FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

DEGREE PROGRAMME IN WIRELESS COMMUNICATIONS ENGINEERING

MASTER'S THESIS

FEDERATED LEARNING-BASED ANOMALY DETECTION AS AN ENABLER FOR SECURING NETWORK AND SERVICE MANAGEMENT AUTOMATION IN BEYOND 5G NETWORKS

Author	Suwani Jayasinghe
Supervisor	Mika Ylianttila
Second Examiner	Pawani Porambage
Technical Advisor	Madhusanka Liyanage Yushan Siriwardhana

March 2022

Jayasinghe Suwani. (2022) federated learning-based anomaly detection as an enabler for securing network and service management automation in beyond 5g networks. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Wireless Communications Engineering. Master's Thesis, 73 p.

ABSTRACT

Zero-touch network architecture (ZSM) is proposed to cater to unprecedented performance requirements, including network automation. 5G and beyond networks include exceptional latency, reliability, and bandwidth requirements. As a result, network automation is a necessity. ZSM architecture combines closed-loop mechanisms and artificial intelligence (AI) to meet the network automation requirement. Even though AI is prevalent, privacy concerns and resource limitations are growing concerns. However, techniques such as federated learning (FL) can be applied to address such issues. The proposed solution is a hierarchical anomaly detection mechanism based on the ZSM architecture, divided into domains by considering technical or business features. The network flow is categorized as an anomaly or not, and abnormal flows are removed from both stages. Detectors and aggregation servers are placed inside the network based on their purpose. The proposed detector is simulated with the UNSW-NB15 Dataset. The simulation results show accuracy improvement after the 2nd stage, and the detection accuracy varies with training data composition.

Keywords: 5G, beyond 5G, Network automation, Security, Federated learning, ZSM

TABLE OF CONTENTS

FOREWORD

This master's thesis, written at the Centre for Wireless Communications at the University of Oulu for the Master Program in Wireless Communications Engineering, introduces a federated learning-based anomaly detector for 5G and beyond 5G networks. It was written as part of the Master's Program in Wireless Communications Engineering. The ZSM architecture, federated learning, and anomaly detection are all discussed in detail in this thesis.

I would like to express my gratitude to Prof. Mika Ylianttila for his assistance and guidance throughout the course of my master's thesis. In addition, I would like to express my heartfelt gratitude to Dr. Pawani Porambage and Dr. Madushanka Liyanage for their tremendous assistance and guidance throughout the thesis process. I'd also like to express my gratitude to Yushan Siriwardhana for guiding me and providing insightful suggestions. I'd also like to thank my husband, Dhananjaya, as well as my friends and family for their support.

Oulu, March, 2022

Suwani Jayasinghe

LIST OF ABBREVIATIONS AND SYMBOLS

5G	Fifth generation
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
ANN	Artificial Neural Network
ASD	Anomaly Symptom detection
BRAC	Base Closure and Realignment
B5G	Beyond 5G
DAD	Deep Anomaly Detection
DoS	Denial of Service
DDoS	Distributed Denial of Service
DP	Differential privacy
E2E	End to end
ETSI	European Telecommunications Standards Institute
FL	Federated Learning
FTP	File Transfer Protocol
GADC	Global Anomaly Detection Centre
GAN	Generative adversarial networks
GRU	Gated recurrent unit
IDS	Intrusion Detection Service
IoT	Internet of Things
JWT	JSON Web Token
KPI	Key Performance Indicators
LADC	Local Anomaly Detection Centre
LSTM	Long Short-Term Memory
MAC	Message Authentication Algorithms
MITM	Man in the Middle
ML	Machine Learning
NAD	Network anomaly detection
NFV	Network Function Virtualization
OS	Operation systems
RADC	Regional Anomaly Detection Centre
RAN	Radio Access Network
SDN	Software Defined Networking
SGD	Stochastic gradient descent
SLA	Service level agreements
SLS	Service level specifications
SLO	Service level Objectives
TLS	Transport Layer Security
TPM	Trusted Platform Module
VTPM	Virtual Trusted Platform Module
VNF	Virtual network functions
VR	Virtual Reality
UNSW	The University of New South Wales
URL	Uniform Resource Locator
ZSM	Zero-touch network and Service Management

w	Weights
x_i	Input features/values to the model
y_i	Label of the input features
K	Number of clients
p_k	Number of sample

1 INTRODUCTION

The fifth-generation (5G) and beyond 5G networks are projected to meet previously unheard-of network requirements. Its primary goal is to enable game-changing applications that require high-quality, low-latency visual and auditory telepresence, as well as massive connectivity and capacity. Because 5G networks are ultra-reliable, always-on, and have low latency, they have the potential to enable new services that will change industries. Medical operations, remote control of infrastructure, and automobiles are a few examples. 5G mobile technology has the potential to usher in new immersive experiences such as virtual reality and augmented reality (VR and AR) due to more consistent and faster data rates, lower cost, and lower latency. It is expected that 6G will raise the bar even higher, with speeds predicted to be 100 times quicker than 5G and increased bandwidth to keep people more connected than they have ever been. Both network generations are expected to be sustainable, stable, scalable, efficient, dependable, and agile. However, as technology advances, network architecture becomes more complex.

New network architectures include elements such as network function virtualization (NFV), network slicing, and multi-access edge computing to meet expectations (MEC). 5G uses a much smarter architecture, with radio access networks (RANs) no longer limited by base station proximity or the complexity of the infrastructure required [1].5G enables a flexible, disaggregated, and virtualized RAN, with new interfaces producing new data entry points. Modern edge computing (MEC) is a cloud computing trend in which applications are moved from data centers to the edge of the network, bringing them closer to the consumers and the devices they use. This creates a content delivery shortcut between the user and the host, eliminating the need for the previously lengthy network path. NFV enables virtualization in the 5G infrastructure. Network slicing technology enables running multiple virtual networks concurrently on a single physical infrastructure. NFV can help with the 5G challenges by enabling the provisioning of storage, virtualized computing, and resources tailored to the specific needs of customer segments and applications [1]. Similarly, 6G is expected to include a plethora of new elements. Researchers in both types of networks are interested in artificial intelligence (AI).

However, as technology advances, it is becoming increasingly difficult to operate the network manually. As a result, network automation has become a requirement. It has become clear over recent years that using AI technology to enable fully automated network operations and maintenance is essential for lowering operating expenses while improving network key performance indicators (KPIs) for 5G and beyond 5G (B5G) and 6G networks, as well as for lowering costs of capital. The use of AI technology has already been adopted by many operators to automate some repetitive operations tasks and reduce dependence on employee expertise, such as implementation simplification, troubleshooting, and KPI optimization. There have been numerous research efforts from both industry and academia in order to develop network automation solutions that are powered by AI.

The European Telecommunications Standards Institute (ETSI)-proposed Zero-touch network and Service Management (ZSM) architecture can be viewed as an effort by both industries and universities to create a network capable of self-optimization, self-configuration, self-healing, and self-monitoring without human involvement. It is optimized for AI algorithms and data-driven machine learning (ML) and utilizes closed-loop automation. The ZSM architecture is designed to protect the privacy and security of data transmitted, processed, or stored within the ZSM network and includes data analytics and related services, allowing ML techniques to be widely used for a variety of purposes. Federated Learning (FL) is a relatively new branch of ML that is well known for protecting user data privacy. As a result, FL-based

solutions can also be used in the ZSM architecture. In an automated network, FL and closed-loop operations can be used to successfully secure a network.

1.1 Background and Motivation

Beyond 5G or 6G mobile networks, intelligent security and automated security management are two critical features to consider [1], and the ZSM network architecture is built to cater to both these features. The ZSM framework was proposed by ETSI to fulfill the automation requirements of network management [2].

In the ZSM architecture, the network is divided into domains based on operational, business, and technological needs. A management domain consolidates administrative duties within a ZSM deployment. It manages resources and resource-facing services for each ZSM network division. Each management domain has service endpoints with various capabilities. Dividing the network into RAN, core, transport, and application domains is a common way to create domains. The AI and ML algorithms are two primary components that play a vital role in supporting the functionalities of closed-loop operations proposed in the ZSM architecture [3]. The closed-loop functions can be used for attack detection, cyber threat intelligence for attack mitigation or prevention, security analytics, security policy updates, and security orchestration [4]. Even though ML is used frequently in many applications, problems must be addressed, including user data privacy. In such instances, FL can be used effectively. It is a relatively novel form of ML algorithm that allows decentralized processing with higher communication efficiency and higher privacy [5].

1.2 Research Problem

The ZSM architecture is intended to include data analytics as well as related data services. A few related services found in the architecture design are the AI model management service, the deployed AI model assessment service, the knowledge base service, the analytics services in management domains, and the end-to-end (E2E) management domain [2]. As a result, it is simple to create ML-based solutions for a variety of purposes. Because the ZSM architecture is divided into domains, detectors can be deployed to monitor the network in each domain.

If ML models are used in the network, the models must be generated and kept up-to-date. To ensure that a model can be created from the data, it is necessary to store all the data in a data center. The process of successfully deploying and operating an ML model can be complicated due to a variety of challenges, such as ensuring the privacy of customers' data. As a result, FL is proposed, which trains the model in end devices using locally available data.

However, deploying a sufficient number of detectors in any network inspired by the ZSM architecture is critical for security monitoring and data collection tasks. To successfully detect anomalies, such detectors must be empowered by ML algorithms. To properly protect networks from all types of possible attacks, each detector must have a local database or similar structure that stores relevant network flow data. As a result, resources like processing power and storage space may be limited. As a result, a hierarchical detection mechanism based on FL is proposed.

1.3 Selected Scope

In its architecture, the ZSM architecture facilitates AI-based solutions. As a result, an infinite number of applications for self-monitoring, self-configuration, self-optimization, and self-healing are possible. Security is often prioritized over other requirements in most network

architectures. It is possible to create a security application that is AI-powered and analyses user data. However, it is possible that users will be concerned about data privacy when using the network. Therefore, an anomaly detector based on FL to protect the network is proposed.

The Fl algorithm is made up of several components, including aggregation criteria, a ML model, and a data set. By changing each element, different cases can be generated, and a suitable setup can be found based on the application. The proposed application was not followed in all possible scenarios. Averaging was the only aggregation criterion used. To produce the final model parameters, all of the parameters sent by the client devices are averaged. It was not necessary to consider the number of samples each client had because the data set was distributed independently and identically among the number of clients. Neural networks were used in both stages 1 and 2. The model in stage 1 has two hidden layers, for a total of four layers. Likewise, the stage 2 model had four hidden layers, for a total of six layers. It was not attempted to change the number of layers of the model as the developed system generated sufficient accuracy. The detection algorithm was developed to detect the nine types of attacks available in the used data set. For the simulation, the number of rounds was limited to 100.

1.4 Methodology of the work

As the proposed anomaly detector is based on an FL model, the methodology of this thesis is primarily comprised of steps required to generate an ML or FL model. It consists of three steps: data preparation, model training, and model testing. The data set in use contains a variety of data types, including network types and protocol names. Only numerical values, however, can be fed into the neural network. As a result, all of these values must first be assigned a numerical value, which is then normalized to fall within a predetermined range of values. There may also be extraneous values, such as anomaly types. This data column was removed because it is sufficient to determine whether or not an anomaly exists. After the data set was prepared, it was divided into testing and training data. Various training data sets were made by changing the amount of anomaly data in each.

Two models were developed for each training data set during the training step. The first step of the cycle in the FL-based training model is to train the model in end devices, which are then communicated back to the central server. The model parameters are aggregated at the central server before being sent back to the end devices for training. After training, it will be sent back to aggregation. This is considered one full cycle in this thesis. The models were trained in this manner for 100 rounds. The model that is generated for each round in each case is saved, and this model is then used to test the model's performance.

The last phase of the thesis is testing the generated model. First, generated model parameters were loaded onto two models. Then the accuracies after stages 1 and 2 were recorded after testing with the 10,000 flows of data allocated beforehand. The results observed were recorded for each case.

1.5 Contribution

This thesis's main contribution is an FL-based anomaly detector that can be successfully deployed in the ZSM architecture. Network automation is still in its early stages, and the ZSM architecture is relatively new. As a result, there aren't many applications related to the aforementioned architecture. The proposed solution is tightly integrated into the new architecture and demonstrates how applications can benefit from closed-loop and FL. The majority of the work presented in the literature for FL-based applications is related to internet

of things (IoT) applications. Only a few have been introduced for other wireless communication applications. This thesis shows how FL can be used in network security applications.

The performance of the model is evaluated using two parameters: overall system accuracy and accuracy improvement from stage 1 to stage 2. The multistage detector identifies anomalies in the used data set with up to 93.6% accuracy. It also shows how the use of two stages improved the accuracy. As a result, the proposed system can successfully defend the network against attacks. The results also show the effect of the composition of the training and testing data. The results emphasize the importance of having enough data flows to cover all possible scenarios.

1.6 Organization of the Thesis

This thesis consists of six chapters, and it is organized in the following manner:

Chapter 2 contains details of the existing literature related to the ZSM architecture, FL, and anomaly detection. This section consists of three sections dedicated to ZSM architecture, FL, and anomaly detection using FL. The last section describes the current work related to detecting anomalies using FL.

Chapter 3 describes the proposed anomaly detector. The detection and training phases are explained in this section, and how they are positioned in the ZSM architecture. This chapter also describes how the proposed detector reacts if any anomaly is undetected, and if it disturbs network operations.

Chapter 4 describes how the simulation is carried out to generate the results for the defined set of test cases. It starts with the data set description, and later the simulation setup is described under training and testing topics.

Chapter 5 contains the results, and it demonstrates how the network's accuracy varies with the training data composition. Three graphs included also show how the accuracy varies along with the number of training rounds.

Chapter 6 critically analyses the thesis work. It also contains a comparison to the thesis results of similar research work. Apart from future directions, this chapter also includes how much success was achieved in reaching the objective of the thesis.

Chapter 7 contains the thesis summary, including research objectives, a summary of the related work, results, and analysis.

2 LITERATURE REVIEW

End-to-end network automation is becoming increasingly important as network complexity rises to meet unprecedented network demands. The domain-based architecture is a key feature of the ZSM architecture. Domains in ZSM deployment divide administrative responsibilities and create a separation of concerns based on functional, operational, and governance constraints [2]. For management purposes, security functions can be grouped together, and AI-enabled components can also be used for security functions. Because of the critical role it plays in the network, researchers are paying close attention to security automation. Combining AI/ML techniques with the closed-loop process proposed in the ZSM architecture can make security automation a reality, which also includes data analytic services for many other functions.

2.1 ZSM Architecture

The ZSM architecture is proposed with the intention of making a fully automated network a reality, and as mentioned in the reference architecture [2], it is more flexible and moves away from tightly coupled management systems. Figure 1 shows the proposed architecture and key building blocks described above, as shown in [2]. The key architectural building blocks are management functions, management services, the integration fabric, management domains, the end-to-end (E2E) service management domain, and data services. Each of these components is described below.

- Management services

This is also the most fundamental building block in the ZSM architecture. Service consumers can use standardized management endpoints offered by management services. Management services may communicate with resources directly using the management interface or by using other management services. In addition, all of these services provide a set of communication and invocation capabilities, that allows for a high degree of automation and continuity to be achieved within management domains. It is also possible to combine a bunch of management services to form a new service management service.

- Management functions

Entities that produce or consume management services are known as "management functions." A management function is categorized as a service producer or a service consumer if it produces or consumes services, respectively, and a function can play both roles simultaneously.

- Integration fabric

The integration fabric, as the name suggests, manages the communication between management domains. The set of functions includes registration, discovery, and invocation of management services. As mentioned in [7], the integration fabric offers services such as service registration, service discovery, communication services, and service invocation routing service.

- Data service

The data service enables the sharing of data among the authorized users across management services, eliminating the requirement for the management functions to handle their own data. The services offered by this building block can be categorized as data storage management service, data processing service, and data persistence service.

- Management domains

A management domain federates administrative responsibilities within a given ZSM deployment. It manages the resources and the resource-facing services based on the division in each ZSM network. Physical and virtual network functions (VNF) and cloud-based services are

some of the resources that each domain can manage. Each management domain provides service endpoints that offer one or more service capabilities. A management domain can consume management services offered by another management domain.

