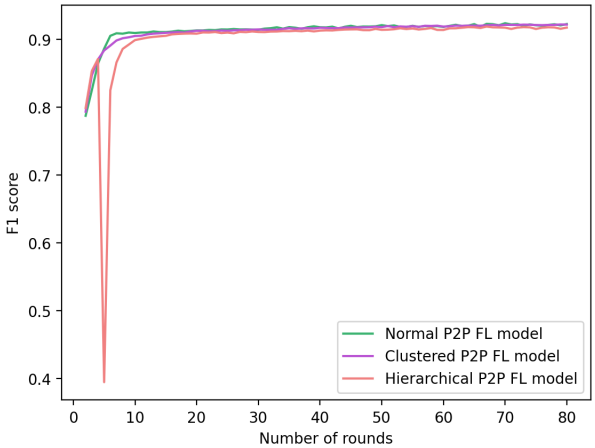
Nevertheless, the clusters have quite similar accuracy and F1 score values in the Hierarchical



(a) Random datasets.

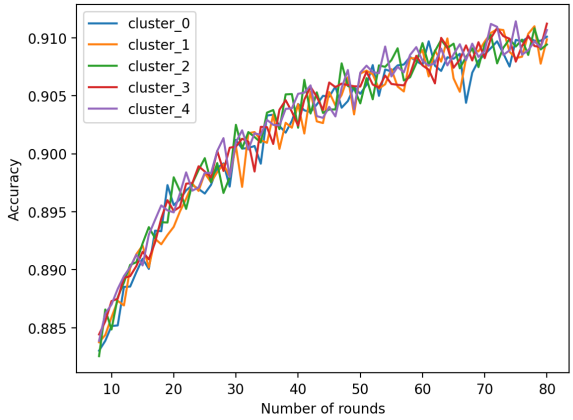


(b) IID datasets.

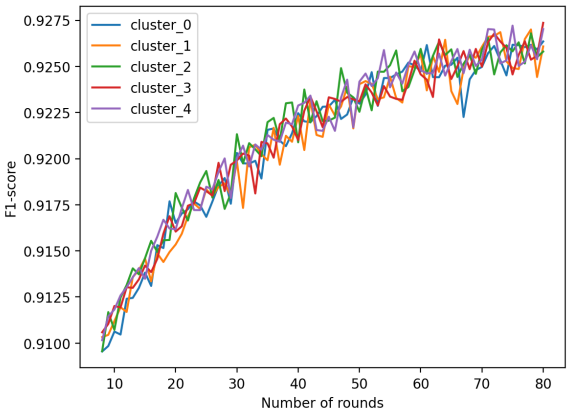


(c) Non-IID datasets.

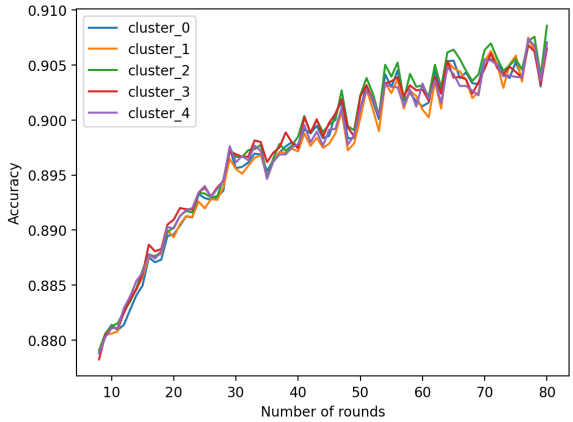
Figure 5.6: Averaged F1-score vs Number of rounds of SAC based P2P FL methods.



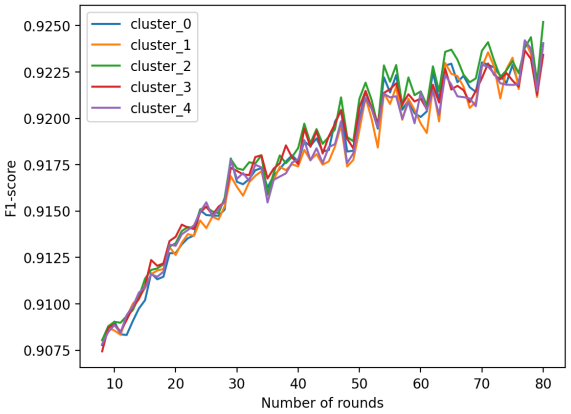
(a) Accuracy in Clustered P2P FL.



(b) F1-score in Clustered P2P FL.

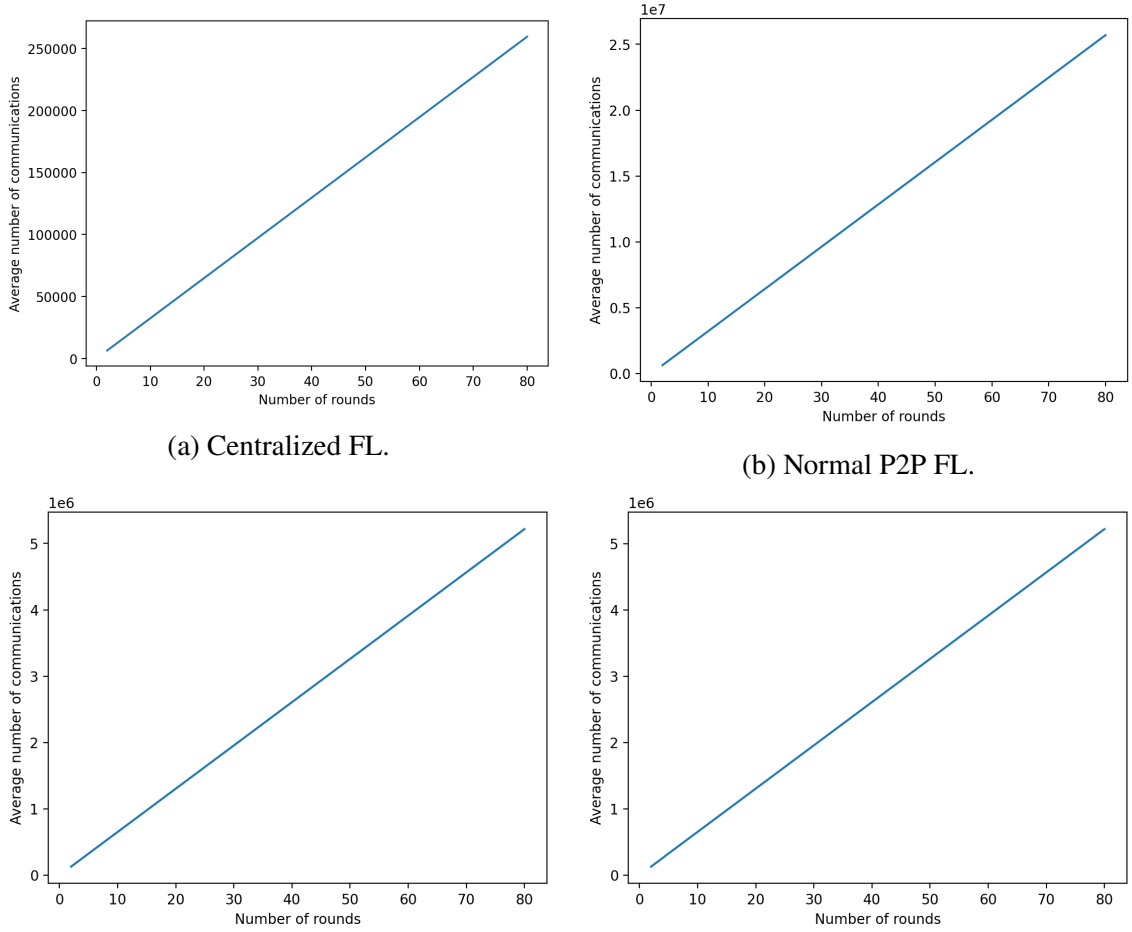


(c) Accuracy in Hierarchical P2P FL.



(d) F1-score in Hierarchical P2P FL.

Figure 5.7: Comparison of Clustered and Hierarchical P2P Fl models with varying Number of rounds (IID datasets).



(a) Centralized FL.

(b) Normal P2P FL.

(c) Clustered P2P FL.

(d) Hierarchical P2P FL.

Figure 5.8: Comparison of Tx communication costs vs Number of rounds.

FL model scenario than in the Clustered FL model due to parameter averaging between the clusters, which can be seen in Figure 5.7.

As illustrated in Figure 5.8, the number of transmissions for training gradually rises in each FL method when the number of rounds increases. However, the total number of Tx communications in a particular round is significantly high in the Normal P2P FL model, which is (number of clients - 1) times more than the Centralized FL model. For an example, at 80 rounds, total Tx communications in the Normal P2P FL model are 25692480, whereas only 259520 communications were in the Centralized FL model, which is 99 times smaller. This is due to the SAC method used for averaging, which provides protection against semi-honest clients. However, Clustered and Hierarchical P2P FL models have about five times lower total communication costs than the Normal P2P FL model due to clustering, where the parameter averaging is performed inside clusters separately.

5.2 Performance of Homomophic P2P FL model

5.2.1 Varying the number of training rounds

The performance of the Homomorphic P2P FL model was observed with a varying number of training rounds for the IID data distribution with the fixed parameters given in Table 5.4. The maximum number of rounds was set at 40 due to simulation time constraints.

Table 5.4: Fixed parameters for the simulations with varying training rounds for Homomorphic P2P FL model

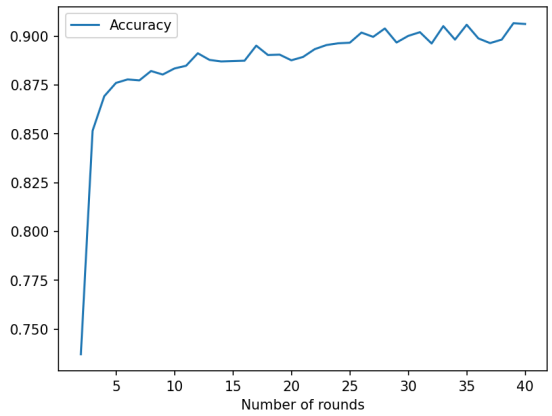
Parameter	**Value**
Number of trainers	100
Number of clusters	5
Number of epochs	10
Batch size	100
Training sample size	150000
Training anomaly percentage	60%
Testing sample size	10000
Testing anomaly percentage	60%

A comparison was done with the Centralized FL method and Normal P2P FL method, as shown in Figure 5.9. The Homomorphic P2P FL model performs the same as the Centralized FL and Normal P2P FL models. However, some degradation of performance can be observed due to higher precision errors in the averaging process due to encryption and decryption.

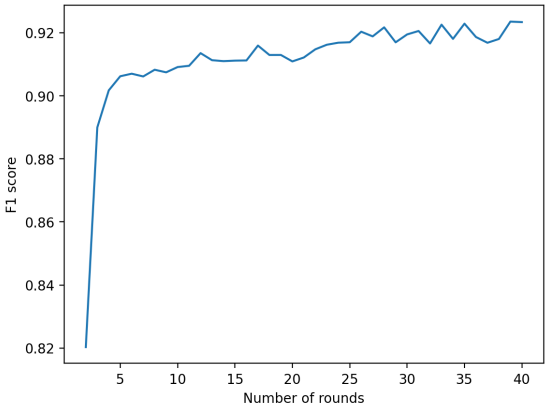
Also, both the Normal P2P FL model and the Homomorphic P2P FL model have the same total Tx communications cost as illustrated in Figure 5.10.

Moreover, the computation cost is significantly higher in Homomorphic P2P FL. However, this performance penalty can be neglected because of the additional security it offers in terms of secure communication and security against semi-honest clients.

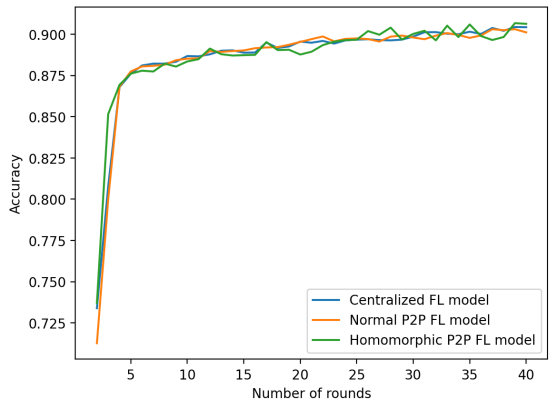
When comparing SAC and HE-based average computation, a semi-honest client receives a partial weight value of an honest client in the SAC-based P2P FL methods. If the actual weight value is negative, then the partial weight values must be negative, and vice versa. Therefore, the semi-honest client can reduce the search range by half, unlike the Homomorphic P2P FL method.



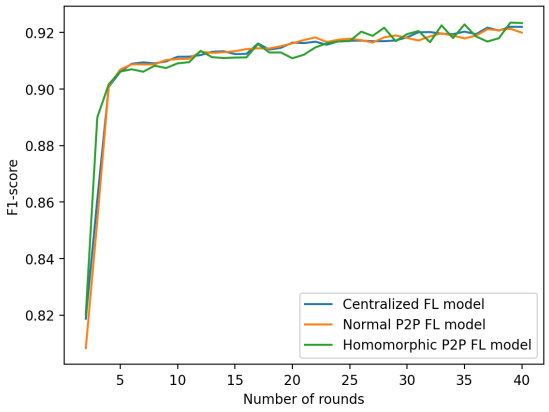
(a) Accuracy vs Number of rounds of the Homomorphic P2P FL model.



(b) F1-score vs Number of rounds of the Homomorphic P2P FL model..

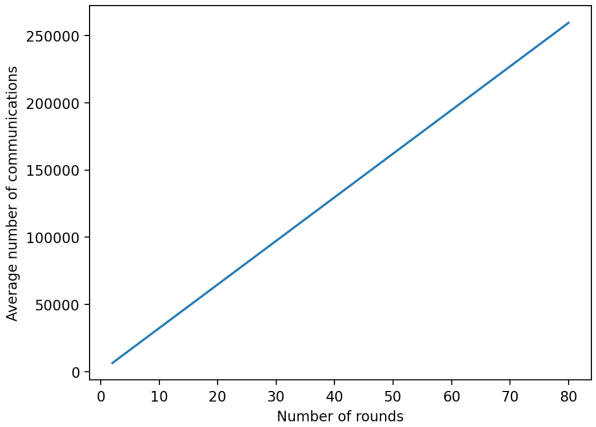


(c) Comparison of Accuracy vs Number of rounds.

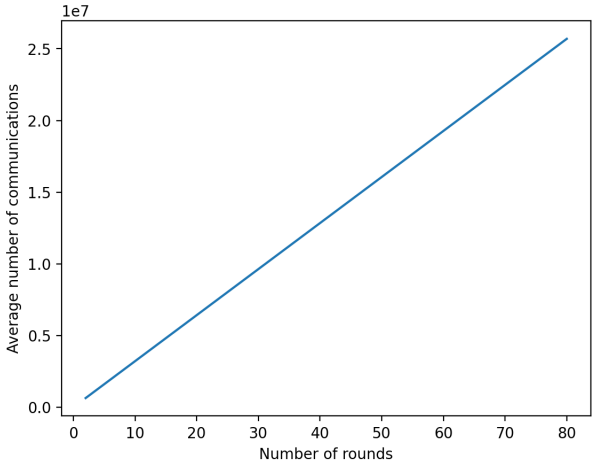


(d) Comparison of F1-score vs Number of rounds.

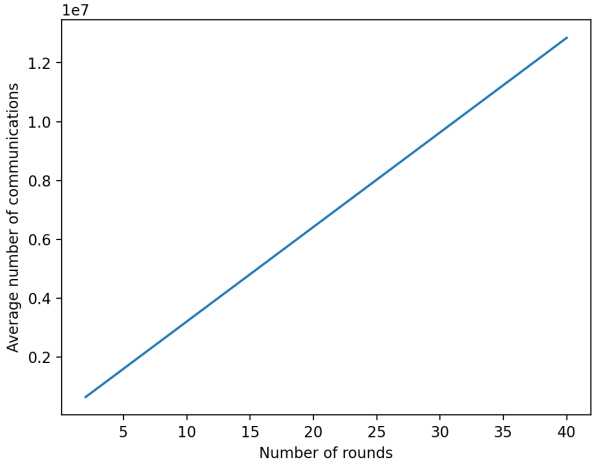
Figure 5.9: Homomorphic P2P FL model performance with varying Number of rounds.



(a) Centralized FL.



(b) Normal P2P FL.



(c) Homomorphic P2P FL.

Figure 5.10: Comparison of Tx communication cost vs Number of rounds of Homomorphic P2P FL model.

6 DISCUSSION

Anomaly detection in O-RAN architecture based on P2P FL is developed in this thesis. Even though network automation is a somewhat mature field, the O-RAN architecture is the first to include built-in intelligence for the network automation in RAN. Since O-RAN is a novel architecture, the automated security algorithms designed for this architecture are still limited.

6.1 Comparison with the state-of-the-art

FL has been utilized in wireless networks, including IoT. As given in [58], most of the IoT applications use resources such as an aggregator server outside their network architecture. However, since AI and ML are embedded parts of the O-RAN architecture with RICs and closed loops, FL solutions can be directly integrated. Some FL-based designs have been implemented for the O-RAN architecture, but they are not related to security automation. Use of FL for resource allocation was presented in [37] and FL DRL-based access control schemes were proposed in [38].

FL-based anomaly detection studies have been carried out on wireless systems, including cyber-physical systems [53], 5G heterogeneous networks [52]. Furthermore, due to the massive distribution, a relatively higher number of architectures are proposed for IoT that use FL for the detection of anomalies including [51], [15]. In addition, for future networks, a FL-based hierarchical anomaly detection mechanism for ZSM architecture is provided in [17]. However, there is limited research about the use of FL for anomaly detection in the O-RAN architecture.

P2P FL is a novel variation of FL that eliminates the use of a central server. Hence it is suitable in massive deployments such as commercial RAN. The privacy-preserving methods, namely, Secure Multi-Party Computation (SMC), Homomorphic Encryption (HE) and Differential Privacy (DP) [41] which are being utilized in centralized FL, can also be used on P2P FL. In this study, two parameter averaging methods were used: secure average computation, which is based on SMC, and homomorphic averaging, which is based on HE.