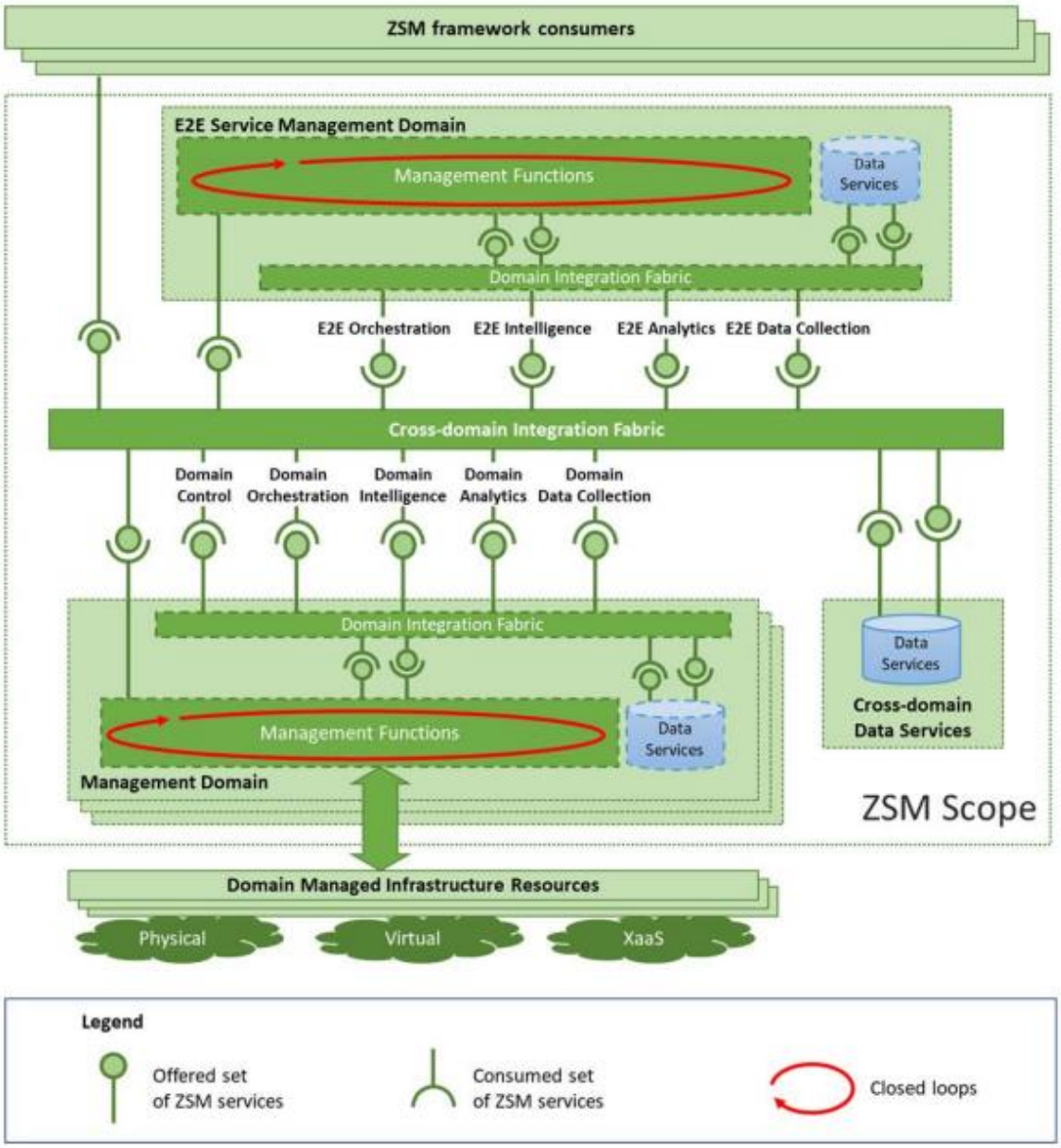

Figure 1 ZSM reference architecture [2]

As mentioned in [6], the primary service categories of a management domain include domain data collections, domain control, domain orchestration, domain intelligence, and domain analytics. Each of these components is further described below as appearing in [2], [6], and [7].

- Domain data collections

This service monitors the network and provides data such as performance matrices and fault reports to the closed-loop operations. As shown in Figure 1, domain data collection interacts with domain control services, domain intelligence services, domain analytics, and domain orchestration services to generate changes required in the management domain. A service in

one management domain may indirectly consume data services in other domains if a service that belongs to the second management domain consumes its data services. The services offered by the data collection services are processing incoming events, inputs related to faults, data objects, security, and performance of the managed entities and managed services consumed. It also generates notifications and data related to the said services. Domain data collections also have the capability to create performance indicators by combining information. The provided services can be further categorized as event notification services, performance measurement services, and performance measurement streaming services.

Event notification services create an notification called asynchronous event notification to report status changes and problems with the managed services promptly. The subscribed services will communicate with the other services via the integration fabric. The list of notifications generated includes fault event service, security event service, and performance management service. The fault events service generates a notification when there is a fault in the system or an abnormal system state in the infrastructure or services related to it. As the name implies, the security events service provides notifications about security-related events. The events service sends notifications about events that it monitors. For example, if the predefined threshold is crossed, the performance events service will notify it. The performance measurements streaming service is responsible for generating performance measurements created by the network services and infrastructure resources. This is generated in a streaming fashion. It also configures control information such as subscription and data to be measured and provides the collected performance data. This service also includes information about the available measurements.

The performance measurements collection service is another service offered by the domain data collection service. It also offers performance measurements from network resources and infrastructure in batches. It will collect the data from the resources during a configured time duration and provide it to authorized consumers. Authorized consumers may include producers of other management services. This service also allows configuring control information such as data to be reported, subscriptions, and how to provide and collect performance data.

The log collection service collects logs, including data about the system's running, security, and operations events. These logs can be used for the health, security, and performance monitoring of software entities. Data optimization services preprocess the data to provide clean, structured, and aggregated data.

- Domain analytics

These services use the information collected by the data collection services and any available data to generate domain-specific insights and predictions. The provided services can be further categorized as data optimization services, analytics services, and domain condition detection services.

Analytics services derive insight from the collected data. Apart from the generic analytics services such as managing subscriptions and configuring analytics, it also offers capabilities such as anomaly detection services, deployed AI model evaluation services, reactive incident analysis services, and proactive incident analysis services. Anomalies in the resources or services are detected by the anomaly detection service using the collected fault, performance, usage information, etc. Anomalies detected may include security violations, short-term fault conditions, and service capacity degradation, even though KPIs indicate that the services are working properly, indicating that the resources need to be scaled or healed. In the event of detecting anomalies, domain intelligence will be triggered to make appropriate

decisions regarding the service or to analyze it further. AI is integrated into the ZSM architecture, and the same needs to be monitored to maintain the required level of performance. This is conducted by the deployed AI model performance evaluation services. It will monitor AI-related services to detect performance degradation, sudden performance drops, irregular performance, etc. Proactive incident analysis services offer services such as collection and classification of data, insights, and inputs to take proactive measures. It also analyses the network structure and performance to produce insights. The reactive incident analysis service collects and classifies data for performing corrective actions. In addition, the proactive network optimization service gathers information from other services in order to produce the metrics required for network performance optimization and optimization.

The domain condition detection service is another primary service offered by the data analytics services, and it monitors domain-specific conditions. Breaking down the set of conditions that need to be monitored is also done by the domain condition detection service. It will also generate a notification of the status of its changes. It is possible to define these cases as case-by-case.

Data optimization services in data analytics services offer the capability of preprocessing the data to be consumed by the data analytics applications directly. It is also possible to provide alternative views of the data and remove irrelevant or redundant data. The services offered by the data optimization services can be further divided into redundancy removal services, irrelevancy removal services, and data aggregation services. The data aggregation service aggregates the data according to different dimensions such as locations, time periods, and topology.

- Domain intelligence

The main function of the domain intelligence service is to make decisions to drive the closed-loop automation process. The decision-making is automated to various levels. It can further be categorized as decision-making, decision support, and action planning. Decision-making with the aid of technologies such as AI is considered a decision support service. Decision-making services generate decisions based on data made available to them. Defining an action plan or orchestration that the ZSM services will execute is also known as action planning. The services offered by domain intelligence services include AI model management services, deployed AI model assessment services, knowledge base services, and AI training data management services.

As the name suggests, AI model management services manage AI models used by the network. It is possible to update the model periodically or at any time as desired by the administrator. It is possible to degrade the performance of the deployed model for reasons such as a change in reality. In such instances, AI model assessment services determine the actions to be taken. A deployed AI model may undergo measures such as retraining, reconfiguring, upgrading, replacement, pause, and termination. The AI training data management service manages the data required for training the model and retraining the model. Training data may contain labelled data as well. Knowledge base services create and maintain a machine-readable knowledge base. It has known problems and descriptions of the underlying course. The health issue-reporting service generates reports that need to be sent to higher-order entities. It contains reports about health issues, faults, and security issues.

- Domain orchestration

The services offered by the domain orchestration services allow automation of workflows and processes inside a management domain. It maintains the inventory of network services and

other virtualized resources of the management domain. It also maintains an updated view of the related topology. Discovery, topology, and inventory management services are used for this. It should be noted that, as per the proposed architecture, inventory of the hardware resources and hardware installation or removal is not managed by the orchestration services. The consumers of the orchestration services include services from the other domains, consumers that provide E2E service orchestration services, and ZSM framework consumers. As depicted in the closed-loop in Figure 2, domain orchestration services will be consumed only by the management functions of domain intelligence services. The services provided by the domain orchestration service can be further divided into feasibility check services, domain orchestration services, managed service catalog management services, domain inventory information services, domain inventory management services, testing services, and domain topology information services.

Domain orchestration services control the accessibility of domain-level network services. It allows only authorized consumers to create, modify, or terminate the network services. A dedicated domain service stores the complete description of the resources and services, topology, configurations, and related policies. It may contain the orchestration workflows that need to be executed.

The feasibility check service can be used to check whether a particular parameterized managed service is deployable. It may indicate whether the requirements of the service can be met with existing resources in the management domain. It is also possible to make necessary reservations in the domain if the requirements can be fulfilled.

The catalog management service maintains a catalog of the services managed by each management domain. It also exposes the catalog to the required management domains. It may include supported coverage areas, supported SLS services, and service templates. The main three functions of this service include managing service models, managing service categories, and providing catalog notifications. This service carries out operations such as creating, reading, updating, deleting, and listing as part of the managed service models. Managing service categories also refers to creating, reading, updating, deleting, and listing service categories. This service also notifies you of catalog changes.

Domain orchestration also offers a testing service. It executes tests based on policies, configurations, and performance data before adding them to the live network. This testing service can be triggered when required. However, a test specification needs to be designed. The service allows checking the status of the ongoing test, its status, and results. The service provided includes managing test specifications, test resources, query tests, and providing test notifications.

The domain inventory information service allows querying information about the infrastructure resources and services offered by the management domains required for end-to-end service management. It is also possible to provide information at different abstraction levels based on the requirements.

The domain inventory management service gathers information about available infrastructure resources and managed services. However, this is a service provided for consumers in the domain only. The domain topology information service is also a service offered by the orchestration service, and it provides information about the topology of the resources and services in the management domain. It is made available via cross-domain data services. The configuration service is responsible for configurations such as setting policies. Query topology information is responsible for allowing querying information about the topology of the infrastructure. Resources include services, physical and virtual resources, physical or virtual links, or network connections. The level of abstraction of the information depends on the requirements of the service.

- Domain control

This service allows you to steer the state of the managed entity. The domain control service will be controlled based on the relevant entities' configuration settings. The service offered by the domain control service can be consumed by the services of the domain orchestration group, which intends to change the state or configuration of a managed entity in the domain. The control group can also control virtualized resources. Domain control services are further classified into three types: resource configuration management services, resource lifecycle management services, and configuration data generation services.

The resource configuration management service is for managing the configurations of the resources. Domain control also manages the life cycle of the resources of the domain. The lifecycle includes operations such as instantiating, updating, scaling, healing, and termination. However, it may vary with the application involved. Apart from the generic services such as managing resource lifecycles, providing notifications of life cycle changes, and managing subscriptions to lifecycle changes, domain control services are also capable of handling the virtualized resource lifecycle.

- E2E management domain

The E2E management domain is a significant component of the ZSM architecture. Similar to the management domain, it can be further divided into E2E service orchestration, E2E service intelligence, E2E service analytics, and E2E service data collection. Each of these is described below as appearing in [2], [6], and [7].

E2E service data collection monitors the availability and quality of the customer-facing services. I.e., the E2E service quality is monitored, and the end-user experience is verified based on data provided by the data collection services. This service provides performance data such as KPIs to authorized consumers.

E2E service analytics handles E2E service impact analysis, and root cause analysis generates service-specific predictions. It also includes service level specifications (SLS) monitoring and KPI monitoring. The analytics services of the domain process incoming events and event information related to faults to analyze the data and derive insights. Analytics services also keep track of the end-to-end service KPIs and user experience. Probes or the underlying service-specific observation mechanism are used for this purpose. It is possible to determine when and how the analysis results are provided. The capabilities offered by the E2E analytics service include the E2E anomaly detection service, root cause analysis service, deployed AI model performance and evaluation service, impact analysis service, security analytics, and service performance analysis service. Each of these is described in Table 1.

Table 1 Specific analytics services defined by the ZSM framework

Service name	**capability**
E2E anomaly detection service	The capabilities of this service include detecting anomalous conditions related to the E2E service by using the information made available by the data collection services. Conditions such as security violations, fault conditions leading to short-term unavailability of the E2E services, and inconsistency of the services are monitored by these services. In the event an anomaly is detected, the service will trigger the E2E service intelligence services to take appropriate actions. It is also

Service name	capability
	possible to carry out further analysis rather than conclude with the available data.
Deployed AI model performance evaluation service	Similar to the domain level, this service also has the capability to detect the performance level of the deployed AI models and to detect performance degradation. This service monitors the model for performance degradation, sudden performance drops, irregular performances, etc.
Root cause analysis service	This service investigates the root cause of the service by correlating available information. The available information includes multiple fault information, dependency information, and additional insights offered by other services.
Impact analysis service	This includes determining the impact of the faults or other problems on the service.
Performance analysis service	The performance analysis service analyses the data provided by the data collection services of the underlying domain to obtain insights into the service performance.
Security analytics service	This service processes security events to recognize specific attack patterns, suspicious behavior, and threats that may lead to security implications.

The E2E service quality management service is another service that belongs to E2E service analytics. It is responsible for allowing only authorized consumers to manage the service level objectives (SLO) and SLS of the E2E services. SLS or SLO are related to the service level agreements that determine network performance. A Service Level Agreement (SLA) is often a legally binding contract between the service provider and any other network user.

The E2E service condition detection service is responsible for decomposing the conditions into parts supported by individual management domains; for example, decomposing the SLS can be done. This service then tracks whether these conditions are met and generates notifications if the conditions change. It is also possible for other services to consume this service.

Service intelligence drives the closed-loop automation in the E2E domain with AI. The level of automation may differ to various degrees based on the application requirements. It is also further categorized as decision support, decision-making, and action planning. Decision support services support making decisions with the help of technologies such as AI and ML. E2E service data collection information is used for E2E service management decision-making. The other sources used for information include E2E service data collection, E2E service analytics, and domain data services in the E2E service management domain. The services provided by the E2E service intelligence service include the AI model management service, the deployed AI model assessment service, the AI training data management service, the AI training data management services, and the E2E service health issue-reporting service.

Table 2 Service capabilities of E2E service intelligence

Service name	**capability**
AI model management service	The main capability is managing AI models and updating the model based on new inputs. Managing means creating, reading, updating, deleting, and listing the AI models.
Deployed AI model assessment service	Monitoring the performance of the AI models. This is because the model may degrade over time. This service allows determining the most appropriate action for the model which has been indicated as degraded.
AI training data management service	The main functionality of this module is to store the data for training and retraining data. This service is capable of creating, reading, updating, deleting, and updating the data.
E2E service health issue-reporting service	This generates reports about the health of the E2E services.

E2E service orchestration manages catalog-driven E2E orchestration of multiple management domains for the creation, deletion, or modification of customer-facing services. In the same way that management has a service model that illustrates how the various services are interconnected, this domain has a service model as well.

Table 3 Service capabilities of E2E service orchestration

Service name	**capability**
E2E service orchestration service	This service allows consumers to create, modify, and terminate E2E services. It also allows for the automation of workflows. This service applies to the E2E service model and is managed by the managed service catalog management service in the E2E management domain. ZSM framework consumers can deploy E2E services by using E2E orchestration services.
Feasibility check service	The feasibility check service checks whether the E2E services can be deployed at the expected service level. In the same way as the management domain, the feasibility check of the E2E management domain includes things like checking deployment capabilities and checking and reserving services.
Managed services catalog management service	This service manages the catalog of the service models of the available E2E services operated by the E2E service management domain. It also makes the catalog available to other consumers. The service models can be grouped into service categories and may include information such as supported coverage areas, associated SLSs, service templates, etc. Service models typically have a life cycle as well.
E2E testing service	The testing service carries out active and passive tests to determine whether the end-to-end services or parts of them are functioning

Service name	capability
	properly. Tests are executed based on the test specifications and can be triggered as required by the administrator. It also shows the test status and results of all performed tests.
E2E services inventory information service	This service allows for querying information about E2E services. This information is based on the aggregating inventory information given by the relevant management domain.
E2E services inventory management service	The E2E service inventory services collect information about E2E services. However, this service can only be consumed by consumers inside the E2E service management domain.
E2E services topology information service	This service is responsible for providing information about the topology of the infrastructure from the perspective of E2E services. It is provided at different abstraction levels based on the application requirements.