Since P2P FL is a relatively new method, the research about its usability is still ongoing. Furthermore, the use of P2P FL in O-RAN is still in preliminary phase. This is, to the best of our knowledge, the first FL-based anomaly detection research related to O-RAN architecture.

6.2 Thesis Objectives

The main objective was to implement a P2P based anomaly detection mechanism for the O-RAN architecture. A Normal P2P FL model, a Clustered P2P model, and a Hierarchical clustered P2P FL model with SAC were proposed to detect anomalies in the O-RAN architecture. Furthermore, a more secure approach to P2P FL training called Homomorphic P2P FL model is proposed, where FHE with secret key sharing is used for average computation. All of these detectors can be fully integrated in O-RAN with Near-RT RICs at edge clouds, which host the local training. Also, detectors could be deployed as xApps in the same Near-RT RICs. In addition, proactive controlling of detected anomalies can be designed within the detection xApp or a separate xApp.

The UNSW-NB15 networking dataset was used for the simulation of the mentioned models, and the accuracy and F1-score were measured to evaluate the performance. First, it was observed that the performance would decrease when the batch size of the local training was increased. However, since the batch size has an effect on the training time, a balanced approach has to be taken; hence, 100 was selected as the fixed batch size for the simulations. Furthermore, the

training anomaly percentage was varied to observe its effects. The accuracy and F1-scores were significantly lower in smaller anomaly percentages and also quite high anomlay percentages resulted relatively lower performance. The performance was at its best when the percentage was around 60%, hence it was selected as the fixed anomaly percentage for the remaining simulations.

It can be observed from the results that the Normal P2P FL model has quite similar accuracy to the Centralized P2P FL model, which is the expected result of replacing the centralized model with a distributed P2P model. Also, normal P2P FL performs better after a certain number of training rounds, depending on the other parameters mentioned in Table 5.3. This achieves a maximum accuracy of 91.5% after 80 rounds. Considering SAC-based P2P FL models, the Clustered model's performance is quite close to the Normal P2P model; however, individual cluster performance may be slightly varied. The Hierarchical P2P FL model performs a little worse than the other two P2P models, and this might be improved by optimizing the parameter averaging method of master trainers. Moreover, the individual cluster models of Hierarchical P2P FL model have significant similarities to those of the Clustered P2P FL model.

The Homomorphic P2P FL model has accuracy and F1-scores close to the Normal P2P FL and Centralized FL models, even though there are some fluctuations due to precision errors. Because all of the proposed models achieved accuracy of around 90% and F1 scores of around 92% , the proposed solutions meet the expectations.

All the P2P FL models have higher communication costs than the Centralized FL model due to the additional steps used for parameter averaging in SAC as well as homomorphic encryption. However, it can be justified by the increased security against semi-honest clients.

6.3 Future Research

In future research, it is expected that the proposed models' performance can be improved by changing the structure of the MLP model with a different number of hidden layers and adjusting the parameters such as the learning rate. Furthermore, the different types of averaging methods could be tested to select the best option to achieve the highest performance.

Moreover, the simple neural network can be replaced by advanced models such as CNN or RNN. These can considerably improve the anomaly detection capabilities of the system. Besides those, federated reinforcement learning can be used for model training, which provides the possibility of online training, which has a significant advantage in a highly changing system like O-RAN.

Another research approach that can be considered is the integration of the anomaly detection models with Explainable AI (XAI) [59] which would be helpful for network operators to identify the causes of the relevant anomalies. This would improve preventive actions and achieve higher performance.

A single dataset, UNSW-NB15, is used for all the training and testing of the models. The proposed solution is for 5G and future networks. The wireless networking datasets that can be used for ML training are limited and rarely publicly available. Hence, generating a publicly available dataset that is intended for ML research on 5G and B5G networks would be another research approach. Besides, the purposed solution can be significantly improved by using a dataset that specifically includes RAN data.

As mentioned in the literature, there are several risks associated with AI, and using AI for RAN will result in several threats. Hence, adapting the best practices and remediation for general and specific threats would be taken into consideration in future research. Furthermore, current simulations were performed considering ideal network systems and environments. Therefore, it is expected to simulate different sub-optimal scenarios in training, such as offline trainers,

communication link failures, and semi-honest trainers.

There are several research going on about deploying proposed solution for O-RAN in testing environments and there are some frameworks already available for O-RAN testing like the one mentioned in [60] and [40]. Therefore, implementing the models purposed in this study in one of these O-RAN frameworks, which will eventually lead to deployment in a proof of concept (POC) would be the final expectation.

7 CONCLUSION

In 5G and future networks, the networks are becoming more complex with diverse requirements. The traditional monolithic RAN architectures are too inefficient to cater to these ever-growing requirements and services. There have been several solutions proposed, such as C-RAN and vRAN, to improve the RAN's performance as well as reduce its costs. However, these methods have not solved the major issue of vendor lock-in. The O-RAN alliance proposed a brand-new architecture called O-RAN, which defined standardized open interfaces.

Another major design introduced by the O-RAN architecture was RICs, which allows the deployment of AI/ML processes at the RAN. There are two RICs namely Non-RT RIC and Near-RT RIC in this architecture. Furthermore, it provides cloudification, which allows different kinds of deployments of the architecture based on the network operators requirements. In addition to those, network operators can onboard additional third-party applications to perform non-conventional tasks via micro-services called xApps and rApps, which are hosted by Near-RT RIC and Non-RT RIC, respectively. Security threats to the O-RAN architecture include those resulting from increased threat surfaces such as open interfaces, cloudification, additional functions, and AI and ML. Potential attacks can be O-RAN specific or more general, such as DoS.

Network automation has become a necessary requirement in 5G and B5G. The O-RAN architecture is defined in such a way that the automation is inherited. RICs with near-real-time, non-real-time, and real-time (future research) closed loops and customized xApps and rApps can be utilized to achieve different levels of network automation. Security of the RAN can be significantly improved by automation, and proactive protection against attack can be achieved by the use of AI and closed loops.

A relatively recent component of ML called FL offers additional benefits including greater anonymity. In FL, only the model weights/gradients are transmitted to a central location after models have been trained at end devices using locally accessible data where an aggregator residing in the central site calculates an optimal weight value according to a predefined algorithm. Then these average values are transmitted back to the end devices and used as new weights for the training. There are vulnerabilities in FL that can cause attacks, but several preventive mechanisms have already been proposed.

P2P FL is a rather new variation of FL where the need for a centralized server is removed. The local trainers directly communicate with each other for the averaging of the model weights to achieve the same results as centralized FL. In a complex system like RAN, P2P FL is preferred because it removes the single point of failure and increases anonymity. There are different ways to perform P2P averaging, including differential privacy, Multi-party computation, and homomorphic encryption.

In the proposed solution for anomaly detection, four different types of anomaly detection P2P FL models are designed. In Normal P2P FL model each local trainer communicates with every other for the weight averaging. However, in some practical deployments, this would be sub-optimal, hence a Clustered P2P FL model is proposed where only the trainers in the cluster communicate with each other. The third model, called the Hierarchical P2P FL model, is in the middle of the previous two models, where a master is selected from each cluster, and these masters perform P2P averaging after a certain number of rounds in addition to the within cluster averaging. This increases the possibility of similar models in each cluster. In all three models mentioned above, a secure average is calculated based on multi-party communication, which provides protection against semi-honest clients. Finally, a Homomorphic P2P FL model that uses homomorphic encryption based averaging is designed. This model provides secure communication due to encryption in addition to the security against semi-honest clients.

The simulations were carried out using the Python scripting language and TensorFlow libraries. The UNSW-NB15 dataset was used for model training and testing. The performance of the SAC-based model was observed by simulating varying batch sizes, anomaly percentages of the dataset, and the number of training rounds. Different data distributions, namely random, IID, and non-IID, were considered. The performance of the Homomorphic P2P FL model was tested by varying the number of rounds. Moreover, a general ML model and a centralized FL model were used for comparison purposes. Accuracy and F1-score were the parameters considered for the performance comparison.

According to the findings, the Normal P2P FL model has similar performance to the centralized FL model and achieves higher accuracy than the general FL model after a certain number of training rounds, which depend on other parameters. The Clustered P2P FL model also performed similarly to the Normal P2P FL model, while Hierarchical model's performance is slightly worse. Furthermore, the Homomorphic P2P FL model had quite similar accuracy and F1-score curves as compared with Normal P2P FL model. Overall, all the models achieved accuracies around 90 percent and F1-score values around 92 percent.

Hence, the proposed model can be utilized for detecting anomalies successfully. The closed loop control in O-RAN with RICs and additional xApps providing control actions for detected anomalies can be used to ensure O-RAN is protected from unprecedented attacks. In future networks where O-RAN is part of the architecture, these models would be able to be directly deployed as a part of large AI-based network automation processes.

8 BIBLIOGRAPHY

[1] Arnaz A., Lipman J., Abolhasan M. & Hiltunen M. (2022) Toward integrating intelligence and programmability in open radio access networks: A comprehensive survey. IEEE Access 10, pp. 67747–67770.

[2] Salameh A.I. & El Tarhuni M. (2022) From 5g to 6g—challenges, technologies, and applications. Future Internet 14, pp. 117.