In the architecture described here, closed-loop operations play a significant role in network automation. Apart from automation, closed-loop operations make network optimization and behavior adaptation possible. The domain-level security closed loop, as shown in Figure 1, also has five stages: knowledge, orient, observe, decide, and act[6].

Closed loops are essential for end-to-end automation and zero-touch management of network services and infrastructure [2]. Figure 2 shows the indicative mapping between architectural building blocks and closed-loop steps [2].

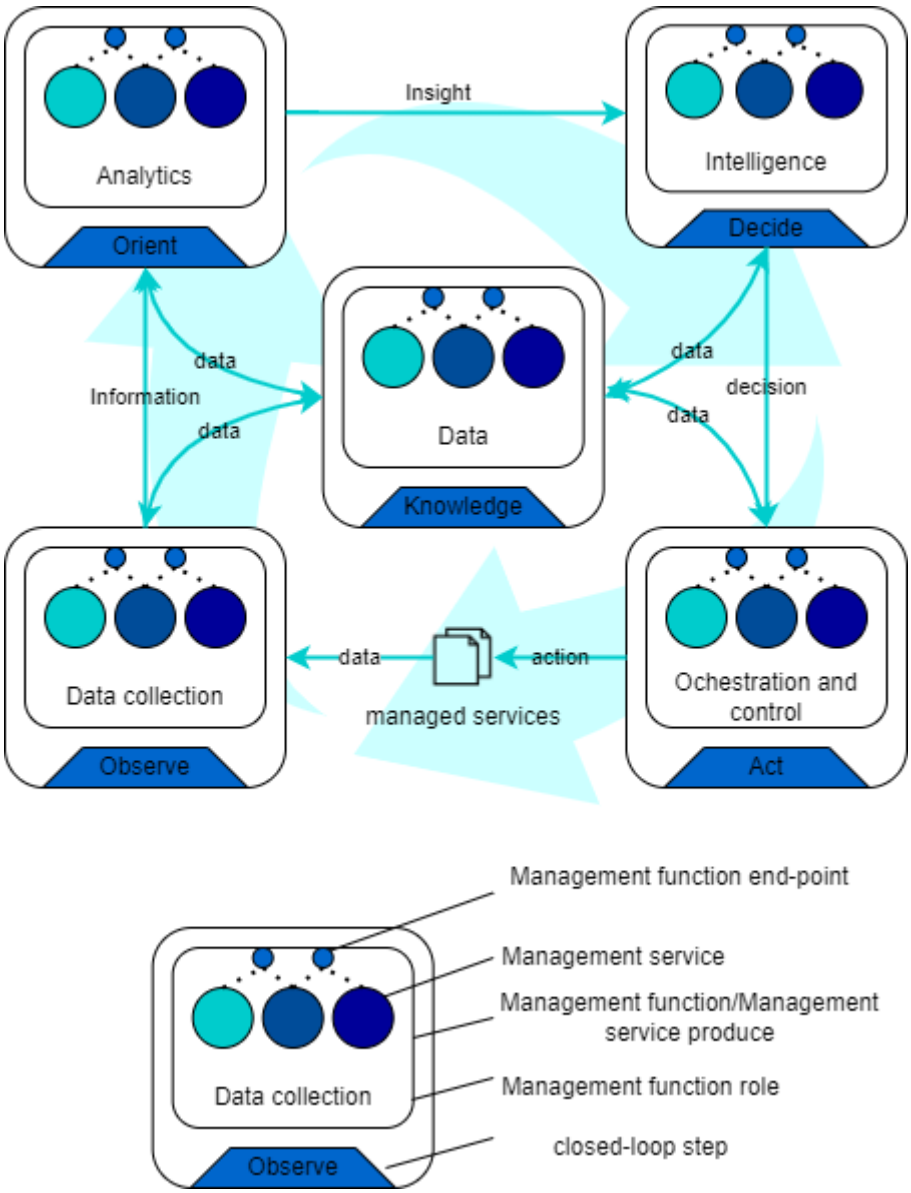

Figure 2 Indicative mapping between architectural building blocks and closed-loop steps [2]

Observation and collection of relevant resources are carried out as part of a closed-loop, after which the data is oriented so that its meaning and significance can be understood. Filtering, correlation, or any other similar mechanism based on application is used in analysis or orientation. Decisions are made using AI/ML mechanisms that can recognize patterns or trends and predict them. Based on the circumstances, the decision-making component also makes plans, determines root causes, or performs similar processes, which are then implemented. Based on the requirements, it is also possible to work collaboratively with other closed loops [2].

Similarly, in a security closed loop, security agents located throughout the network collect data and submit it to the security data collector, who monitors the data. Data processing and analysis are performed by the security analytic service. The inferred information is then provided to the decision engine, which can decide the best course of action to run the network optimally while mitigating any risk signaled by the data. Following that, the security orchestrator will impose the decision engine's recommendations on the relevant entities. The

data analytic engine will then examine the outcomes of the data collected by security agents and security data collectors [6]. As Benzaid et al. [7] have mentioned, open application programming interface (API) security threats, intent-based interface security threats, security threat-driven network automation, AI, ML-based attacks, and security threats due to the adaptation of programmable network technologies are the threat surfaces in the ZSM architecture [6].

Apart from single-closed loops, it is also possible to use cross-closed loops as well. As per Xie et al. [8], crossed closed loops can be used in applications such as URLLC where strict latency requirements are required. They also recommend APIs, intent and policies, stimuli, and workflows as some of the elements required for cross-closed-loop collaborations. There could be other elements as well, depending upon the requirement. However, as Vaishnavi et al. [9] highlight, the use of closed loops in 5G has challenges such as fewer or no standards among closed loops used by different vendors/applications, the possibility of influencing the closed-loop operations, operator reluctance to trust the closed-loop, and conflicts arising due to unplanned interactions.

2.2 Security Threats in ZSM architecture

Due to the architecture shifts in the network to adapt automation, new threats can emerge along with the existing threats. Table 1 shows the summary of the threat landscape as explained by Benzaid et al. [4] in the ZSM architecture.

Table 4 Security Threats in ZSM architecture

Enabler	**Security threat**	**Mitigation measures**
AI/ML	Adversarial attacks - attempting to change the model by injecting malicious data.	• Input validation • Adversarial training: Exposing the model to adversarial examples in order for it to become more resistant to them. • Defensive distillation: increasing the ML model's robustness by using its own knowledge on adversarial examples. • Défense Generative adversarial networks (GANs): Prior to sending the input samples to the ML model, the samples are passed through the GAN's generator to ensure that adversarial examples are not included. • Concept drift
	• Model extraction attacks: stealing the model parameters to recreate the model	• Control the information provided by APIs to the ML • Adding noise to the ML prediction

Enabler	Security threat	Mitigation measures
	• Model inversion attacks- infer training data	• Adding noise to the execution time of the ML model
Software-defined networking (SDN)/NFV	• Spoofing – impersonating OpenFlow switch/controller[10] • Privilege escalation • Information disclosure • Tampering – tampering with configuration data, etc.	• Mutual authentication to prevent impersonation of SDN applications, controllers, and switches. [11] • Authorization mechanisms such as Base Closure and Realignment (BRAC) to control access levels to prevent privilege abuse. • Secure communication by using techniques such as digital signatures, encryption, and message authentication algorithms. (MAC) • Trusted Platform Module (TPM), Virtual Trusted Platform Module (vTPM) for sheltering sensitive data such as passwords and encryption keys.
	• DoS [12],[13],[14]	• Malicious traffic monitoring using intrusion detection service (IDS) and firewalls • Limiting the number of flow requests • Resource monitoring and usage limitation • Resource isolation • Distributed SDN controller architecture
	• Introspection Attacks: threats due to combining networking with virtualization technology	• TEE – hardware solution made for providing confidentiality and integrity
	• Software vulnerabilities	• Secure software patching procedures • System hardening techniques – removing unnecessary services, enabling operation systems (OS)-level access, and customizing default configurations.
Open API APIs are to be used for communication	Parameter attacks: take advantage of data sent into the API, such as Uniform Resource Locator (URL)s, query parameters, and HTTP headers.	• Input validation

Enabler	Security threat	Mitigation measures
between interfaces	• Identity attacks: exploiting the flaws in authentication, authorization, and session tracking, they could gain control of the management domain and E2E domain.	• Authentication and authorization controls: empowering authentication by OAuth2.0 or JSON Web Token (JWT) token, RBAC, ABAC, and ACL for authorization. • Message encryption and using protocols such as Transport Layer Security (TLS) 1.2 for secure communication.
	• Tampering attacks	
	• Man in the middle (MITM) attack: exploiting data by a third party positioned between API consumer and producer, especially the unencrypted data,	
	• (D)DoS – A massive volume of requests can lead to a denial of service.	• Rate limiting/throttling the usage of APIs, • Use of API gateways and micro gateways.
Intent-based interfaces- is used to achieve a high level of interactions	• Information exposure, intent tampering – expose information such as application desires	• Authentication (OpenID Connect, signed JWT tokens etc.) and authorization (OAuth2.0, RBAC, etc.) mechanisms for ensuring authentication between intent producer and intent consumer and controlling the access to the intent-based interface.
	• Malformed intent- could interfere with service, especially orchestration services	• Intent format validation- providing an intent engine with capabilities to validate the intent format.
	• Conflicting intents	• Conflict detection/resolution

Apart from the security threats inherited from the ZSM architecture, it is also possible to have the security threats and anomalies that were available in the previous architectures as well. Analysis, DoS, Backdoors, Exploits, Reconnaissance, Generic, Shellcode, Fuzzers, and Worms are some of the possible types of attacks. While there can be many more, the same set of attacks has been used for the data set as well. Therefore, each of those is described below.

- DoS

DoS is the act of making service unavailable to valid users through interruptions or by suspending the service. According to Moustafa et al. [15], it is a malicious attempt to make a server/network resource unavailable by temporarily interrupting or suspending the services of

an Internet-connected host. Sending many requests to a server is a well-known attack type. Buffer overflow attacks, ICMP floods, and SYN floods are very popular types of attacks that lead to service unavailability.

- Analysis

As per Moustafa et al. [15], port scanning, spam, and HTML file penetrations fall under this category.

- Backdoors

A back door is usually a hidden way by which access can be gained to the system. It is possible to bypass system security mechanisms to access the data or service.[16]

- Exploits

The attacker manipulates the known security issues in various systems, such as operating systems or software programs, and conducts malicious activities. [17]

- Reconnaissance

This refers to collecting information from tools such as debugging tools and configuration tools to gather information about the target. Some common examples are packet snipping and port scanning.

- Generic

It is stated as a technique that works against block ciphers that have a given block and key size, and the structure of the block cipher is not considered.

- Shellcode

Shellcode is a piece of code inserted that can be used by attackers to exploit software vulnerabilities.

- Fuzzers

Fuzzers feed randomly generated data and attempt to suspend a program or network. [18]

- Worms

A computer software that replicates itself or self-propagates in order to reach other targets. Code Red, Slammer, and Nimda are a few examples of worms. [19]

2.3 Federated Learning

FL is a novel ML technique that utilizes decentralized edge devices for training models. As the model is trained on the device using locally available data, privacy is preserved. FL has been used for many IoT applications for anomaly detection. However, FL techniques also have a set of security threats that need to be considered when adapting them for any application.

As described by McMahan et al. [20], the first use case of FL is to take advantage of the rich data available on mobile devices without affecting the customer's privacy. The authors suggest FL as a technique to be used for training on real data from mobile devices. The data to be used or labels can be determined by the user interactions for supervised tasks.

As in any ML model, there is an optimization problem in FL. The model proposed by McMahan et al. [20] mainly targets the photos or language-related applications on mobile phones, tabs, or devices of a similar kind. Therefore, the optimization problem has properties such as non-IID, unbalanced, massively distributed, and limited communication [21]. The ML optimization problem is stated as below in McMahan et al. [20], which is the loss of the prediction on $(x_{i,}y_i)$ for the weight set w.

$$min_{w\,\epsilon\,R(D)}\, f(w)$$

$$f_i(w) = l(x_{i,}y_{i,}: w)$$

For FL, the problem can be mathematically demonstrated as below.

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

Where,

$$F_k(w) = \frac{1}{n_k} \sum_{i \epsilon\, p_k} f_i(w)$$

'k' is the number of clients, and p_k is the number of samples available for each client, and n_k is defined as $n_k = |p_k|$. They also propose using a federated stochastic gradient descent (SGD) algorithm for the optimization. Nevertheless, as presented in the results, simple averaging techniques have shown better accuracy for the used test data and training data.

Kairouz et al. [22] explain how the life cycle of a federated model takes place. It starts with the problem identification done by a model engineer. The next step is client instrumentation, where clients locally store the necessary data for training. Then the model engineer prototypes a model and sends it for federated model training. The trained models that are received are further evaluated for the purpose of selecting the best models. Finally, the best model is deployed on all devices. Figure 3 shows the federated model as presented by Kairouz et al. [22],

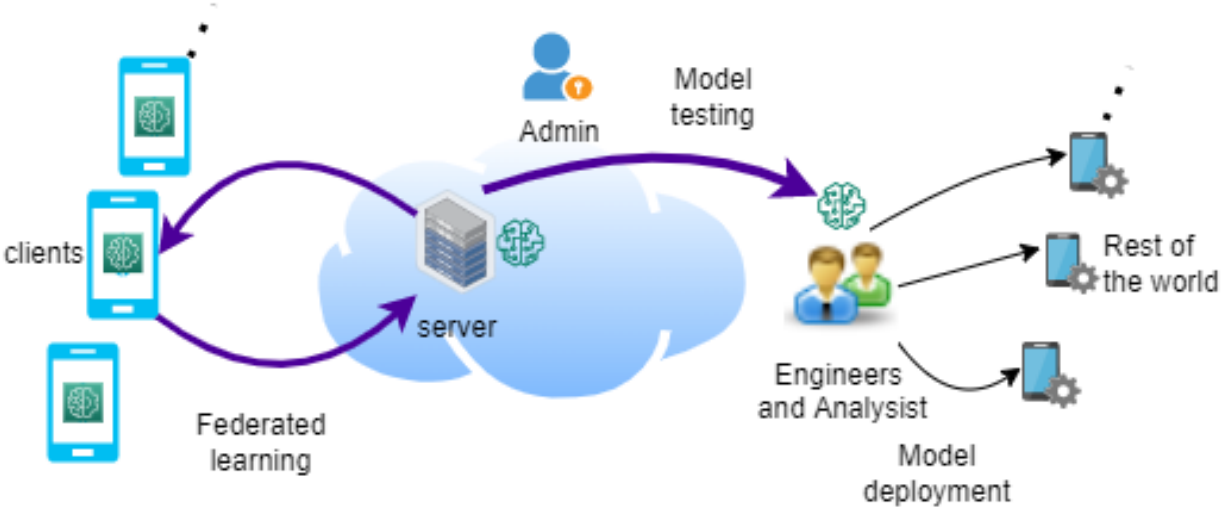

Figure 3 The FL model [22]

The federated training process in the cycle above plays a significant role, and its components are as below.

1. Client selection: The server selects a set of clients meeting eligibility criteria.
2. Broadcast: The selected set of clients download the set of current model weights and the training program from the server.
3. Client computation: The set of devices locally computes a model.
4. Aggregation: The server aggregates the parameters received from the devices.
5. Model update: The server shares the aggregate model back with the clients.

It is possible to change the model type used in FL. The proposed model uses artificial neural networks (ANN) for detection. Neural networks are a subset of ML that mimic the human brain [23]. It optimizes the set of weights assigned to each neural network. ANN can be further divided into types, such as recurrent neural networks and convolution neural networks. An ANN model is made up of three layers: an input layer, a hidden layer or layers, and an output layer. Apart from the weights assigned to each layer, optimization and activation functions also play a significant role.

Long Short-Term Memory (LSTM) is also a commonly used network model for intrusion detection. For example, Malhotra et al. [24] use LSTM in anomaly detection. In LSTM, a new activation layer known as "gates" is introduced to ensure that the network retains a memory of the previous event [25]. A gated recurrent unit (GRU) is also similar to LSTM and is considered a recurrent neural network [26]. Qu et al. [27] have successfully used GRU for their anomaly detection algorithm.

FL can be used for wireless communication applications, as highlighted in Niknam et al. [28]. Edge computing and caching, autonomous driving, federated ML for spectrum management, the coexistence of heterogeneous systems (e.g., DSRC and c-V2X), and federated ML in 5G core networks are among the applications highlighted by the authors.

While using FL for various purposes can generate many benefits, it also inherits some challenges. As Niknam et al. [28] mentioned, even though FL preserves privacy, it is also possible to analyze the global data and disclose clients who participated. The FL models may undergo model re-training with new data. An attacker can carry out model-poisoning attacks during this instance. I.e., training data is carefully altered to generate a result expected by an intruder. For example, Bhagoji et al. [29] demonstrate how model poisoning can be done in FL-based applications. The FL models are also susceptible to membership inference attacks. As described by Melis et al. [30], during an inference attack, one of the adversaries pretends to be an honest client and infers the training data by participating in the training algorithm. Inference can be further classified as parameter inference, input inference, and attribute inference attack.

Apart from the privacy breaching or security breaching incidents mentioned above, it is also possible to have challenges relevant to the algorithm, as highlighted in Nikman et al. [28]. Some of them are related to convergence and optimization problems, such as deciding the optimum number of participants. Nikman et al. [28] also mention a few issues related to wireless communication in the wireless channels. Due to network capacity, information is quantized before being sent over the wireless network; as a result, parameter quantization needs to be considered for FL. As a result, it will naturally inherit quantization errors. Wireless channel quality could also affect convergence [28]. Liu et al. [5] also explain using FL in 6G. High security and privacy, and high operational, service, and application intelligence are key features of the 6G network. Apart from the benefits, there are challenges such as expensive communication, security problems, privacy concerns, and effectiveness issues.

Apart from the challenges, Liu et al. [5] also highlight the possible remedies, such as using a communication-efficient FL algorithm. An asynchronous FL system could reduce the computation time of devices by asynchronously aggregating the model updates to increase communication efficiency. Xie et al. [31]., Chai et al. [32], van Dijk et al. [33], and Chen et al. [34], present some of the algorithms available for asynchronous FL. To increase the efficiency level at the algorithm level, it is also possible to increase the communication efficiency by using gradient compression techniques. For the SGD algorithm based on zero, first, or second-order, optimization can be used for gradient compression [35]. It is also possible to reduce the communication overhead by gradient sparsification and gradient quantization, which lead to efficiency. Lin et al. [36] propose a top-k selection-based gradient compression scheme.

To secure the FL mechanism from malicious attacks, Liu et al. [37] suggest using robust aggregation algorithms, robust detection algorithms, and reliable reputation management. Liu et al. [37] propose a smart contract technique using blockchain to detect malicious devices. A reputation management scheme can be found in [38], which calculates the historical reputation of the device. Differential privacy (DP) techniques, such as deep net pruning and gradient compression, are popular techniques for preserving privacy, as presented in related work. To increase the effectiveness of FL, it is also possible to use techniques such as federated parallelization, federated distillation, and efficient inference [39], [40].

FL-based applications also face an inevitable set of challenges such as system and statistical heterogeneity other than security-related problems. System heterogeneity refers to the difference between storage, computational, and communication capabilities in devices participating in FL-based applications [21]. As a result, unreliable results are possible, and the number of participating devices may be drastically reduced. Each device may collect non-identically distributed data, and it may not be possible to assume that the data is independent and identical. Therefore, each of these needs to be anticipated when designing the application. As highlighted by Trab et al. [41], uncertainty in the wireless channel could also contribute to the algorithm, and it needs to be considered in the optimization technique.

2.4 FL-based anomaly detection

FL has been used for anomaly detection, mostly in IoT devices. Ferrag et al. [42] provide a summary of the set of IoT-related applications that use FL. Some of the available work is presented here.

Wang et al. [43] developed a hierarchical FL anomaly detector for industrial IoT devices that uses deep reinforcement learning to train the local model on each device. They have used three types of anomaly detectors: the global anomaly detection center (GADC), the local anomaly detection center (LADC), and the regional anomaly detection center (RADC). FL enables the use of anomaly detectors. It is proposed to check for anomalies in internal users and use an FL-trained model. The model used consists of both reinforcement learning and deep learning algorithms. Most of the other cases where FL is used are IoT-related applications.

To address the growing number of devices, Nguyen et al. [44] suggest a federated-based solution for anomaly detection in the IoT. The proposed model, which is built for a small office or home office, consists of a security gateway and an IoT security service. The security gateway monitors the systems and identifies compromised or malicious devices. IoT devices connect to the internet via these security gateways, and anomaly detectors placed in the gateways monitor devices for anomalies. The IoT security service maintains a repository of device-type-specific anomaly detection models. The security gateway trains a local model with available data and shares the parameters with the aggregation server. The aggregation server caters to a set of security gateways, and it aggregates similar parameters sent by other servers. GRU models were used for detection.

Yadav et al. [45] also present an unsupervised FL-based IoT intrusion detection system. For detection, they used an autoencoder and an ANN model, and the CICIDS 2017 data set was used to test and train the proposed model. The IoT devices connected to the network first receive a random set of weights, which are then trained with locally available data. The new parameters are then shared with the server, as in any FL model. The anomalies detected include brute force (File Transfer Protocol (FTP)-Patator and SSH Patator), DoS Solaris, DoS Slow HTTP Test, DoS Hulk, DoS Golden Eye, Heartbleed, Web attack-Brute Force, Web attack-XSS, Web

attack-SQL Injection, Botnet, Port Scan, Infiltration, and Distributed Denial of Service (DDoS) LIoT. The maximum accuracy achieved is mentioned as 97.75%.

Liu et al. [46] use a FL framework to train a deep anomaly detection (DAD) model in a collaborative manner in industrial IoT applications. The proposed system consists of a cloud aggregator, and the main two mechanisms mentioned are anomaly detection and gradient compression. The cloud aggregator server is responsible for initializing the global model, sending it to the local devices, and aggregating the model parameters received from the end devices. The gradient compression mechanism, which reduces the number of gradients exchanged between server and end device by compressing, is a significant feature. The tested model types include LSTM, GRU, stacked autoencoders, and support vector mechanisms.

Wei et al. [47] present a FL-based anomaly detection solution for 5G heterogeneous networks, which includes three major components: user end devices, the edge, and the cloud. The proposed system focuses on a network with end devices, edge devices, and a cloud. The anomaly detecting mechanisms are distributed on edge and end devices. The end devices use deep reinforcement learning to train models to detect attacks. The model parameters of the trained models are shared with the aggregator server located at the edge. The detectors situated on the edge have their aggregator servers in the cloud.

Li et al. [48] present "DeepFed", a system for utilizing federated deep learning for intrusion detection in industrial cyber-physical systems. It consists of three major entities: trust authorities, cloud servers, and industrial agents. The trust authority is responsible for generating public keys and private keys as the proposed system uses a Paillier public-key cryptosystem-based secure communication protocol. The cloud server is used as the aggregator server, and industrial agents are the local devices for building local models based on available data. The proposed system is built to handle attacks such as reconnaissance attacks, response injection attacks, command injection attacks, and DoS and DDoS attacks. They are proposing a CNN-GRU-based intrusion detection system.

The proposed model in [49] is a deep learning-based self-adaptive algorithm with two stages for detecting anomalies in 5G networks. However, FL is not used in this scenario. They have used the ETSI NFV architecture as the basis with two VNFs, namely Anomaly Symptom detection (ASD) and Network Anomaly Detection (NAD). ASD is located in the RAN, and it inspects the network flows to identify anomalies. NAD collects timestamped and RAN-associated symptoms. There is also a central process that analyses the timeline and the relationship between these symptoms. They have proposed using input performance monitoring indicators as inputs to the security policy manager in the 5G network. The security policy manager will be able to adapt RAN resources, extending ASD and NAD functions. Each function is described below.

- Adapting RAN resources: deploying new virtual resources if an overload of any single resource is detected.
- Optimizing ASD and NAD functions – implies changing the deep learning framework or detection model to suit the variations in the network flow volume.
- Extending ASD and NAD – deploying the specific detection component allowing inspection of L2/3 flows etc.

The proposed system consists of two stages. In the first stage, detectors do not have the capability to figure out the responsible party. This task is done by the second stage of the proposed model, which is placed in the evolved packet core. The proposed system also allows the deployment of symptom detectors in the network where the network traffic increases.

In addition to security, there are also AI models in the proposed architecture that are used for things like network resource optimization and proactive and reactive incident analysis in the

domain analytic service. AI can help with service management, and Rizwan et al. [50] show how to use AI-enabled CDR analysis for service management in zero-touch networks. For identifying suboptimal network performance and root causes, they used k-means clustering, support vector mechanisms, and their own algorithm.

3 PROPOSED ARCHITECTURE

The ZSM architecture includes data analytics services, and many of the components are equipped with ML techniques. As a result, ML can be used for network security, among other things. However, it also brings with it unavoidable issues such as resource constraints and emerging privacy concerns. FL provides advantages such as privacy protection and improved communication efficiency. As a result, the mechanism presented here employs FL in the ZSM architecture for anomaly detection. Model training and detection are the two major components of the proposed mechanism.

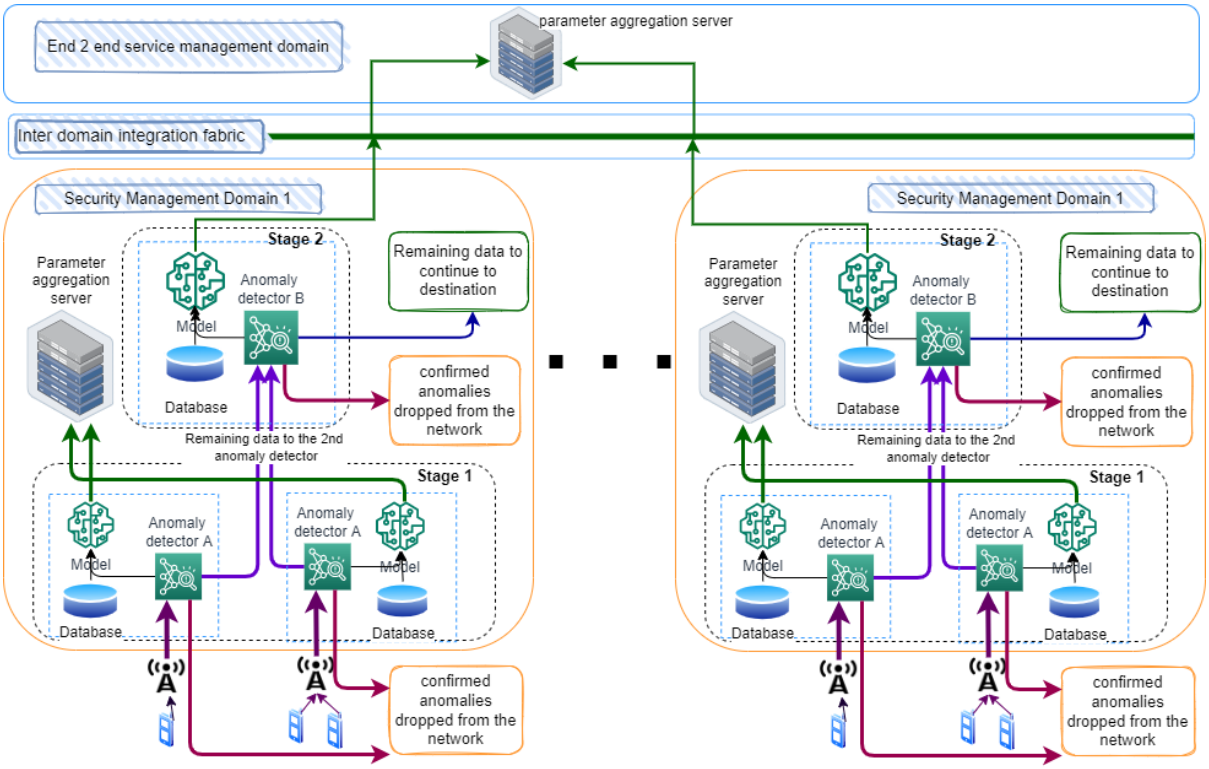

Figure 4 Proposed hierarchical FL based anomaly detection mechanism and its mapping with the ZSM architecture

It is possible to build a domain inside another domain if it is useful to group a set of functions together. Therefore, for this purpose, the proposed system is placed inside the security management domain of each domain. Figure 4 depicts the security management domain only. It is important to note that it resides inside the domains, such as RAN, described in the early chapters. There are two types of detectors placed inside each domain: type A and type B. Type A detectors communicate with the parameter aggregation server placed inside the domain. However, anomaly detector B communicates with the aggregation server placed in the E2E management domain. The aggregation server inside the domain communicates with the type A detectors inside each security management domain. The servers inside the E2E management domain communicate with the type B detectors of all domains.

As illustrated in Figure 4, each detector consists of a database, a model, and a detector. The database stores a set of network flows that could be used for training the model. The model represents the training component of each detector. The model, communicated by the relevant aggregation server, will be trained using the data available in the database. Both type A and

type B detectors have the same structure. Type B detectors, on the other hand, are more capable. I.e., the databases of type B detectors are capable of storing more network flows, and models can be made more complex compared with stage A.

The proposed model is an FL-based hierarchical security scheme that detects anomalies in network traffic in two stages. The two stages each have a different ML model, and the idea is inspired by two reference security schemes proposed for IoT networks [49] and [47]. As shown in Figure 4, and in compliance with the ZSM architecture, network traffic anomaly detection is performed at the management domain and the E2E management domain levels, respectively. Each anomaly detector comes with an FL-based model and a database, as well as two types of parameter aggregating servers that are positioned for detection. The setup is further described in the following sections, which cover two major mechanisms.

3.1 The detection mechanism

The network flow will be routed through two anomaly detectors as it enters the network. The first anomaly detector in the security management domain uses a simple ML model with smaller databases. It enables the placement of an adequate number of detectors in the network while adhering to resource constraints and addressing privacy concerns. The ML models of all the anomaly detectors relate to an aggregation server. Each stage of a domain has a single aggregation server that serves a set of anomaly detectors that aggregate the model parameters based on a pre-determined criterion. Network flows that are confirmed to be anomalies will be dropped from stage 1, anomaly detector A, and the remaining data will be analyzed at stage 2, detector B.

The data filtered from stage 1 will be sent through the detector in stage 2, and it is marked as detector B in Figure 4. As per the proposed model, detector B has a complex ML model and a larger database compared with the stage 1 detector. Stage 2 detectors serve as a set of detectors that belong to stage 1. Stage 1 detectors and stage 2 detectors are considered part of the security management domain. However, the stage 2 aggregation servers are placed in the E2E management domain. Like in stage 1, the network flow identified as anomalies will be dropped from the network, allowing the regular traffic to continue to its destination.

3.2 The training mechanism

A training mechanism is required to generate the model parameters for the anomaly detectors in stages 1 and 2. The training mechanism can be further divided into initial training and training in case of a missed anomaly. It is required to train the model at the start of the detector in the network, and it is necessary to train the model if any anomaly goes through the network undetected.

As expected in an FL-based model, model parameters generated by the model at each detector will be aggregated at the aggregation server based on a predefined criterion. Initially, a set of data will be stored in the database of each model, allowing it to generate parameters that are eventually shared with the aggregation server. The aggregation server will take the average of the parameters shared with it and transmit them back to the detectors as the new set of parameters for detection.

Future networks are expected to experience unprecedented events, and ZSM is no exception. A mechanism is proposed to prevent recurrences if the detectors fail to identify it as an anomaly during an unknown attack. A security analytics engine in each domain can generate alarms if any undetected anomaly affects the network performance. As per the proposed

mechanisms, two messages will be generated by the same. A notification containing the undetected anomaly will be sent to both the stage 1 aggregation server and the E2E orchestration engine. Two mechanisms for both messages are demonstrated in Figures 5 and 6, respectively.

Figure 5 demonstrates how the components related to stage 1 react to the notification from the security analytics engine. The message will be first received by the stage 1 aggregation server. The aggregation server will communicate it to the model of stage 1 detector, which will train the model with the network flows relevant to the period as directed by the stage 1 aggregation server. After completing the training, the generated parameters will be sent to the stage 1 aggregation server. The aggregation server will generate a set of parameters based on the received values and will communicate them back to the stage 1 anomaly detector model. Even though only one cycle is depicted in Figure 5, several rounds of communication will take place as configured in the network. After completing the given number of rounds, the final set of parameters will be sent to the stage 1 anomaly detector to be used for future detection. Meanwhile, as a part of the closed-loop mechanism embedded in the ZSM architecture, the effect of the new parameters will be observed by the network, and modification will take place if required.