[3] Ahmad I., Shahabuddin S., Kumar T., Okwuibe J., Gurtov A. & Ylianttila M. (2019) Security for 5g and beyond. IEEE Communications Surveys & Tutorials 21, pp. 3682–3722.

[4] Mimran D., Bitton R., Kfir Y., Klevansky E., Brodt O., Lehmann H., Elovici Y. & Shabtai A. (2022) Security of open radio access networks. Computers & Security 122, pp. 102890.

[5] Brik B., Boutiba K. & Ksentini A. (2022) Deep learning for b5g open radio access network: Evolution, survey, case studies, and challenges. IEEE Open Journal of the Communications Society 3, pp. 228–250.

[6] O-RAN ALLIANCE O-ran: Towards an open and smart ran White paper, 2018.

[7] Yuan Y., Yang J., Duan R., Chih-Lin I. & Huang J. (2020) Anomaly detection and root cause analysis enabled by artificial intelligence. In: *2020 IEEE Globecom Workshops (GC Wkshps*, pp. 1–6.

[8] Liu Y., Yuan X., Xiong Z., Kang J., Wang X. & Niyato D. (sep 2020) Federated learning for 6g communications: Challenges, methods, and future directions. China Communications 17, pp. 105–118.

[9] Masur P.H., Reed J.H. & Tripathi N.K. (2022) Artificial intelligence in open-radio access network. IEEE Aerospace and Electronic Systems Magazine 37, pp. 6–15.

[10] O-RAN ALLIANCE O-ran security threat modeling and remediation analysis O-RAN.SFG.Threat-Model-v02.01, 2022.

[11] Moustafa N. & Slay J. (2015) Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6.

[12] Wink T. & Nochta Z. (2021) An approach for peer-to-peer federated learning. In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 150–157.

[13] Kairouz P., McMahan H.B., Avent B., Bellet A., Bennis M., Bhagoji A.N., Bonawitz K., Charles Z., Cormode G., Cummings R. et al. (2021) Advances and open problems in federated learning. Foundations and Trends® in Machine Learning 14, pp. 1–210.

[14] McMahan B., Moore E., Ramage D., Hampson S. & Arcas B.A.y. (20–22 Apr 2017) Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh A. & Zhu J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, PMLR, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282.

[15] Nguyen T.D., Marchal S., Miettinen M., Fereidooni H., Asokan N. & Sadeghi A.R. (2019) DÏot: A federated self-learning anomaly detection system for iot. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 756–767.

[16] Liu Y., Kumar N., Xiong Z., Lim W.Y.B., Kang J. & Niyato D. (2020) Communication-efficient federated learning for anomaly detection in industrial internet of things. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6.

[17] Jayasinghe S., Siriwardhana Y., Porambage P., Liyanage M. & Ylianttila M. (2022) Federated learning based anomaly detection as an enabler for securing network and service management automation in beyond 5g networks. In: *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 345–350.

[18] Polese M., Bonati L., D'Oro S., Basagni S. & Melodia T. (2022) Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges. arXiv preprint arXiv:2202.01032 .

[19] What is openran. `https://www.juniper.net/us/en/research-topics/what-is-open-ran.html`.

[20] O-RAN ALLIANCE O-ran architecture description O-RAN.WG1.O-RAN-Architecture-Description-v06.00, 2022.

[21] WypiLór D., Klinkowski M. & Michalski I. (2022) Open ran—radio access network evolution, benefits and market trends. Applied Sciences 12.

[22] O-RAN ALLIANCE O-ran use cases and deployment scenarios ORAN. WG6.CAD-v02.01, 2020.

[23] Garcia-Saavedra A. & Costa-Pérez X. (2021) O-ran: Disrupting the virtualized ran ecosystem. IEEE Communications Standards Magazine 5, pp. 96–103.

[24] Niknam S., Dhillon H.S. & Reed J.H. (2020) Federated learning for wireless communications: Motivation, opportunities, and challenges. IEEE Communications Magazine 58, pp. 46–51.

[25] Nishio T. & Yonetani R. (2019) Client selection for federated learning with heterogeneous resources in mobile edge. In: *ICC 2019-2019 IEEE international conference on communications (ICC)*, IEEE, pp. 1–7.

[26] Abiodun O.I., Jantan A., Omolara A.E., Dada K.V., Mohamed N.A. & Arshad H. (2018) State-of-the-art in artificial neural network applications: A survey. Heliyon 4, pp. e00938.

[27] O-RAN ALLIANCE Ai/ml workflow description and requirements O-RAN.WG2.AIML-v01.03, 2021.

[28] Bhagoji A.N., Chakraborty S., Mittal P. & Calo S. (2019) Analyzing federated learning through an adversarial lens. In: *International Conference on Machine Learning*, PMLR, pp. 634–643.

[29] Melis L., Song C., De Cristofaro E. & Shmatikov V. (2019) Exploiting unintended feature leakage in collaborative learning. In: *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 691–706.

[30] Xie C., Koyejo S. & Gupta I. (2019) Asynchronous federated optimization. arXiv preprint arXiv:1903.03934 .

[31] Chai Z., Chen Y., Zhao L., Cheng Y. & Rangwala H. (2020) Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. ArXivorg .

[32] Chen Y., Ning Y., Slawski M. & Rangwala H. (2020) Asynchronous online federated learning for edge devices with non-iid data. In: *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 15–24.

[33] Shi Y., Yang K., Jiang T., Zhang J. & Letaief K.B. (2020) Communication-efficient edge ai: Algorithms and systems. IEEE Communications Surveys & Tutorials 22, pp. 2167–2191.

[34] Lin Y., Han S., Mao H., Wang Y. & Dally W.J. (2017) Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887 .

[35] Liu Y., Peng J., Kang J., Iliyasu A.M., Niyato D. & Abd El-Latif A.A. (2020) A secure federated learning framework for 5g networks. IEEE Wireless Communications 27, pp. 24–31.

[36] Cao T.D., Truong-Huu T., Tran H. & Tran K. (2020) A federated learning framework for privacy-preserving and parallel training. arXiv preprint arXiv:2001.09782 .

[37] Singh A.K. & Khoa Nguyen K. (2022) Joint selection of local trainers and resource allocation for federated learning in open ran intelligent controllers. In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1874–1879.

[38] Cao Y., Lien S.Y., Liang Y.C., Chen K.C. & Shen X. (2021) User access control in open radio access networks: A federated deep reinforcement learning approach. IEEE Transactions on Wireless Communications .

[39] Erdol H., Wang X., Li P., Thomas J.D., Piechocki R., Oikonomou G., Inacio R., Ahmad A., Briggs K. & Kapoor S. (2022) Federated meta-learning for traffic steering in o-ran. arXiv preprint arXiv:2209.05874 .

[40] Polese M., Bonati L., D'Oro S., Basagni S. & Melodia T. (2022) Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms. IEEE Transactions on Mobile Computing .

[41] Liu X., Li H., Xu G., Lu R. & He M. (2020) Adaptive privacy-preserving federated learning. Peer-to-Peer Networking and Applications 13, pp. 2356–2366.

[42] Young T., Hazarika D., Poria S. & Cambria E. (2018) Recent trends in deep learning based natural language processing. ieee Computational intelligenCe magazine 13, pp. 55–75.

[43] Aono Y., Hayashi T., Wang L., Moriai S. et al. (2017) Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security 13, pp. 1333–1345.

[44] Geyer R.C., Klein T. & Nabi M. (2017) Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557 .

[45] Turina V., Zhang Z., Esposito F. & Matta I. (2020) Combining split and federated architectures for efficiency and privacy in deep learning. In: *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pp. 562–563.

[46] Roy A.G., Siddiqui S., Pölsterl S., Navab N. & Wachinger C. (2019) Braintorrent: A peer-to-peer environment for decentralized federated learning. arXiv preprint arXiv:1905.06731 .

[47] López-Alt A., Tromer E. & Vaikuntanathan V. (2012) On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 1219–1234.

[48] Damgård I., Pastro V., Smart N. & Zakarias S. (2012) Multiparty computation from somewhat homomorphic encryption. In: *Annual Cryptology Conference*, Springer, pp. 643–662.

[49] Cramer R., Damgård I. & Nielsen J.B. (2001) Multiparty computation from threshold homomorphic encryption. In: *International conference on the theory and applications of cryptographic techniques*, Springer, pp. 280–300.

[50] Maimó L.F., Gómez Á.L.P., Clemente F.J.G., Pérez M.G. & Pérez G.M. (2018) A self-adaptive deep learning-based system for anomaly detection in 5g networks. Ieee Access 6, pp. 7700–7712.

[51] Yadav K., Gupta B., Hsu C.H. & Chui K.T. (2021) Unsupervised federated learning based iot intrusion detection. In: *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pp. 298–301.

[52] Wei Y., Zhou S., Leng S., Maharjan S. & Zhang Y. (2021) Federated learning empowered end-edge-cloud cooperation for 5g hetnet security. IEEE Network 35, pp. 88–94.

[53] Li B., Wu Y., Song J., Lu R., Li T. & Zhao L. (2021) Deepfed: Federated deep learning for intrusion detection in industrial cyber–physical systems. IEEE Transactions on Industrial Informatics 17, pp. 5615–5624.