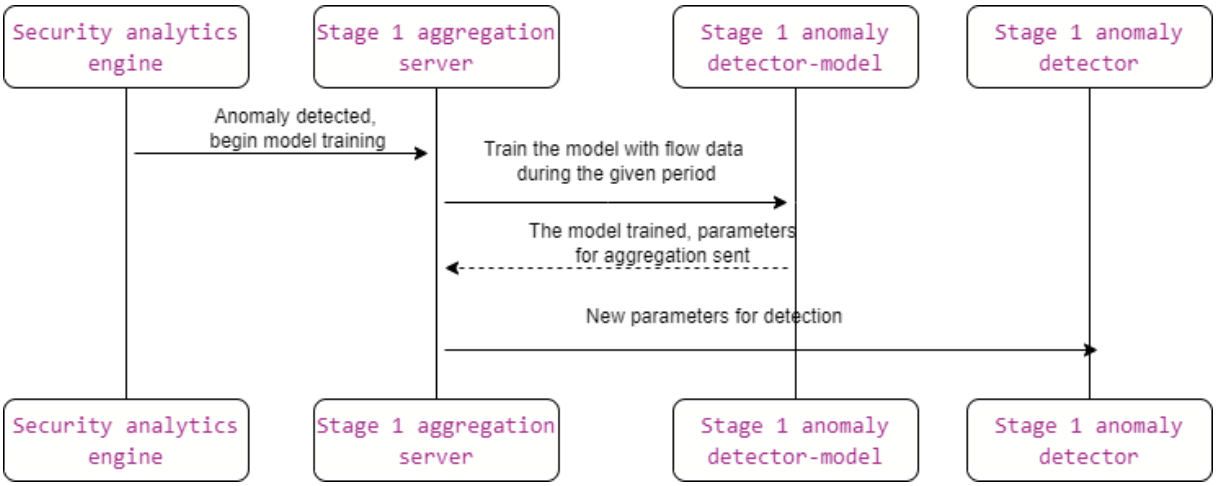

Figure 5: Flow diagram for stage 1 if anomaly is not detected

A notification will be sent to the E2E aggregation server, similar to the message sent to the stage 1 aggregation detector. The mechanism that takes place after receiving the notification is demonstrated in Figure 6. It should be noted that components such as integration fabric are not represented in the diagram for simplicity. The E2E orchestration engine will send two notifications, which are received by stage 2 aggregation servers and security analytics engines of other domains. Upon receiving the notification, the security analytics engines in other domains will check for similar events. The process explained above will take place if there is any undetected anomaly that is identified. The stage 2 aggregation server of the domain for which the anomaly is relevant will begin training when the notification from the security analytics engine is received. Similar to the process demonstrated above for stage 1, only one cycle for model training is indicated in the image for simplicity. However, several rounds will take place as expected in any FL-based application.

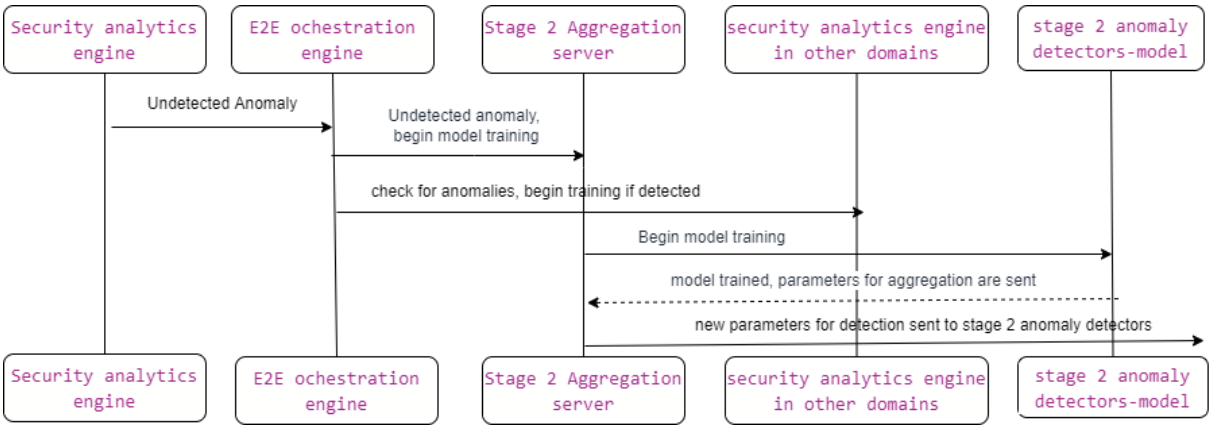

Figure 6 Flow diagram for stage 2 if anomaly is not detected

4 SIMULATION

This section describes the simulation setup used for two-stage anomaly detection. The simulation is described first, followed by the data set used for it. The simulation setup is further subdivided into model training and model testing. The data set was first chosen and prepared for the purpose. Then, the proposed model was trained and tested to see how well it worked.

The Jupyter notebook is the interface used for the simulation. An environment was created by installing the set of packages and libraries as required. A set of packages installed is shown in Figure 7, and a complete list is attached to the appendices. TensorFlow federated is the library used for developing FL environments. A set of programs were written for processing data, training, and testing the environment. The written program can be found in the appendices.

```
# packages in environment at C:\Conda\envs\OuluTFFProject:
#
# Name                    Version                   Build  Channel
_tflow_select             2.3.0                       mkl
abseil-cpp                20210324.2            hd77b12b_0
absl-py                   0.9.0                     pypi_0    pypi
aiohttp                   3.7.4.post0       py38h2bbff1b_2
argon2-cffi               21.1.0                    pypi_0    pypi
astor                     0.8.1             py38haa95532_0
astunparse                1.6.3                     py_0
async-timeout             3.0.1             py38haa95532_0
attrs                     19.3.0                    pypi_0    pypi
backcall                  0.2.0                     pypi_0    pypi
blas                      1.0                         mkl
bleach                    4.1.0                     pypi_0    pypi
blinker                   1.4               py38haa95532_0
brotlipy                  0.7.0             py38h2bbff1b_1003
ca-certificates           2021.10.26            haa95532_2
cachetools                3.1.1                     pypi_0    pypi
certifi                   2021.10.8         py38haa95532_0
cffi                      1.15.0            py38h2bbff1b_0
chardet                   4.0.0             py38haa95532_1003
charset-normalizer        2.0.4               pyhd3eb1b0_0
click                     8.0.3               pyhd3eb1b0_0
colorama                  0.4.4                     pypi_0    pypi
```

Figure 7 Packages installed in the Jupyter notebook environment

4.1 The UNSW-NB 15 data set

The data set contains 257673 data flows related to 49 features. Source and destination IP addresses, source and destination port numbers, protocols, destination to source, and source to destination transaction bytes are among the features. Table 2 shows the complete list of features [15], [51].

Table 5 List of Features described in the data set [15]

No.	Name of the feature as indicated in CSV files	Type	Description
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	Indicates the state and its dependent protocol, e.g. AC, C, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	http, FTP, SMTP, ssh, DNS, FTP-data, IRC, and (-) if not much-used service
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the row packet size transmitted by the src
24	dmeansz	integer	Mean of the row packet size transmitted by the dst
25	trans_depth	integer	Represents the pipelined depth into the connection of HTTP request/response transaction
26	res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server's HTTP service.
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)

No.	Name of the feature as indicated in CSV files	Type	Description
33	tcprtt	Float	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
34	synack	Float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
35	ackdat	Float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
36	is_sm_ips_ports	Binary	If source (1) and destination (3) IP addresses equal and port numbers (2) (4) equal then, this variable takes value 1 else 0
37	ct_state_ttl	Integer	The number for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd	Integer	Number of flows that has methods such as Get and Post in HTTP service.
39	is_ftp_login	Binary	If the FTP session is accessed by user and password, then 1 else 0.
40	ct_ftp_cmd	integer	Number of flows that has a command in FTP session.
41	ct_srv_src	integer	Number 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	integer	Number 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	integer	Number of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ ltm	integer	Number of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm	integer	Number 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	integer	Number 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	integer	Number of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
48	attack_cat	nominal	The name of each attack category. In this data set, nine categories, e.g., Fuzzers, Analysis, Backdoors, DoS

No.	Name of the feature as indicated in CSV files	Type	Description
			Exploits, Generic, Reconnaissance, Shellcode, and Worms
49	Label	binary	0 for normal and 1 for attack records

However, some of the features, such as the Attack category, were removed during the simulation process. Before beginning the simulation, 10,000 randomly selected samples of the data set were allocated for testing, while the remaining data was used to train the two models. The number of samples used to train the stage 1 model was in the 6000–14000 range, and around 99,000 samples were used to train the stage 2 model.

4.2 Simulation set-up

Model training and model testing are two major parts of the simulation. First, model parameters for different scenarios were generated and saved. In the training phase, the saved parameters

Algorithm 1: Training and testing algorithm

```
Data: Data set, Number of clients, Array of anomaly ratios for stage 1,
      Array of anomaly ratios for stage 2
Result: Test accuracy for Anomaly percentages
Array1 =Array of anomaly ratios for stage 1;
Array2 =Array of anomaly ratios for stage 2;
Test data= randomly select 10,000 from Data set;
Test data for each client = Test data÷Number of clients;
for Each value in array2 do
    Train data for stage 2= Randomly select train data with relevant
     data percentage from array2;
    Train data for each client in stage 2 = Train data for stage
     2÷Number of clients;
    for 100 rounds do
        Train the model with train data for stage 2;
        save model parameters;
    end
end
for Each value in array1 do
    Train data for stage 1= Randomly select train data with relevant
     data percentage from array1;
    Train data for each client in stage 1 = Train data for stage
     2÷Number of clients;
    for 100 rounds do
        Train the model with train data for stage 1;
        Save model parameters;
    end
end
for 100 rounds do
    for each value in Array1 do
        for each value in Array2 do
            Load the model parameters;
            Test the model;
            Save test accuracy;
        end
    end
end
```

Figure 8 : Algorithm for training and testing the system

were used for detection. This section describes two phases separately. Figure 8 shows the algorithm for training and testing.

4.2.1 Model training

The first task was to develop an FL-based model that will be used as a detection model in anomaly detectors in each stage. Before starting the training, different scenarios were designed by changing the number of anomaly network flows in the training data. For example, 20% of the data flows were taken from the available anomaly network flows, and 80% were taken from normal network flows. The training data composition was changed similarly in both the stage 1 and stage 2 training data. Anomaly data percentages were kept at 6 values: 20%, 30%, 40%, 50%, 60%, and 70%. These different sets of training data were sent through the proposed model, and model parameters were generated and saved for use during the testing phase.

FL was used to train the models used in stages one and two. Devices (also known as clients) have a small data set and train models for a number of local rounds, or epochs. The parameters are then shared by each client with a central server, which aggregates the model parameters and redistributes them to the client. Clients train the model again, starting with the model created using the parameters sent by the central server, and they share the parameters with the central server once more. A simulation round is defined as the process of generating model parameters from clients, sharing them with the central server, and aggregating them. For each case, hundreds of such rounds were used.

The ML model used for stage 1 has four layers, two of which are hidden. The number of data samples was divided into ten groups independently and identically for training model 1. After receiving a set of data, each client trains a model, and the parameters generated are shared with a central server. For training, each client employs five epochs. The central server aggregates the parameters, computes the average, and sends it back to the devices. For each case, the model parameters for 100 such rounds were recorded.

In stage 2, a similar methodology was used to train the model. The second-stage model consists of six layers, four of which are hidden, and 99,200 samples are used for training. As a result, each client receives approximately 9920 data flows that are used for training with five epochs to generate model parameters. It will then be shared with the central server for aggregation in the following step. For each case, one hundred such rounds are used, which is generated by changing the anomaly percentage of the training data.

4.2.2 Model testing

A set of testing data randomly picked from the data pool available was used to test the model parameter generated. First, the model parameters generated for each case were given to the stage 1 and stage 2 models. Then the testing data was classified as anomalous or not by the stage 1 detector. The parameters that were detected as normal flow were again classified by the stage 2 model. After sending through both models, the accuracies for each model were recorded. Stage 1 accuracy is simply the number of false positives and false negatives. The accuracy after the 2nd stage consists of false positives for the location 1 model, which was not analyzed further.

5 RESULTS

The results generated from the simulation process described above are included and analyzed in this section. It can be considered under two categories: overall system accuracy and improvement due to the use of two stages. First, as described in the simulation section, the anomaly percentage of stage 1 and stage 2 was fixed at a certain percentage, and stage 1 and overall accuracy were recorded. Then, the outcome of the 100 rounds in each case is included and followed by a description. Finally, a summary of the data is presented.

First, the anomaly data ratio is fixed at 20%, and the stage 2 ratio is changed from 20% to 70%. Figures 9, 10, 11, 12,13, and 14 show how the stage 1 and stage 2 accuracy change with the stage 2 anomaly percentage. It should be noted that stage 2 accuracy is the same as overall system accuracy. Upon inspecting the figures, it is clear that in each case, overall system accuracy reaches a value of around 90% after ten or more rounds. The value remains stable while fluctuating only by an insignificant amount from the mean value. Stage 1 accuracy remains around 0.85 for all cases, but it shows a slight improvement within 10 and 100 rounds. Also, both stage 2 accuracy and the gap between stage 1 and stage 2 accuracy have increased as the anomaly percentage in stage 2 training has increased.

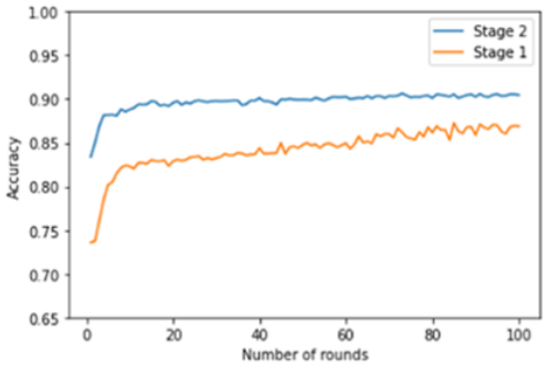

Figure 9 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 20

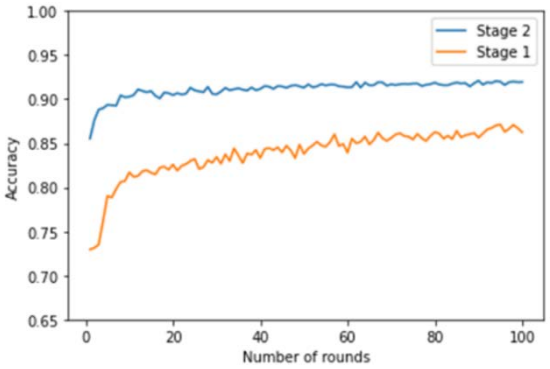

Figure 10 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 30

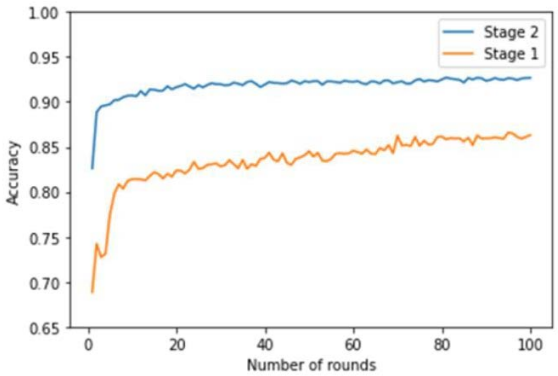

Figure 11 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 40

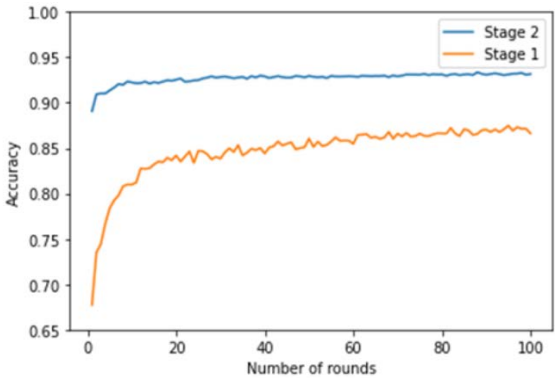

Figure 12 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 50

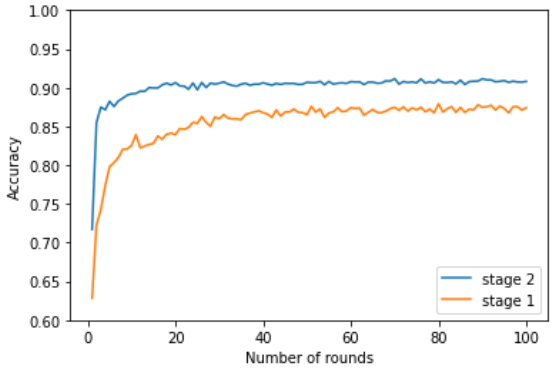

Figure 11 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 60

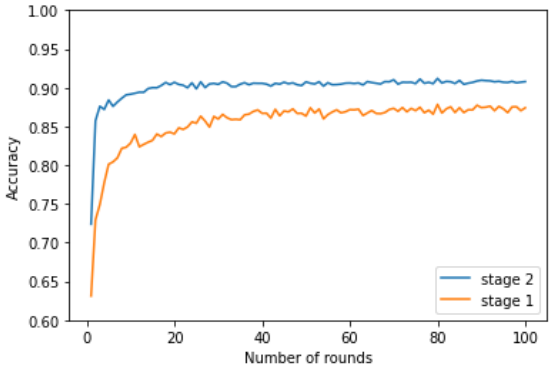

Figure 12 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 20, Stage 2: 70

Next, the anomaly ratio in stage 1 is set at 30%, and in stage 2, the anomaly ratio is varied from 20% to 70%. Figures 15, 16, 17, 18, 19, and 20 show how the overall system accuracy varies for 100 rounds. The proposed system behavior is similar to the previous case, where the stage 1 anomaly ratio is 20%. Both stage 1 and stage 2 accuracy values show only a small improvement after the initial rounds. Stage 1 shows very small improvement within 100 rounds. Overall system accuracy has improved when the stage 2 anomaly ratio is varied from 20% to 70%.

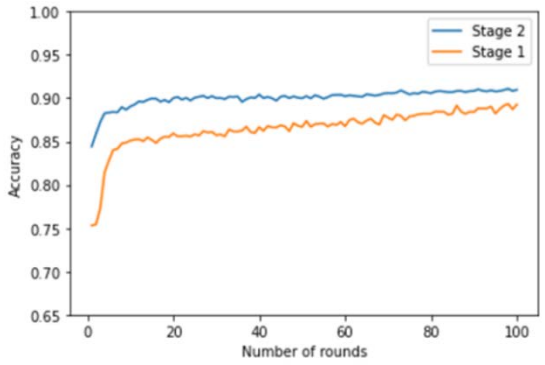

Figure 15 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 20

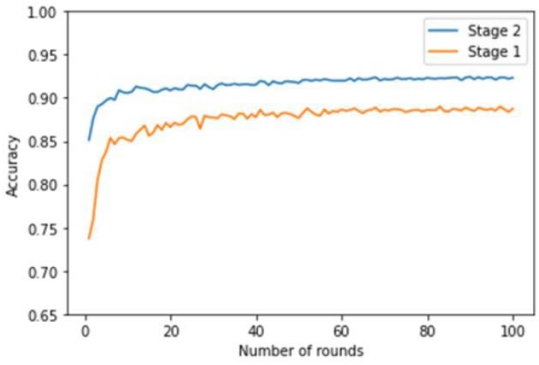

Figure 16 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 30

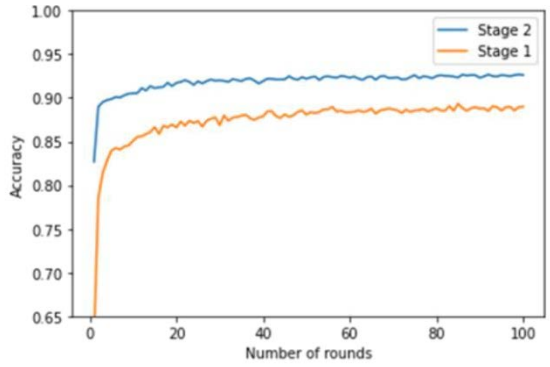

Figure 17 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 40

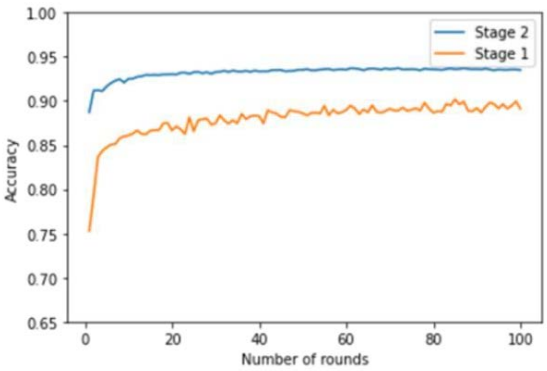

Figure 18 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 50

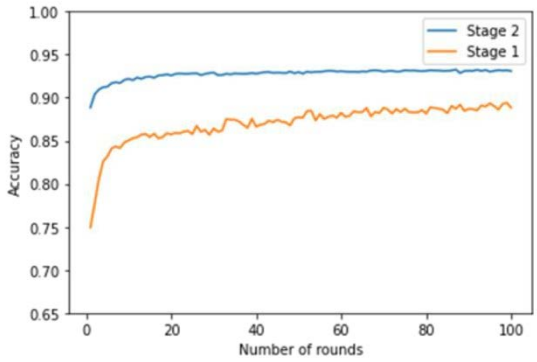

Figure 13 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 60

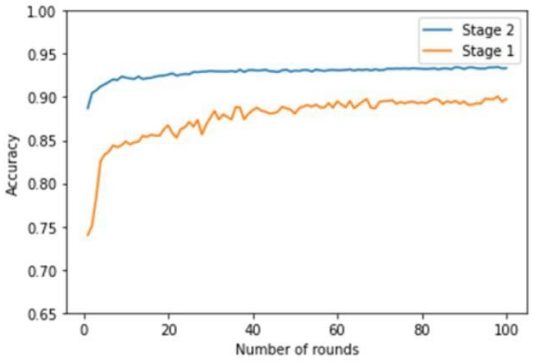

Figure 14 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 30, Stage 2: 70

Figures 21, 22, 23, 24, 25, and 26 demonstrate the system's behavior when the stage 1 anomaly ratio is fixed at 40% and the stage 2 anomaly ratio is varied from 20% to 70%. As per the figures, stage 1 accuracy reaches 90% after 20 rounds. There is a significant reduction in the gap between stage 1 and stage 2 accuracy. When there are more than 20 rounds, the total accuracy stays above 90%, even though the number of rounds has gone up.

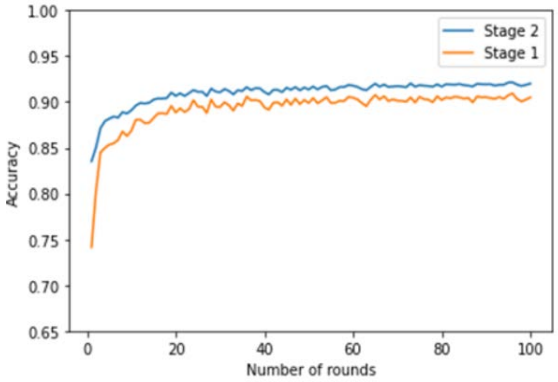

Figure 15 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 20

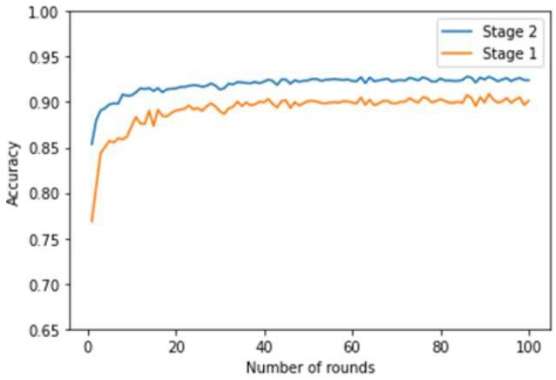

Figure 22 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 30

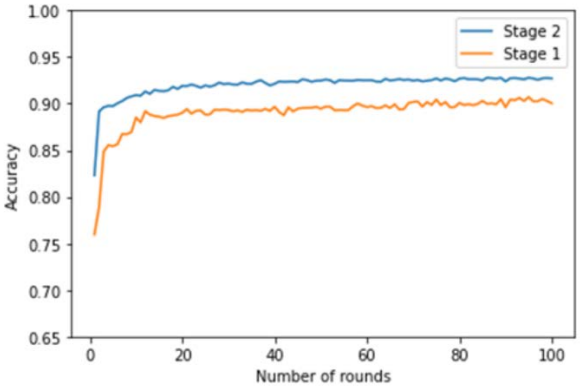

Figure 16 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 40

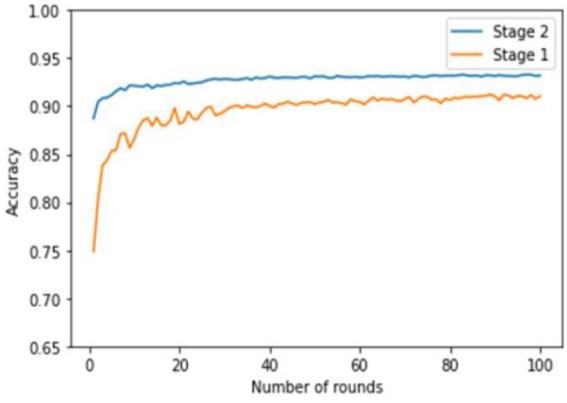

Figure 24 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 50

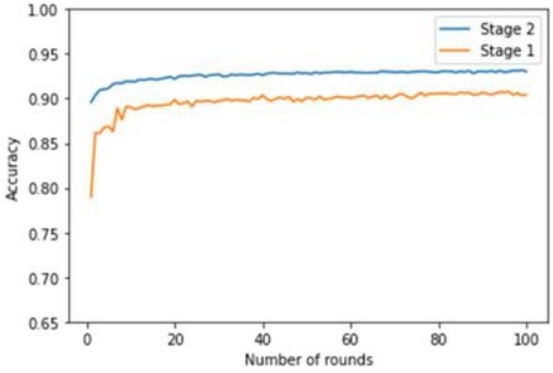

Figure 25 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 60

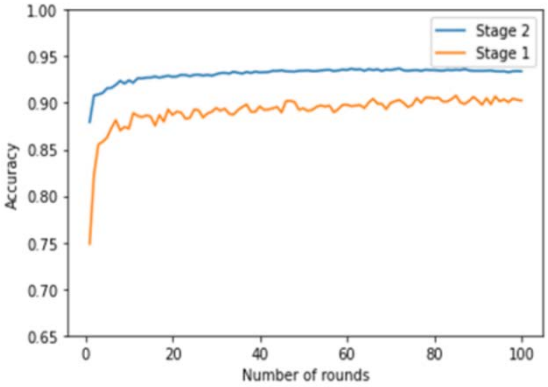

Figure 26 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 40, Stage 2: 70

The anomaly percentage in the training data for stage 1 is set to 50% for the next scenario, and the anomaly ratio of training data in stage 2 is varied. As per the set of figures from Figures 27 to 32, there is a significant reduction in the gap between stage 1 and stage 2 accuracy. However, the accuracy of stage 1 and stage 2 remains higher than 90% for almost all cases when the number of rounds is greater than 20.

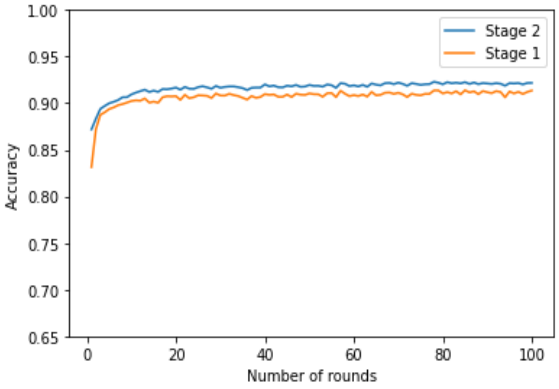

Figure 17 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 20

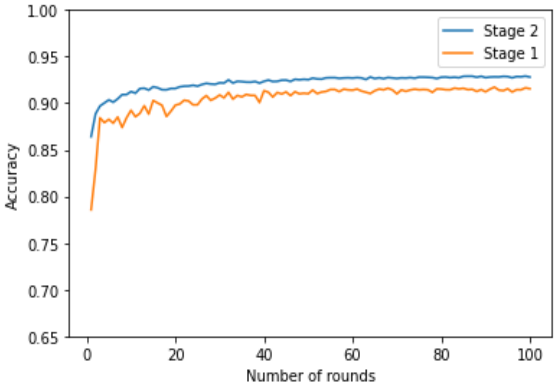

Figure 18 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 30

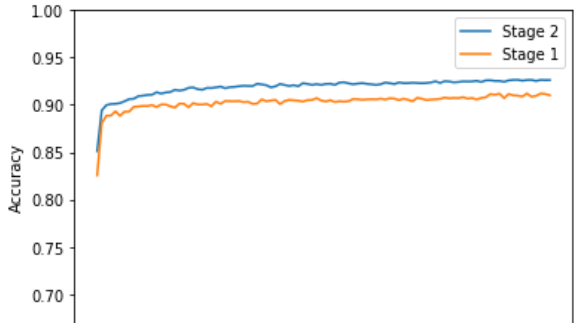

Figure 29 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 40

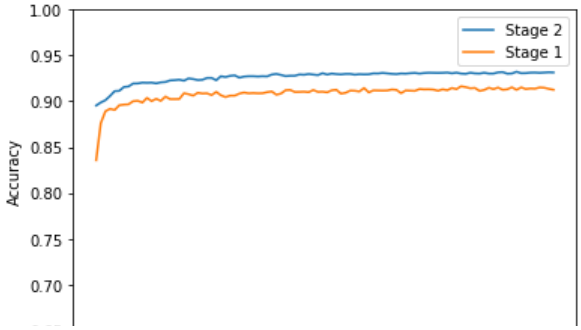

Figure 30 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 50

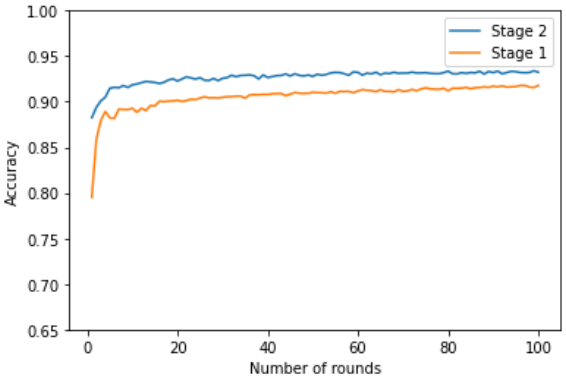

Figure 31 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 60

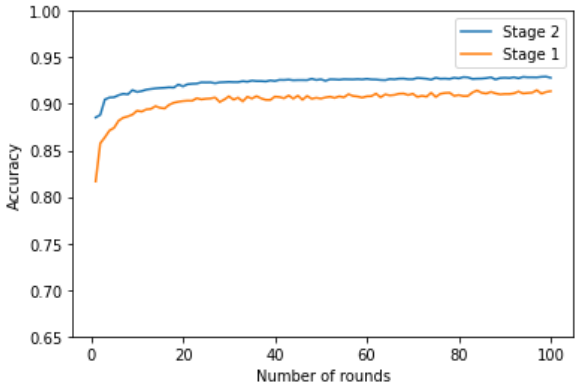

Figure 32 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 50, Stage 2: 70

Figures 33, 34, 35, 36, 37, and 38 are included here to show the system's behavior when the stage 1 anomaly data ratio is set to 60%, and the stage 2 anomaly ratio is varied from 20% to 70%. The accuracy of all cases has reached a value closer to 90%, and the gap between stage 1 and stage 2 accuracy has further reduced. Also, for all cases, the number of rounds taken to reach a stable value seems to be lower than in the previous cases. Also, almost all the graphs seem to be identical.

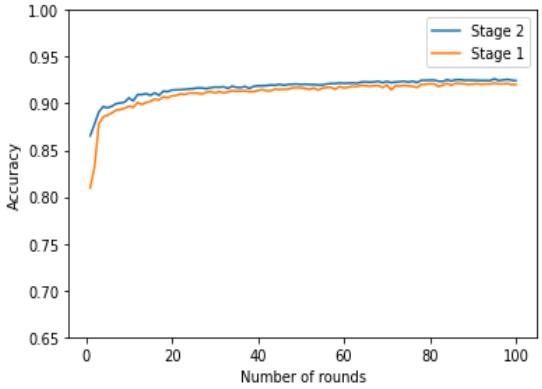

Figure 33 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2: 20

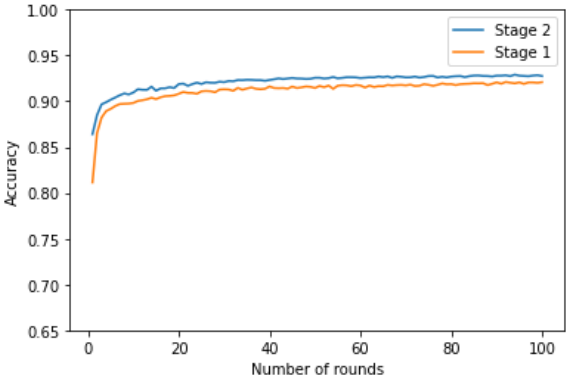

Figure 34 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2: 30

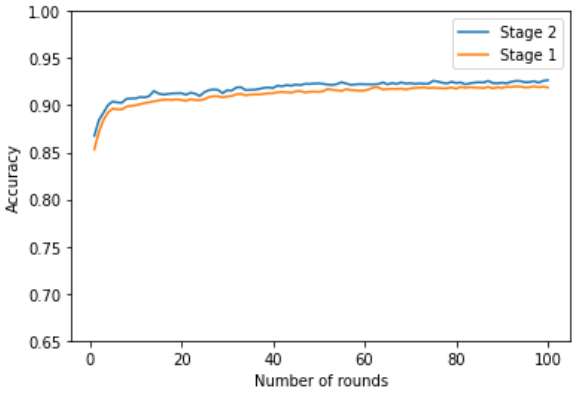

Figure 35 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2: 40

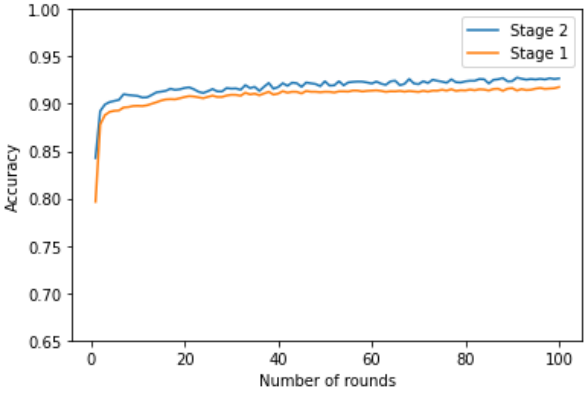

Figure 36 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2: 50

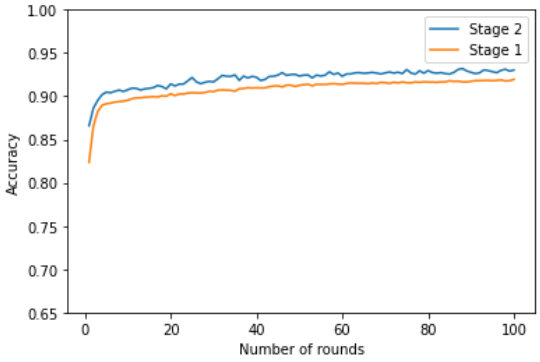

Figure 20 Accuracy vs Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2: 60

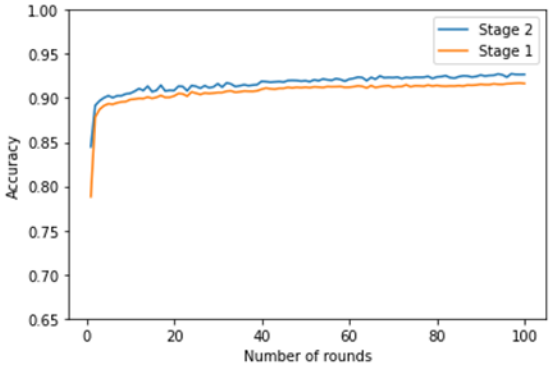

Figure 19 Accuracy vs Number of rounds, Anomaly ratio of Stage 1: 60, Stage 2:70

The anomaly ratio of the training data was changed to 70% at the final stage. Similar to the above instances, the system response when the stage 2 anomaly ratio is changed from 20% to 70% is observed. The most significant outcome of this scenario is that the stage 1 accuracy exceeds or overlaps with the overall accuracy in certain cases. In all other cases, there is no significant difference between stage 1 accuracy and stage 2 accuracy. However, it should be noted that the overall system accuracy of all situations remains as low as 90%, and it does not improve with the number of rounds after the 20th.