[54] Karnwal D., Anomaly detection use case. `https://wiki.o-ran-sc.org/display/RICP/Anomaly+Detection+Use+Case`, Accessed: 2022-06-25.

[55] Arthur D. & Vassilvitskii S. (2006) k-means++: The advantages of careful seeding. Technical report, Stanford.

[56] Technical details about puhti. `https://docs.csc.fi/computing/systems-puhti`.

[57] Hamid Y., Balasaraswathi V.R., Journaux L. & Sugumaran M. (2018) Benchmark datasets for network intrusion detection: A review. Int. J. Netw. Secur. 20, pp. 645–654.

[58] Ferrag M.A., Friha O., Maglaras L., Janicke H. & Shu L. (2021) Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis. IEEE Access 9, pp. 138509–138542.

[59] Huong T.T., Bac T.P., Ha K.N., Hoang N.V., Hoang N.X., Hung N.T. & Tran K.P. (2022) Federated learning-based explainable anomaly detection for industrial control systems. IEEE Access 10, pp. 53854–53872.

[60] Bonati L., Polese M., D'Oro S., Basagni S. & Melodia T. (April 2022) OpenRAN Gym: An Open Toolbox for Data Collection and Experimentation with AI in O-RAN. In: *Proc. of IEEE WCNC Workshop on Open RAN Architecture for 5G Evolution and 6G*, Austin, TX, USA.

9 APPENDICES

Appendix 1 Used Libraries in the simulations

```
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import backend as K

tf.get_logger().setLevel(logging.FATAL)

from category_encoders import TargetEncoder

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
np.set_printoptions(precision=3, suppress=True)
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

from IPython.display import display
import matplotlib.pyplot as plt
import seaborn as sns

import csv
import random
import pickle

import nest_asyncio
nest_asyncio.apply()

from typing import List, Tuple
import random
import collections

import time
from datetime import timedelta

import math
import copy

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from multiprocessing import Pool, Manager, freeze_support
from functools import partial
from itertools import repeat
```

Appendix 2 Code for K-means clustering of the clients

```
## Randomize locations of the clients
location_list = []
for user in range(NUM_CLIENTS):
    location_list.append([random.randint(1,500),random.randint(1,500) ])

gnb_loc = pd.DataFrame(location_list, columns=['x','y'])

gnb_loc_ls = gnb_loc.to_numpy()


dist_dic = {}

for index in range(NUM_CLIENTS):
    client_name = "client_" + str(index)
    tmp_ls = []
    for itr in range(NUM_CLIENTS):
        tmp_ls.append(np.linalg.norm(gnb_loc_ls[index,:]-gnb_loc_ls[itr,:]))
    dist_dic[client_name] = tmp_ls



## K-means clustering
kmeans = KMeans(n_clusters=NUM_CLUSTERS)
kmeans.fit(gnb_loc[['x','y']])

cluster_centroids = kmeans.cluster_centers_
cluster_labels = kmeans.labels_

print('\n')
print(f'cluster_centroids : {cluster_centroids}')
print('\n')
print(f'cluster_labels{cluster_labels}')
```

Appendix 3 Code for selection of Masters for Hierarchical P2P FL

```
resource_dic = {}

for index in range(NUM_CLIENTS):
    client_name = "client_" + str(index)
    resource_dic[client_name] =  random.uniform(0., 100.)

resource_lead = {}
resource_lead_mod = {}
cluster_weights_avg = {}
len_avg = 0

for cluster in range(NUM_CLUSTERS):
    cluster_name = "cluster_" + str(cluster)
    len_avg += len(client_clusters[cluster])
    tmp_val = 0
    for index in range(len(client_clusters[cluster])):
        client_name = "client_" + str(client_clusters[cluster][index])
        model_name = "model_" + str(client_clusters[cluster][index])

        if resource_dic[client_name] > tmp_val:
            tmp_val = resource_dic[client_name]
            resource_lead[cluster_name] = client_name
            resource_lead_mod[cluster_name] = model_name

for cluster in range(NUM_CLUSTERS):
    cluster_name = "cluster_" + str(cluster)
    cluster_weights_avg[cluster_name] = len(client_clusters[cluster])/len_avg
```

Appendix 4 Code for Formatting the dataset

```
# Drop id as it is not relevant
data.drop(columns=['id'], axis=1, inplace=True)

# Drop attack_cat as it is not relevant
data.drop(['attack_cat'], axis=1, inplace=True)


X=data.iloc[:,:-1]
y=data.iloc[:,-1]



# Convert to cetegorical columns
categorical_cols = ["proto",
                    "service",
                    "state"]

encoder = TargetEncoder()
encoded = encoder.fit_transform(X[categorical_cols],y)

# Update data with new columns
X.drop(categorical_cols, axis=1, inplace=True)
X = pd.concat([encoded, X], axis=1)



## Normalize the data
scaler = MinMaxScaler()
X_array = scaler.fit_transform(X)

X = pd.DataFrame(X_array, columns=X.columns)
```

Appendix 5 Code for test dataset creation

```
def create_test_data_set(number_of_test_data, Anomaly_data_ratio):
    X_test=[]
    y_test =[]
    Number_of_Anomaly_data_rows = round(number_of_test_data*Anomaly_data_ratio)
    Number_of_Normal_data_rows = number_of_test_data - Number_of_Anomaly_data_rows
    Anomaly_index = random.sample(Total_anomalies,Number_of_Anomaly_data_rows)
    Normal_index = random.sample(Total_normal_flow,Number_of_Normal_data_rows)

    index_list= list(range(0,Full_dataset_len))
    index_list = list(set(index_list)-set(Anomaly_index)-set(Normal_index))

    for i in Anomaly_index:
        m=Data_set[i].tolist()
        X_test.append(m)
        y_test.append(Label[i])
    for i in Normal_index:
        m=Data_set[i].tolist()
        X_test.append(m)
        y_test.append(Label[i])

    return X_test, y_test, index_list
```

Appendix 6 Code for Random dataset creation

```
def create_data_sets_random(num_of_data, Anomaly_data_ratio):
    X_train=[]
    y_train =[]

    Number_of_Anomaly_data_rows = math.floor(num_of_data*Anomaly_data_ratio)
    Number_of_Normal_data_rows = num_of_data - Number_of_Anomaly_data_rows
    Anomaly_index = random.sample(Available_anomaly_indexes,Number_of_Anomaly_data_rows)
    Normal_index = random.sample(Available_normal_indexes,Number_of_Normal_data_rows)

    for i in Anomaly_index:
        m=Data_set[i].tolist()
        X_train.append(m)
        y_train.append(Label[i])
    for i in Normal_index:
        m=Data_set[i].tolist()
        X_train.append(m)
        y_train.append(Label[i])



    idx_list= list(range(0,len(X_train)))
    NUM_CLIENTS_list= list(range(0,NUM_CLIENTS))
    sample_size=len(X_train)//NUM_CLIENTS
    client_train_dataset_random = collections.OrderedDict()


    comb_df = pd.DataFrame(
        {'X': X_train,
         'y': y_train,
        })



    for index in range(NUM_CLIENTS):
        client_name = "client_" + str(index)
        client_data_ind = random.sample(idx_list,sample_size)
        idx_list = list(set(idx_list)-set(client_data_ind))

        user_df = pd.DataFrame(comb_df.loc[client_data_ind,:])
        user_df = user_df.sample(frac = 1)


        user_X = user_df['X'].to_numpy()
        new_dict =np.asarray([user_X[y] for y in range(len(user_X))])
        user_y = user_df['y'].to_numpy()
        new_dict_label = np.asarray([user_y[y] for y in range(len(user_y))])
        client_train_dataset_random[client_name] =collections.OrderedDict((('y', new_dict_label),
                                                                           ('x', new_dict)))


    return client_train_dataset_random
```

Appendix 7 Code for IID dataset creation

```
def create_data_sets_IID(num_of_data, Anomaly_data_ratio):
    X_train=[]
    y_train =[]

    Number_of_Anomaly_data_rows = math.floor(num_of_data*Anomaly_data_ratio)
    Number_of_Normal_data_rows = num_of_data - Number_of_Anomaly_data_rows
    Anomaly_index = random.sample(Available_anomaly_indexes,Number_of_Anomaly_data_rows)
    Normal_index = random.sample(Available_normal_indexes,Number_of_Normal_data_rows)

    for i in Anomaly_index:
        m=Data_set[i].tolist()
        X_train.append(m)
        y_train.append(Label[i])
    for i in Normal_index:
        m=Data_set[i].tolist()
        X_train.append(m)
        y_train.append(Label[i])


    comb_df = pd.DataFrame(
        {'X': X_train,
         'y': y_train,
        })


    normal_idx_list = comb_df.index[comb_df['y']==0].to_list()
    malicious_idx_list = comb_df.index[comb_df['y']==1].to_list()

    NUM_CLIENTS_list= list(range(0,NUM_CLIENTS))
    sample_size=len(X_train)//NUM_CLIENTS
    client_train_dataset_IID = collections.OrderedDict()

    for user in range(NUM_CLIENTS):
        #get the subset of data for client from database
        #get normal dataset iid
        client_name = "client_" + str(user)
        normal_data_idx = random.sample(normal_idx_list,
                                        math.floor(sample_size*(1-anomaly_percentage)))
        normal_idx_list = list(set(normal_idx_list)-set(normal_data_idx))
        normal_df = pd.DataFrame(comb_df.loc[normal_data_idx,:])
        #get malicious dataset iid
        malicious_data_idx = random.sample(malicious_idx_list,
                                           math.floor(sample_size*anomaly_percentage))
        malicious_idx_list = list(set(malicious_idx_list)-set(malicious_data_idx))
        malicious_df = pd.DataFrame(comb_df.loc[malicious_data_idx,:])
        #combine malicious and normal datasets
        user_df = pd.concat([normal_df, malicious_df])
        user_df = user_df.sample(frac = 1)


        user_X = user_df['X'].to_numpy()
        new_dict =np.asarray([user_X[y] for y in range(len(user_X))])
        user_y = user_df['y'].to_numpy()
        new_dict_label = np.asarray([user_y[y] for y in range(len(user_y))])
        client_train_dataset_IID[client_name] =collections.OrderedDict((('y', new_dict_label),
                                                                        ('x', new_dict)))

    return client_train_dataset_IID
```

Appendix 8 Code for Non-IID dataset creation

```
def create_data_sets_non_IID(num_of_data, cluster_anomaly_list):
    sample_size = num_of_data//NUM_CLIENTS

    non_IID_anomaly_rows = 0
    non_IID_normal_rows = 0
    cluster_anomaly_percentage = {}

    for cluster in range(NUM_CLUSTERS):
        cluster_name = 'cluster_' + str(cluster)
        cluster_anomaly_percentage[cluster_name] = cluster_anomaly_list[cluster]
        non_IID_anomaly_rows += cluster_anomaly_list[cluster]*len(client_clusters[cluster])*sample_size
        non_IID_normal_rows += (1 - cluster_anomaly_list[cluster])*len(client_clusters[cluster])*sample_size

    print(f'Total Anomaly data needed :{non_IID_anomaly_rows}')
    print(f'Total Normal data needed :{non_IID_normal_rows}')

    if int(non_IID_anomaly_rows)> len(Available_anomaly_indexes):
        print("Error : Number of required anomalies is larger than available anomalies")
        return

    if int(non_IID_normal_rows)>len(Available_normal_indexes):
        print("Error : Number of required normal flow is larger than available noraml flows")
        return


    comb_df = pd.DataFrame(
    {'X': Data_set.tolist(),
     'y': Label.tolist(),
    })

    client_train_dataset_non_IID = collections.OrderedDict()

    normal_idx_list = Available_normal_indexes.copy()
    malicious_idx_list = Available_anomaly_indexes.copy()

    for cluster in range(NUM_CLUSTERS):
        cluster_name = 'cluster_' + str(cluster)
        cluster_anomaly =  cluster_anomaly_percentage[cluster_name]

        for index in range(len(client_clusters[cluster])):
            client_name = "client_" + str(client_clusters[cluster][index])
            normal_data_idx = random.sample(normal_idx_list,math.floor(sample_size*(1-cluster_anomaly)))
            normal_idx_list = list(set(normal_idx_list)-set(normal_data_idx))
            normal_df = pd.DataFrame(comb_df.loc[normal_data_idx,:])

            #get malicious dataset iid
            malicious_data_idx = random.sample(malicious_idx_list,math.floor(sample_size*cluster_anomaly))
            malicious_idx_list = list(set(malicious_idx_list)-set(malicious_data_idx))
            malicious_df = pd.DataFrame(comb_df.loc[malicious_data_idx,:])
            #combine malicious and normal datasets
            user_df = pd.concat([normal_df, malicious_df])
            user_df = user_df.sample(frac = 1)

            user_X = user_df['X'].to_numpy()
            new_dict =np.asarray([user_X[y] for y in range(len(user_X))])
            user_y = user_df['y'].to_numpy()
            new_dict_label = np.asarray([user_y[y] for y in range(len(user_y))])
            client_train_dataset_non_IID[client_name] =collections.OrderedDict((('y', new_dict_label)
                                                                                , ('x', new_dict)))

    return client_train_dataset_non_IID, cluster_anomaly_percentage
```

Appendix 9 Code for MLP model definition

```
def create_keras_model():
  return tf.keras.models.Sequential([
  tf.keras.layers.InputLayer(input_shape=(42, )),
  tf.keras.layers.Dense(30,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(2),
  tf.keras.layers.Softmax(),
  ])
```

Appendix 10 Code for initial Model training

```
def init_model_train(index, multi_prev_dic, client_train_dataset, NUM_EPOCHS, BATCH_SIZE):

    model_name = "model_"  + str(index)
    client_name = "client_"  + str(index)
    init_model = create_keras_model()

    optim = tf.keras.optimizers.Adam(learning_rate=alpha)
    init_model.compile(optimizer=optim,loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                       metrics=['accuracy'])
    init_model.fit(client_train_dataset[client_name]['x'], client_train_dataset[client_name]['y'],
                   epochs=NUM_EPOCHS, batch_size=BATCH_SIZE,verbose=0)

    temp_ls = []

    for layer in range(len(init_model.layers)-1):
        temp_ls.append([init_model.layers[layer].get_weights()[0], init_model.layers[layer].get_weights()[1]])

    multi_prev_dic[model_name] = np.array(temp_ls, dtype=object)


    tf.keras.backend.clear_session()


init_start_time = time.time()
print(f'Inital training')
with Manager() as manager:
    multi_prev_dic = manager.dict()
    multi_model_dic = manager.dict()
    pool = Pool()
    pool.starmap(init_model_train, zip(range(NUM_CLIENTS), repeat(multi_prev_dic),
                                       repeat(client_train_dataset), repeat(NUM_EPOCHS), repeat(BATCH_SIZE)))
    pool.close()
    pool.join()
    stop_time = time.time()

    prev_dic = multi_prev_dic.copy()

init_elap_time = time.time() - init_start_time
```

Appendix 11 Code for weight averaging in Normal P2P FL model

```
######################## Split the parameters according to random ratios ##########################
sp_time = time.time()

split_dict = {}

for index in range(NUM_CLIENTS):
    model_name = "model_" + str(index)

    split_dict[model_name] = []


    for layer in range(len(dim_list)):
        layer_list = []
        for itr1 in range(2):
            if itr1 == 0:
                split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],avg_array[index,:])

            else:
                split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],avg_array[index,:])

            layer_list.append(split_arr)

        split_dict[model_name].append(layer_list)
    split_dict[model_name] = np.array(split_dict[model_name], dtype=object)

######################################## Calculating partital sums ################################
calc_time = time.time()

calc_dic = {}

for index in range(NUM_CLIENTS):
    model_name = "model_" + str(index)

    calc_dic[model_name] = [[] for layer in range(len(dim_list))]

    comm_count = 0


    for layer in range(len(dim_list)):
        for itr1 in range(2):
            if itr1 == 0:
                sum_calc = np.zeros([dim_list[layer][itr1][0],
                                     dim_list[layer][itr1][1]], dtype= float)
                for user in range(NUM_CLIENTS):
                    temp_name = "model_" + str(user)
                    sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,:,index]


                calc_dic[model_name][layer].append(sum_calc)
                comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (NUM_CLIENTS -1)

            else:
                sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)
                for user in range(NUM_CLIENTS):
                    temp_name = "model_" + str(user)
                    sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,index]


                calc_dic[model_name][layer].append(sum_calc)
                comm_count += dim_list[layer][itr1] * (NUM_CLIENTS -1)


    communication_cost[model_name] = communication_cost[model_name] + comm_count
    calc_dic[model_name] = np.array(calc_dic[model_name], dtype=object)
```

```
######################################## Calculating the parameter averages ############################
fin_time = time.time()

final_dic = {}

for index in range(NUM_CLIENTS):

    sub_comm_count = 0

    model_name = "model_" + str(index)
    final_dic[model_name] = [[] for layer in range(len(dim_list))]


    for layer in range(len(dim_list)):
        for itr1 in range(2):
            if itr1 == 0:
                sum_calc = np.zeros([dim_list[layer][itr1][0], dim_list[layer][itr1][1]], dtype= float)

                for user in range(NUM_CLIENTS):
                    temp_name = "model_" + str(user)
                    sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                avg = sum_calc/NUM_CLIENTS

                final_dic[model_name][layer].append(avg)
                prev_dic[model_name][layer][itr1] = avg
                sub_comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (NUM_CLIENTS -1)


            else:
                sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)

                for user in range(NUM_CLIENTS):
                    temp_name = "model_" + str(user)
                    sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                avg = sum_calc/NUM_CLIENTS

                final_dic[model_name][layer].append(avg)
                prev_dic[model_name][layer][itr1] = avg
                sub_comm_count += dim_list[layer][itr1] * (NUM_CLIENTS -1)


    communication_cost[model_name] = communication_cost[model_name] + sub_comm_count
    final_dic[model_name] = np.array(final_dic[model_name], dtype=object)
```