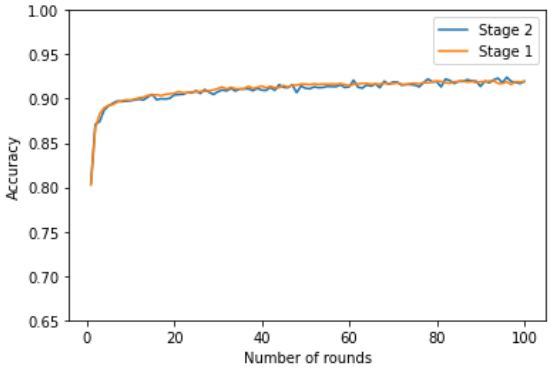

Figure 39 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 20

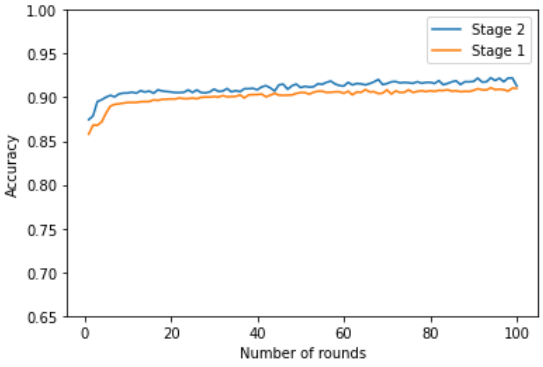

Figure 40 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 30

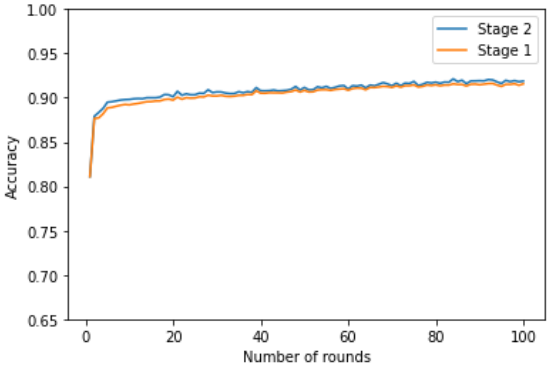

Figure 22 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 40

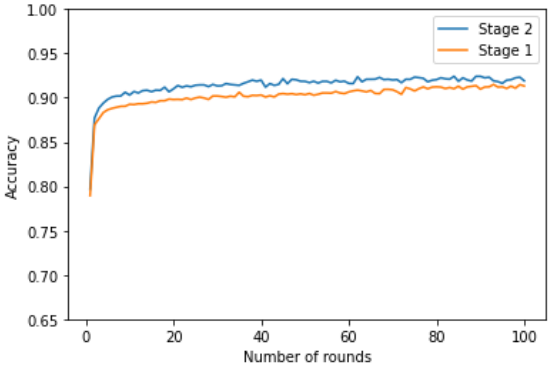

Figure 21 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 50

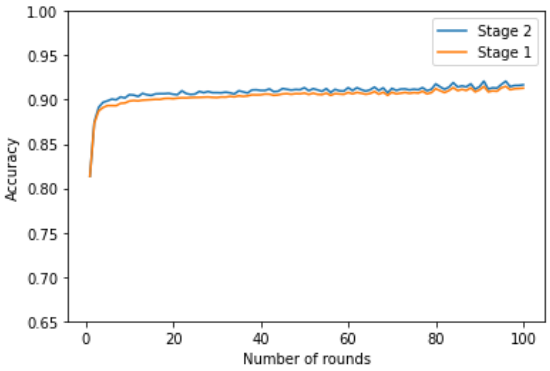

Figure 23 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 60

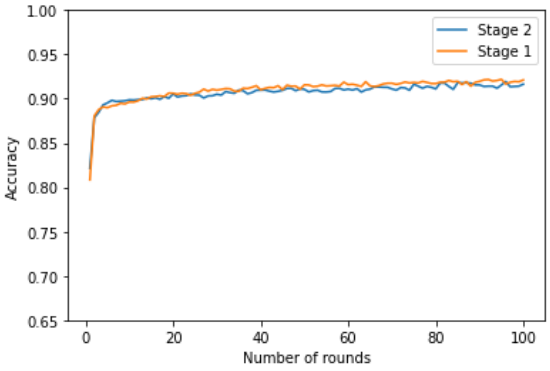

Figure 24 Accuracy vs. Number of rounds, Anomaly ratio of Stage 1: 70, Stage 2: 70

Figure 9 to 44 shows how the proposed system accuracy changes with the anomaly data ratio in training data in stages 1 and 2. The results of simulations can be displayed in a variety of ways, allowing different aspects to be evaluated. The rest of the section looks at the results with the help of different graphs.

Figure 25 shows the overall accuracy of the system when the stage 2 accuracy is varied from 20% to 70% in one figure. All the cases presented in figures 45 and 46 are presented in table 6. It is drawn for the case where the stage 1 anomaly data percentage is 20%. The highest accuracy stage is recorded when the stage 2 percentage is 60%. Both have the second-highest accuracy stage. When the percentage is 50% and 70% remains equal. The lowest accuracy is recorded when the stage 2 anomaly percentage is 20%.

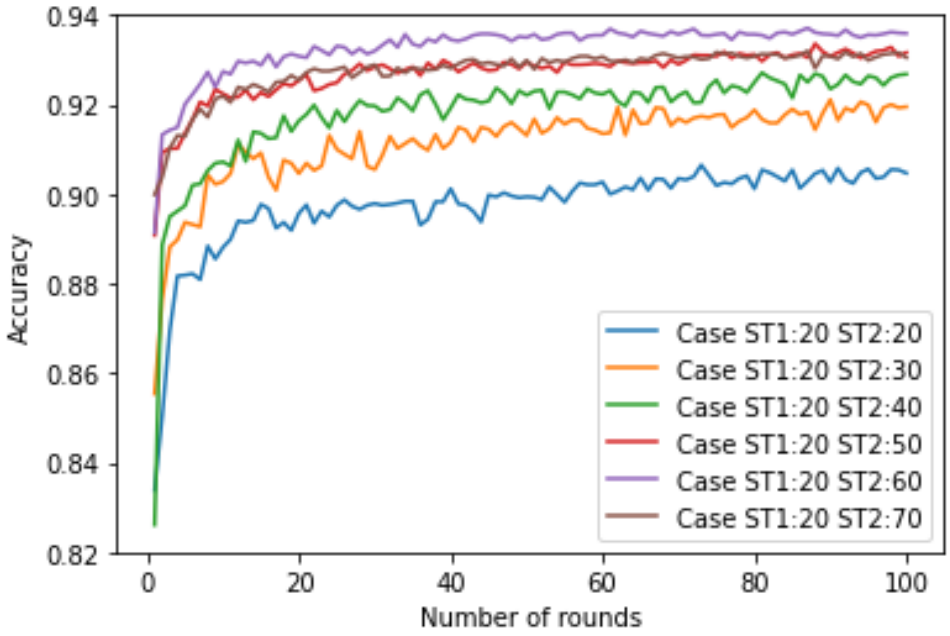

Figure 25 Overall Testing accuracy against anomaly data percentage in training data stage 2

It is also possible to evaluate the results by keeping the anomaly data rate in stage 2 fixed and by varying the anomaly data percentage in stage 1 data as well. However, as per the data presented in the above figures, the accuracy values remain the same when the number of rounds reaches a value of around twenty. Therefore, to investigate the overall performances, the accuracy recorded after 100 rounds is included in Table 3. Table 3 is colored based on the accuracy levels as presented in Table 6. The highest rates are indicated in green, and the lowest rates are indicated in orange.

When the anomaly percentage in stage 2 is 20% or 30%, accuracy has gradually increased with the anomaly percentage in the stage 1 training data. The highest values were recorded at around 50% and 60%. However, it has dropped slightly when 70% of the training data for stage 1 are anomalies. In the case where the stage 2 percentage is 40%, the rate fluctuates around 92.6% for all stage 1 percentages except for 70%. However, it should be noted that the accuracy rate remains relatively high in comparison with the two cases discussed above. When the anomaly rate is increased to 50% in stage 2, it is visible that accuracy has the highest value when the stage 1 value is set at 30%. Still, the accuracy has gradually decreased when the stage 1 percentage is increased from 30% to 70%. When the stage 2 percentage is 60% or 70%, the highest accuracy is shown when the percentage of anomaly data in stage 1 is 20%. Figure 44 shows how the overall accuracy has behaved during all 100 rounds for the case where stage 2 percentage is fixed at 60%, and stage 2 anomaly percentage is varied from 20% to 70%. Similar to the pattern observed in the table, the highest percentage of accuracy is observed when stage 1 percentage is around 30% for all 100 rounds.

When inspecting the pattern along with the columns of Table 6, it is visible that accuracy gradually increases for most of the cases, reaching a peak when stage 2 percentage is around 60%. The values fluctuate when the stage 1 percentage is 70%. Figure 43 shows how the accuracy changes when the stage 2 anomaly percentage is varied from 20% to 70% and the stage 1 percentage is kept at 20%. The same pattern observed in the table is displayed in the graph throughout 100 rounds. In addition to the table, to show how the stage 1 accuracy varies when the stage 2 anomaly ratio is fixed, see Figure 44.

Table 6 Overall Testing accuracy against anomaly data percentage in training data stage 1 and stage 2

Anomaly data percentage in training data-Stage 2	**Anomaly data percentage in training data- stage 1**					
	20%	30%	40%	50%	60%	70%
20%	90.5%	91.0%	92.0%	92.2%	92.4%	91.9%
30%	91.9%	92.3%	92.4%	92.8%	92.8%	91.3%
40%	92.7%	92.6%	92.7%	92.6%	92.7%	91.9%
50%	93.2%	93.4%	93.2%	93.1%	92.7%	91.9%
60%	93.6%	93.5%	93.4%	93.2%	93.0%	91.7%
70%	93.0%	93.1%	93.0%	92.8%	92.7%	91.6%

Table 7 Legend for table 4

Colour	**Accuracy percentage**
	93.1-:94.0%
	92.1-93.0%
	90.0-92.0%

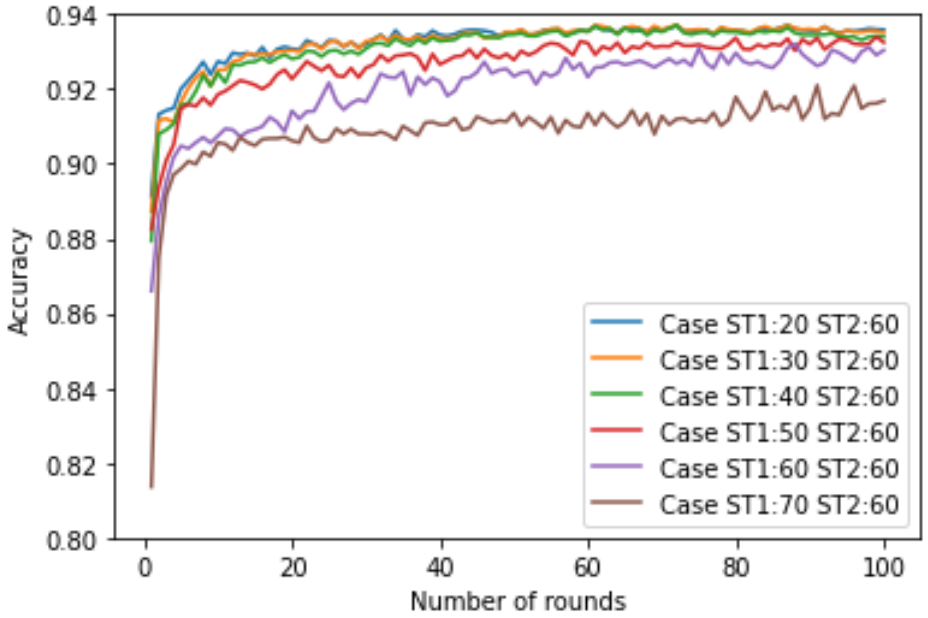

Figure 26 Overall Testing accuracy against anomaly data percentage in training data stage 1

Table 8 Set of cases shown in Figures 45 and 46

Case	**Anomaly data percentage in training data**	
	Stage 1	**Stage 2**
Case ST1:20 ST2:60	20	60
Case ST1:30 ST2:60	30	60
Case ST1:40 ST2:60	40	60
Case ST1:50 ST2:60	50	60
Case ST1:60 ST2:60	60	60
Case ST1:70 ST2:60	70	60
Case ST1:20 ST2:20	20	20
Case ST1:20 ST2:30	20	30
Case ST1:20 ST2:40	20	40
Case ST1:20 ST2:50	20	50
Case ST1:20 ST2:60	20	60
Case ST1:20 ST2:70	20	70

The measure of effectiveness of the proposed system can be measured by the improvement in accuracy between stage 1 and stage 2. Table 9 shows that overall system accuracy has increased after the two stages. The highest increment from stage 1 to stage 2 is recorded when the anomaly percentage in stage 1 is 20%. Even though there is an increment for all the cases except for one, the amount gradually reduces when the stage 1 anomaly percentage increases from 20% to 70%. A significantly higher increment is observed when the stage 1 percentage is as low as 20%.

Table 9 Overall Testing accuracy increase after stage 2 against anomaly data percentage in training data stage 1 and stage 2

Anomaly data percentage in training data-Stage 2	**Anomaly data percentage in training data- stage 1**					
	20%	30%	40%	50%	60%	70%
20%	3.6%	1.7%	1.5%	0.8%	0.4%	0%
30%	5.7%	3.6%	2.3%	1.3%	0.7%	0.3%
40%	6.4%	3.6%	2.7%	1.6%	0.8%	0.3%
50%	6.5%	3.6%	2.2%	1.9%	0.9%	0.6%
60%	5.5%	4.4%	3.1%	1.5%	1.1%	0.4%
70%	6.3%	4.3%	2.6%	1.5%	0.1%	-0.5%

Table 10 Legend for table 6

Colour	**Increment percentage**
	5.0-7.0%
	3.0-5.0%
	1.1-3.0%
	-1.0-1.0%

The testing data was drawn at random from the set of data, with 50–60% of the data being anomalies. As a result, naturally tested data will follow the same distribution. According to the data, accuracy improves when the training data for stage 1 is the same as the testing data. When stage 2 percent is around 60%, the accuracy is at its best. In stage 2, the number of training samples is 99,200, which is ten times the number of data samples in stage 1. When 60% of the anomalies are taken, the majority of the anomalies are included in the training data, allowing the system to detect the greatest number of anomalies accurately. When the anomaly percentage reaches 70%, there is insufficient normal data in the training data. As a result, accuracy suffers.