Appendix 12 Code for weight averaging in Clustered P2P FL model

```
############################### Split the parameters according to random ratios ########################
sp_time = time.time()
split_dict = {}

for cluster in range(NUM_CLUSTERS):

    for index in range(len(client_clusters[cluster])):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])
        split_dict[model_name] = []


        for layer in range(len(dim_list)):
            layer_list = []
            for itr1 in range(2):
                if itr1 == 0:
                    split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],
                                                  avg_dic[cluster][index,:])

                else:
                    split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],
                                                  avg_dic[cluster][index,:])

                layer_list.append(split_arr)

            split_dict[model_name].append(layer_list)
        split_dict[model_name] = np.array(split_dict[model_name], dtype=object)

#################################### Calculating partital sums ###################################
calc_time = time.time()

calc_dic = {}

for cluster in range(NUM_CLUSTERS):

    cluster_len = len(client_clusters[cluster])
    for index in range(cluster_len):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])

        comm_count = 0
        calc_dic[model_name] = [[] for layer in range(len(dim_list))]


        for layer in range(len(dim_list)):
            for itr1 in range(2):
                if itr1 == 0:
                    sum_calc = np.zeros([dim_list[layer][itr1][0],
                                         dim_list[layer][itr1][1]], dtype= float)
                    for user in range(cluster_len):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,:,index]

                    calc_dic[model_name][layer].append(sum_calc)
                    comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (cluster_len -1)

                else:
                    sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)
                    for user in range(cluster_len):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,index]

                    calc_dic[model_name][layer].append(sum_calc)
                    comm_count += dim_list[layer][itr1] * (cluster_len -1)

        communication_cost[model_name] = communication_cost[model_name] + comm_count
        calc_dic[model_name] = np.array(calc_dic[model_name], dtype=object)
```

```
######################################## Calculating the parameter averages ############################
fin_time = time.time()

final_dic = {}

for cluster in range(NUM_CLUSTERS):

    cluster_size = len(client_clusters[cluster])

    for index in range(cluster_size):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])

        final_dic[model_name] = [[] for layer in range(len(dim_list))]
        comm_count = 0



        for layer in range(len(dim_list)):
            for itr1 in range(2):
                if itr1 == 0:
                    sum_calc = np.zeros([dim_list[layer][itr1][0], dim_list[layer][itr1][1]],
                                        dtype= float)

                    for user in range(cluster_size):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                    avg = sum_calc/cluster_size

                    final_dic[model_name][layer].append(avg)
                    prev_dic[model_name][layer][itr1] = avg
                    comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (cluster_size -1)

                else:
                    sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)

                    for user in range(cluster_size):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                    avg = sum_calc/cluster_size

                    final_dic[model_name][layer].append(avg)
                    prev_dic[model_name][layer][itr1] = avg
                    comm_count += dim_list[layer][itr1] * (cluster_size -1)


        communication_cost[model_name] = communication_cost[model_name] + comm_count
        final_dic[model_name] = np.array(final_dic[model_name], dtype=object)
```

Appendix 13 Code for weight averaging in Hierarchical P2P FL model

```
############################Split the parameters according to random ratios ##########################
sp_time = time.time()
split_dict = {}

for cluster in range(NUM_CLUSTERS):

    for index in range(len(client_clusters[cluster])):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])

        split_dict[model_name] = []


        for layer in range(len(dim_list)):
            layer_list = []
            for itr1 in range(2):

                if itr1 == 0:
                    split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],
                                                  avg_dic[cluster][index,:])
                else:
                    split_arr = np.multiply.outer(prev_dic[model_name][layer][itr1],
                                                  avg_dic[cluster][index,:])

                layer_list.append(split_arr)
            split_dict[model_name].append(layer_list)
        split_dict[model_name] = np.array(split_dict[model_name], dtype=object)
##################################### Calculating partital sums ###################################
calc_time = time.time()

calc_dic = {}

for cluster in range(NUM_CLUSTERS):

    cluster_len = len(client_clusters[cluster])
    for index in range(cluster_len):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])

        comm_count = 0
        calc_dic[model_name] = [[] for layer in range(len(dim_list))]


        for layer in range(len(dim_list)):
            for itr1 in range(2):
                if itr1 == 0:
                    sum_calc = np.zeros([dim_list[layer][itr1][0],
                                         dim_list[layer][itr1][1]], dtype= float)
                    for user in range(cluster_len):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,:,index]

                    calc_dic[model_name][layer].append(sum_calc)
                    comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (cluster_len -1)

                else:
                    sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)
                    for user in range(cluster_len):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + split_dict[temp_name][layer][itr1][:,index]

                    calc_dic[model_name][layer].append(sum_calc)
                    comm_count += dim_list[layer][itr1] * (cluster_len -1)

        communication_cost[model_name] = communication_cost[model_name] + comm_count
        calc_dic[model_name] = np.array(calc_dic[model_name], dtype=object)
```

```
######################################## Calculating the parameter averages ############################
fin_time = time.time()

final_dic = {}

for cluster in range(NUM_CLUSTERS):

    cluster_size = len(client_clusters[cluster])

    for index in range(cluster_size):
        model_name = "model_" + str(client_clusters[cluster][index])
        client_name = "client_" + str(client_clusters[cluster][index])

        final_dic[model_name] = [[] for layer in range(len(dim_list))]
        comm_count = 0



        for layer in range(len(dim_list)):
            for itr1 in range(2):
                if itr1 == 0:
                    sum_calc = np.zeros([dim_list[layer][itr1][0],
                                         dim_list[layer][itr1][1]], dtype= float)

                    for user in range(cluster_size):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                    avg = sum_calc/cluster_size

                    final_dic[model_name][layer].append(avg)
                    prev_dic[model_name][layer][itr1] = avg
                    comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (cluster_size -1)

                else:
                    sum_calc = np.zeros([dim_list[layer][itr1]], dtype= float)

                    for user in range(cluster_size):
                        temp_name = "model_" + str(client_clusters[cluster][user])
                        sum_calc = sum_calc + calc_dic[temp_name][layer][itr1]

                    avg = sum_calc/cluster_size

                    final_dic[model_name][layer].append(avg)
                    prev_dic[model_name][layer][itr1] = avg
                    comm_count += dim_list[layer][itr1] * (cluster_size -1)


        communication_cost[model_name] = communication_cost[model_name] + comm_count
        final_dic[model_name] = np.array(final_dic[model_name], dtype=object)
```

```
######################################## Sharing of parameters between master clients ######################
if share_round == SHARE_INTERVAL:

    share_time = time.time()
    share_avg_dic = {}

    for index in range(NUM_CLUSTERS):
        cl_name = "cluster_" + str(index)
        model_name = resource_lead_mod[cl_name]

        share_avg_dic[model_name] =  [[] for layer in range(len(dim_list))]


    for cluster in range(NUM_CLUSTERS):
        cl_name = "cluster_" + str(cluster)
        model_name = resource_lead_mod[cl_name]
        comm_count = 0

        for layer in range(len(dim_list)):
            for itr1 in range(2):
                if itr1 == 0:
                    sum_avg = np.zeros([dim_list[layer][itr1][0],
                                        dim_list[layer][itr1][1]], dtype= float)

                    for user in range(NUM_CLUSTERS):
                        temp_cl_name = "cluster_" + str(user)
                        temp_name = resource_lead_mod[temp_cl_name]
                        sum_avg += cluster_weights_avg[temp_cl_name] * prev_dic[temp_name][layer][itr1]

                    share_avg_dic[model_name][layer].append(sum_avg)
                    comm_count += dim_list[layer][itr1][0] * dim_list[layer][itr1][1] * (NUM_CLUSTERS -1)

                else:
                    sum_avg = np.zeros([dim_list[layer][itr1]], dtype= float)

                    for user in range(NUM_CLUSTERS):
                        temp_cl_name = "cluster_" + str(user)
                        temp_name = resource_lead_mod[temp_cl_name]
                        sum_avg += cluster_weights_avg[temp_cl_name] * prev_dic[temp_name][layer][itr1]

                    share_avg_dic[model_name][layer].append(sum_avg)
                    comm_count += dim_list[layer][itr1] * (NUM_CLUSTERS -1)


        communication_cost[model_name] = communication_cost[model_name] + comm_count
        share_avg_dic[model_name] = np.array(share_avg_dic[model_name], dtype=object)


    for cluster in range(NUM_CLUSTERS):
        cluster_size = len(client_clusters[cluster])
        cl_name = "cluster_" + str(cluster)
        master_model = resource_lead_mod[cl_name]

        for index in range(cluster_size):

            model_name = "model_" + str(client_clusters[cluster][index])
            prev_dic[model_name] = copy.deepcopy(share_avg_dic[master_model])
            final_dic[model_name] = copy.deepcopy(share_avg_dic[master_model])


            if model_name != master_model:
                communication_cost[master_model] = communication_cost[master_model] + total_model_parameters
```

Appendix 14 Code for weight averaging in Homomorphic P2P FL model

```
def encryption_func(index, multi_encrypted_prev_dic, prev_dic, dim_list, public_key):
    model_name = "model_" + str(index)

    multi_val_temp_list = []

    for layer in range(len(dim_list)):
        layer_list = []
        for itr1 in range(2):
            tmp_list = []
            if itr1==0:
                for itr2 in range(dim_list[layer][itr1][0]):
                    w_list = []
                    for itr3 in range(dim_list[layer][itr1][1]):
                        enc_val = public_key.encrypt(int(prev_dic[model_name][layer][itr1][itr2][itr3]*1e15))
                        w_list.append(enc_val)
                    tmp_list.append(w_list)
            else:
                for itr2 in range(dim_list[layer][itr1]):
                    enc_val = public_key.encrypt(int(prev_dic[model_name][layer][itr1][itr2]*1e15))
                    tmp_list.append(enc_val)
            layer_list.append(tmp_list)
        multi_val_temp_list.append(layer_list)

    multi_encrypted_prev_dic[model_name] = multi_val_temp_list.copy()

######################################## Encrpting the parameters ############################
encrypt_time = time.time()
with Manager() as manager:
    multi_encrypted_prev_dic = manager.dict()
    pool = Pool()
    pool.starmap(encryption_func, zip(range(NUM_CLIENTS), repeat(multi_encrypted_prev_dic),
                                      repeat(prev_dic), repeat(dim_list), repeat(public_key)))
    pool.close()
    pool.join()

    encrypted_prev_dic = multi_encrypted_prev_dic.copy()



encrypt_elap = time.time() - encrypt_time
time_calc_dic['Encryption'].append(encrypt_elap)
```

```
######################################## Calculating Encrypted sums ########################################
calc_time = time.time()

encrypted_sum_dic = {}

encrypted_aggr_list= []
for layer in range(len(dim_list)):
    tmp = []
    for itr1 in range(2):
        tmp2 = []
        if itr1==0:
            for i in range(dim_list[layer][itr1][0]):
                tmp2.append([])
        else:
            tmp2 = []
        tmp.append(tmp2)
    encrypted_aggr_list.append(tmp)


for layer in range(len(dim_list)):
    for itr1 in range(2):
        if itr1 == 0:
            for itr2 in range(dim_list[layer][itr1][0]):
                for itr3 in range(dim_list[layer][itr1][1]):
                    sum_calc = encrypted_prev_dic['model_0'][layer][itr1][itr2][itr3]
                    for itr4 in range(1, NUM_CLIENTS):
                        temp_name = "model_" + str(itr4)
                        sum_calc += encrypted_prev_dic[temp_name][layer][itr1][itr2][itr3]

                    encrypted_aggr_list[layer][itr1][itr2].append(sum_calc)



        else:
            for itr2 in range(dim_list[layer][itr1]):
                sum_calc = encrypted_prev_dic['model_0'][layer][itr1][itr2]
                for itr3 in range(1, NUM_CLIENTS):
                    temp_name = "model_" + str(itr3)
                    sum_calc += encrypted_prev_dic[temp_name][layer][itr1][itr2]


                encrypted_aggr_list[layer][itr1].append(sum_calc)


## Copying the values to each model
for user in range(NUM_CLIENTS):
    model_name = "model_" + str(user)
    encrypted_sum_dic[model_name] = copy.deepcopy(encrypted_aggr_list)

    communication_cost[model_name] = communication_cost[model_name] + total_model_parameters * (NUM_CLIENTS - 1)
```

```
def partital_decrypt_func(index, multi_partial_decrypt_dic, encrypted_sum_dic, dim_list, private_key_dic):
    model_name = "model_" + str(index)

    model_priv_key = private_key_dic[model_name]

    temp_partial_decrypt_list = []

    for layer in range(len(dim_list)):
        tmp = []
        for itr1 in range(2):
            tmp2 = []
            if itr1==0:
                for i in range(dim_list[layer][itr1][0]):
                    tmp2.append([])
            else:
                tmp2 = []
            tmp.append(tmp2)
        temp_partial_decrypt_list.append(tmp)



    for layer in range(len(dim_list)):
        for itr1 in range(2):
            if itr1 == 0:
                for itr2 in range(dim_list[layer][itr1][0]):
                    for itr3 in range(dim_list[layer][itr1][1]):
                        temp_partial_decrypt_list[layer][itr1][itr2].append(model_priv_key.partialDecrypt(encrypted_sum_dic[model_name]
[layer][itr1][itr2][itr3]))
            else:
                for itr2 in range(dim_list[layer][itr1]):
                    temp_partial_decrypt_list[layer][itr1].append(model_priv_key.partialDecrypt(encrypted_sum_dic[model_name][layer][itr1]
[itr2]))

    multi_partial_decrypt_dic[model_name] = temp_partial_decrypt_list.copy()

########################################### Partital decryption ##########################################
partital_decrypt_time = time.time()
with Manager() as manager:
    multi_partial_decrypt_dic = manager.dict()
    pool = Pool()
    pool.starmap(partital_decrypt_func, zip(range(NUM_CLIENTS), repeat(multi_partial_decrypt_dic),
                                    repeat(encrypted_sum_dic), repeat(dim_list), repeat(private_key_dic)))
    pool.close()
    pool.join()

    partial_decrypt_dic = multi_partial_decrypt_dic.copy()

partital_decrypt_elap = time.time() - partital_decrypt_time

time_calc_dic['Partital_decrypt'].append(partital_decrypt_elap)
```

```
###################################### Full decryption and average calculation ##############################
decrypt_avg_time = time.time()

final_dic = {}

decrypted_avg_list= []
for layer in range(len(dim_list)):
    tmp = []
    for itr1 in range(2):
        tmp2 = []
        if itr1==0:
            for i in range(dim_list[layer][itr1][0]):
                tmp2.append([])
        else:
            tmp2 = []
        tmp.append(tmp2)
    decrypted_avg_list.append(tmp)


for layer in range(len(dim_list)):
    for itr1 in range(2):
        if itr1 == 0:
            for itr2 in range(dim_list[layer][itr1][0]):
                for itr3 in range(dim_list[layer][itr1][1]):
                    temp_share_list = []
                    for itr4 in range(0, NUM_CLIENTS):
                        temp_name = "model_" + str(itr4)
                        temp_share_list.append(partial_decrypt_dic[temp_name][layer][itr1][itr2][itr3])
                    decrypt_val = combineShares(temp_share_list, public_key.w, public_key.delta,
                      public_key.combineSharesConstant, public_key.nSPlusOne, public_key.n, public_key.ns)
                    decrypted_avg_list[layer][itr1][itr2].append(decrypt_val/(1e15*NUM_CLIENTS))


        else:
            for itr2 in range(dim_list[layer][itr1]):
                temp_share_list = []
                for itr3 in range(0, NUM_CLIENTS):
                    temp_name = "model_" + str(itr3)
                    temp_share_list.append(partial_decrypt_dic[temp_name][layer][itr1][itr2])

                decrypt_val = combineShares(temp_share_list, public_key.w, public_key.delta,
                  public_key.combineSharesConstant, public_key.nSPlusOne, public_key.n, public_key.ns)
                decrypted_avg_list[layer][itr1].append(decrypt_val/(1e15*NUM_CLIENTS))


## copy values
for user in range(NUM_CLIENTS):
    model_name = "model_" + str(user)
    prev_dic[model_name] = copy.deepcopy(decrypted_avg_list)
    final_dic[model_name] = copy.deepcopy(decrypted_avg_list)
    communication_cost[model_name] = communication_cost[model_name] + total_model_parameters * (NUM_CLIENTS - 1)


# Make final_dic a numpy array
for user in range(NUM_CLIENTS):
    model_name = "model_" + str(user)
    for layer in range(len(dim_list)):
        for itr1 in range(2):
            final_dic[model_name][layer][itr1] = np.array(final_dic[model_name][layer][itr1])
    final_dic[model_name] = np.array(final_dic[model_name], dtype=object)
```

Appendix 15 Model prediction functions

```
def model_prediction_targetEnc(index, multi_predict_dic, prev_dic, X_test):
    model_name = "model_"  + str(index)
    new_model = create_keras_model()
    for layer in range(len(new_model.layers)-1):
            new_model.layers[layer].set_weights([prev_dic[model_name][layer][0],
                                                 prev_dic[model_name][layer][1]])

    optim = tf.keras.optimizers.Adam(learning_rate=alpha)
    new_model.compile(optimizer=optim,loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])
    predictions = new_model.predict(X_test, verbose=0)
    multi_predict_dic[model_name] = predictions
    tf.keras.backend.clear_session()


def model_prediction_cluster(cluster, client_clusters, multi_precision_dic, multi_recall_dic,
                             multi_f1_score_dic, multi_test_acc_dic, prev_dic, X_test, y_test):
    model_name = "model_"  + str(client_clusters[cluster][0])
    client_name = "client_" + str(client_clusters[cluster][0])
    cluster_name = "cluster_" + str(cluster)

    new_model = create_keras_model()
    for layer in range(len(new_model.layers)-1):
            new_model.layers[layer].set_weights([prev_dic[model_name][layer][0],
                                                 prev_dic[model_name][layer][1]])

    optim = tf.keras.optimizers.Adam(learning_rate=alpha)
    new_model.compile(optimizer=optim,loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])

    predictions = new_model.predict(X_test, verbose=0)
    y_pred_fl = np.argmax(predictions, axis = 1)

    test_acc_ls = accuracy_score(y_test, y_pred_fl)
    recall_ls = recall_score(y_test, y_pred_fl)
    f1_score_ls = f1_score(y_test, y_pred_fl)
    precision_ls = precision_score(y_test, y_pred_fl)

    multi_precision_dic[cluster_name] = precision_ls
    multi_recall_dic[cluster_name] = recall_ls
    multi_f1_score_dic[cluster_name] = f1_score_ls
    multi_test_acc_dic[cluster_name] = test_acc_ls


    tf.keras.backend.clear_session()
```