In the proposed system, only the network flow that is marked as "normal flow" is sent to the 2nd stage. However, it is also possible to analyze the anomalies in the 2nd stage as well. In this case, first, all the data is filtered by stage 1, and then all anomalies are directed to the 2nd stage for filtering. The results similar to table 6 and table 9 were obtained in order to check the feasibility of a similar scenario and are presented in table 11 and table 13. Based on tables 12 and 14, coloring cells for tables 11 and 13 have taken place.

Table 11 Overall Testing accuracy against anomaly data percentage in training data stage 1 and stage 2- Anomalies sent to 2nd stage

Anomaly data percentage in training data-Stage 2	**Anomaly data percentage in training data- stage 1**					
	20%	30%	40%	50%	60%	70%
20%	83.2%	85.6%	87.6%	88.4%	88.7%	89.0%
30%	83.1%	85.8%	88.0%	89.3%	89.7%	90.3%
40%	83.7%	87.2%	90.1%	90.5%	91.0%	91.8%
50%	83.8%	87.5%	89.3%	91.3%	91.5%	92.4%
60%	83.9%	86.7%	90.8%	91.8%	92.4%	93%
70%	83.2%	88.0%	90.0%	92.0%	92.4%	93.1%

Table 12 Legend for table 10

Colour	Accuracy percentage
	90.1-:93.5%
	86.5-90.0%
	96.4-83.0%

Table 13 Overall Testing accuracy increase after stage 2 against anomaly data percentage in training data stage 1 and stage 2 - Anomalies sent to 2nd stage

Anomaly data percentage in training data-Stage 2	**Anomaly data percentage in training data- stage 1**					
	20%	30%	40%	50%	60%	70%
20%	-0.3%	-0.6%	-2.1%	-0.3%	-2.4%	-1.3%
30%	-0.3%	-0.4%	-1.3%	-1.8%	-1.3%	-0.3%
40%	-0.1%	-0.2%	-0.3%	-0.2%	0.2%	1.4%
50%	0%	0%	-0.3%	0.4%	0.4%	1.6%
60%	0%	0.2%	0.2%	0.8%	1.7%	2.3%
70%	0.1%	0.1%	0.2%	0.4%	1.1%	2.8%

Table 14 Legend for table 12

Colour	Increment percentage
	3.0-1.6%
	1.5-0%
	0 - (-1.5)%
	(-1.6)- (-3.0)%

As per table 8, when the anomaly percentage in training data takes a higher value, the accuracy has significantly improved. When the anomaly percentage in both stages 1 and 2 is 70%, the highest accuracy is recorded at 93.1 percent. A similar observation can be made when considering Table 9 as well. A reasonable improvement can be observed when the stage 2 anomaly percentage is 60% or above. However, compared with the case where normal data is evaluated at the 2nd stage, only half of the cases record positive improvement. As a result, evaluating the anomalies at the second stage may be inefficient.

Model training and model testing are the two major components of the simulation process. After defining a set of cases, model parameters were generated using a set of pre-selected training data. The testing data was then passed through the models with the generated model parameters, which are used to classify network flows as anomalous or not. It is possible to send both anomalous and normal data to the second stage. In the simulation, both of these approaches were demonstrated. According to the results, it is more efficient to analyze the data marked as normal traffic in stage 2 to achieve higher accuracy. The use of training data that includes all possible scenarios in its database is critical for the success of the system.

6 DISCUSSION

The purpose of this section is to critically analyze the results obtained in this thesis. First, the work carried out is critically analyzed with similar work. This is followed by an examination of the thesis's objectives and the extent to which they were met after the thesis's completion. The final section focuses on the possible future directions of the thesis.

6.1 Comparative analysis with Similar Work

This thesis presents an anomaly detector for the ZSM architecture developed based on FL. Even though network automation has been discussed for a while, the ZSM architecture is one of the very first architectures proposed. It is designed with wireless networks in mind, including 5G and beyond 5G networks. Therefore, only a few security algorithms designed for the ZSM architecture can be found in the current state-of-the-art.

FL has been used in many applications on the wireless network, especially in IoT-related applications. A comprehensive list of such applications is available in [42]. However, most of the applications are developed to identify the devices that are affected by a third party. Secure industrial IoT things, secure internet of drones, and secure internet of health care things are some of the applications available in the literature. Most of these applications use servers at the edge as aggregation servers and are not embedded in the architecture of the network, nor do they consider the ML-enabled networks as their architecture. However, ML, or AI, is an integral part of the network in the ZSM architecture, which allows using ML-based applications at the core of the network as well. The proposed system is embedded in the ZSM architecture, and aggregation servers can be found inside the network as well.

Two algorithms presented in [47] and [49] are based on wireless networks. The first mechanism [47] demonstrates an FL-based end-edge cloud corporation for network security. The attack detection takes place at the end devices, the edge, and the cloud. As only the aggregation server is at the higher level of the network, this work is different from the proposed setup. The self-adaptive algorithm proposed in [49] uses the ETSI NFV architecture. If an anomaly is detected, they propose using a virtual IDS set up to specifically analyze the network flow that is considered an anomaly. However, FL is not used for developing the system.

There are relatively more anomaly detectors that use ML than FL for detection. The used algorithms include but are not limited to neural networks, support vector mechanisms, decision trees, and recurrent neural networks. The proposed system uses a simple neural network, However, it is easily adaptable to network models that are available in the literature. The most suitable network model can vary with the application and data set and other practical constraints that could affect the deployment. It is possible to measure the accuracy and other outcomes based on the network model. This will be further discussed in future research directions.

6.2 Evaluation on Meeting the Thesis Objectives

The main objective of the thesis is to develop an FL-based anomaly detector for the ZSM architecture, which is to be used along with the ZSM architecture for 5G and beyond 5G networks. The proposed model incorporates the ZSM architecture and makes use of the components proposed in the architecture. The thesis also proposes how the anomaly detector is updated when the detector fails to identify any possible threats, thus incorporating a closed-loop mechanism along with AI, which are two major enablers in the ZSM architecture.

The performance of the proposed system can be measured by two values: the system accuracy and the accuracy improvement from stage 1 to stage 2. The highest recorded system accuracy is 93.6%, and the value remains above 90% for all cases. Most of the proposed algorithms presented in the literature review were able to achieve an accuracy higher than 90%. Therefore, the proposed system accuracy can be considered adequate compared with the previous work. The maximum improvement from stage 1 to stage 2 is 6.5%. There is a positive improvement in all cases except the case where the anomaly data percentage in both stage 1 and stage 2 is 70%. Therefore, it can be said that the proposed system performance is within the expected range.

According to the observations, training data should adequately cover all possible scenarios encountered by the detector. The proposed method allows for a larger database at the second stage, allowing for coverage of all possible scenarios. As a result, even if an abnormal flow is marked as normal data and sent to the second stage, the second stage can detect it. As a result, the two-stage model improves detector accuracy while maintaining data privacy and communication efficiency. Aside from the benefits derived from using the FL model, it also allows for the accommodation of unavoidable resource constraints.

6.3 Future Research Directions

This thesis proposes an anomaly detector for the ZSM network architecture, which is being developed for 5G and beyond 5G networks. The development of 6G networks is still in its early stages, and it is possible that it will change over time. AI for everything is one of the technological research areas being investigated by 6G. As a result, the ZSM architecture can be modified to work with both 6G and 5G networks. It is possible, however, that it will be modified in response to new research directions. To account for this, the proposed anomaly detector will need to be modified.

The UNSW-NB 15 is used as the data set for training and testing models. The proposed system can be further modified by using a different data set. The proposed system is intended for 5G and beyond 5G networks. However, data sets related to wireless communication that can be used for AI-related applications are rare. One research direction is to develop a data set by utilizing 5G and beyond 5G network data and to use the new data set along with the proposed model. The ZSM architecture divides the network into domains. RAN and core are two major domains that can be used. If a suitable data set can be found, it is possible to test the solution and find the functionality in one domain specifically. For example, a data set that is based on the data collected from the RAN network can be utilized to test the functionality in the RAN domain.

Further modifications that can be made to the proposed system include changing aggregation techniques and changing model types. “FedSGD” is a popular algorithm that can be used instead of averaging for aggregation. There are many aggregation techniques proposed in the literature for various applications. Therefore, it is possible to use various aggregation techniques to find the best aggregation technique. Apart from the aggregation technique, it is also possible to change the network model used for FL. LSTM, GRU, and RNN models have been frequently used in other proposed algorithms and can be adapted to the system proposed in this thesis too.

As highlighted in the literature review, there can be many threats arising from the use of AI for network applications. It is also important to pay attention to both generic and new threats that might affect the network's security. However, the proposed model did not specifically concentrate on the protection of the network models from possible threats such as

poisoning and inference attacks. It is also possible to conduct research on practical implications arising from the research, such as quantization, done before sending the model parameters.

7 SUMMARY

Network automation is a critical component in future networks that must fulfill unprecedented demands. The ZSM architecture is one such architecture designed to meet the needs of network automation. Multi-domain network architecture, closed-loop, AI, and data analytics services embedded in the architecture are pivotal for network automation. Networks can be protected from attacks by using AI and closed loops. They can also be used for tasks like optimizing network resources.

Based on technological or business requirements, the ZSM architecture is divided into multiple domains. Edge, RAN, and Core are a few examples of possible multi-domains. End-to-end management domains oversee these domains. Each domain includes functions such as domain analytics, and domain data services. The closed-loop is a critical enabler for network automation. Orient, decide, act, observe, and knowledge is the proposed closed loops for the ZSM architecture. The network's security mechanism is built on the same closed-loops other components, such as AI. Security threats to the ZSM architecture include those resulting from the use of new enablers such as API, SDN, NFV, and AI/ML, as well as previous network attacks such as DoS.

FL is a relatively new field in ML that provides advantages such as privacy protection. FL models are developed on end devices with locally available data, as opposed to the centralized training used in other ML models. The model trained, on the other hand, is determined by a central server, also known as an aggregation server. The model parameters are generated based on the model sent by the end devices. The aggregation criterion can be determined by the central server. Model parameters, for example, can be weighted based on the number of samples found in each client. FL is vulnerable to attacks such as adversarial attacks and member poisoning. There are already mechanisms in the literature to prevent this from happening. FL is also employed in a variety of applications for anomaly detection. However, because of the large number of devices, the majority of it is applied to IoT applications.

The proposed system makes use of FL as an enabler and a hierarchical detection mechanism. It is made up of two stages, each with its own set of detectors. The first detector is straightforward and close to the end user. The central server in each domain serves these detectors. The network flow that passes through a single detector will come into contact with another detector in stage 2 that is located in the same domain. The detector at the top of the hierarchy, on the other hand, has a more complex model and a larger database to accommodate more data. Aside from the detection mechanism, a training mechanism is proposed because unprecedented events may occur in 5G and beyond networks.

The proposed system was then run through a simulation using the Jupiter notebook and TensorFlow federated libraries. For the simulation, the UNSW-NB 15 data set was used. For this purpose, 42 features describing nine types of attacks were used. The testing consists of 10,000 samples drawn at random from the data set. The training set was then created based on the required anomaly percentage for each stage for the presented test cases. The proposed system was then run through a simulation using the Jupiter notebook and TensorFlow federated libraries. For the simulation, the UNSW-NB 15 data set was used. For this purpose, 42 features describing nine types of attacks were used. The testing consists of 10,000 samples drawn at random from the data set. The training set was then created based on the required anomaly percentage for each stage for the presented test cases.

According to the findings, the network's accuracy has improved following the two-stage mechanism. However, as shown in the results section, there was a greater improvement in accuracy in certain cases. The overall system accuracy followed a similar pattern. The highest

recorded accuracy is 93.6 percent, with anomaly percentages of 20% and 60%, respectively, in stages 1 and 2. Upon inspection of the results, it is clear that when the test data composition is similar to the training data, higher accuracy can be obtained. If unknown threats or network flows are detected, the closed-loop mechanism ensures that the relevant flows are added to the database and that models are updated accordingly. It is also important to include all the possible network flow types in the training data. Due to limitations such as resource limitations, it can be difficult to include a larger database for each detector. However, the proposed system contains a two-stage system, and the database of the stage 2 detector is capable of storing more network flows. As a result, it can successfully deal with resource constraints.

It has been demonstrated that the proposed detector can successfully protect the network from a variety of threats. The closed-loop mechanism, in conjunction with the anomaly detector, ensures that the network is safe from unprecedented attacks. Future networks, including 6G, will include AI-related network automation mechanisms. As a result, the proposed mechanism, along with the ZSM architecture, can be successfully adapted to such networks while adhering to resource constraints.

8 REFERENCES

[1] 5G architecture 3gpp. [Online]. Available: https://www.viavisolutions.com/en-uk/5g-architecture

[2] ETSI, "Zero-touch Network and Service Management (ZSM)," ETSI GS ZSM 002 - Reference Architecture, Aug 2019

[3] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "AI and 6G Security: Opportunities and Challenges," 2021 IEEE Joint European Conference on Networks and Communications (EuCNC) 6G Summit, 2021, pp. 1–6, 2019.

[4] C. Benzaid and T. Taleb, "ZSM security: Threat Surface and best Practices," IEEE Network, vol. 34, no. 3, pp. 124–133, 2020.

[5] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated Learning for 6G Communications: Challenges, Methods, and Future directions," China Communications, vol. 17, no. 9, pp. 105–118, 2020.

[6] The etsi zsm framework reference architecture. [Online]. Available: https://content.comms.euromoneyplc.com/rs/376-KVV-177/ images/20200917 ZSMForum ZSM reference architecture Uwe Rauschenbach.pdf

[7] C. Benzaid, P. Alemany, D. Ayed, G. Chollon, M. Christopoulou, G. Gur, V. Lefebvre, E. M. de Oca, R. Mu ¨ noz, J. Ortiz, ˜ A. Pastor, R. Sanchez-Iborra, T. Taleb, R. Vilalta, and G. Xilouris, "White paper intelligent security architecture for 5g and beyond networks," 2020. [Online]. Available: https://www.inspire-5gplus.eu/ white-paper-on-intelligent-security-architecture-for-5g-and-beyond-networks/

[8] M. Xie, P. H. Gomes, J. Harmatos, and J. Ordonez-Lucena, "Collaborated closed loops for autonomous end-to-end service management in 5g," in 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2020, pp. 64–70.

[9] I. Vaishnavi and L. Ciavaglia, "Challenges towards automation of live telco network management: Closed control loops," in 2020 16th International Conference on Network and Service Management (CNSM), 2020, pp. 1–5.

[10] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos, "Privacy and security issues in deep learning: A survey," IEEE Access, vol. 9, pp. 4566–4593, 2021. Open Networking Foundation,, "Threat Analysis for the SDN Architecture," Threat Analysis for the SDN Architecture, July 2016.

[11] ETSI GS NFV-SEC 003, "Network Functions Virtualisation (NFV);NFV Security; Security and Trust Guidance," Network Functions Virtualisation (NFV);NFV Security; Security and Trust Guidance, Dec 2014.

[12] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," IEEE/ACM Transactions on Networking, vol. 25, no. 6, pp. 3514–3530, 2017.

[13] K. Elleithy, D. Blagovic, W. Cheng, and P. Sideleau, "Denial of service attack techniques: Analysis, implementation and comparison," Journal of Systemics, Cybernetics and Informatics, vol. 3, pp. 66–71, 01 2006.

[14] What is a denial of service attack (dos) ? [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/ what-is-a-denial-of-service-attack-dos

[15] N. Moustafa and J. Slay, "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), 2015, pp. 1–6.

[16] O. Nasser, S. AlThuhli, M. Mohammed, R. AlMamari, and F. Hajamohideen, "An investigation of backdoors implication to avoid regional security impediment," in 2015 Global Conference on Communication Technologies (GCCT), 2015, pp. 409–412.
[17] Exploits. [Online]. Available: https://www.malwarebytes.com/exploits
[18] Fuzzing explained. [Online]. Available: https://www.contrastsecurity. com/knowledge-hub/glossary/fuzzing
[19] S. H. Sellke, N. B. Shroff, and S. Bagchi, "Modeling and automated containment of worms," IEEE Transactions on Dependable and Secure Computing, vol. 5, no. 2, pp. 71–86, 2008.
[20] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," CoRR, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/ 1602.05629
[21] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," CoRR, vol. abs/1908.07873, 2019. [Online]. Available: http://arxiv.org/abs/1908.07873
[22] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascon, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, ´ L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecnˇ y, A. Korolova, F. Koushanfar, S. Koyejo, ´ T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Ozg ¨ ur, R. Pagh, ¨ M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramer, P. Vepakomma, J. Wang, ` L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," CoRR, vol. abs/1912.04977, 2019. [Online]. Available: http://arxiv.org/abs/1912.04977
[23] Recurrent neural networks. [Online]. Available: https://www.ibm.com/ cloud/learn/recurrent-neural-networks
[24] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," CoRR, vol. abs/1607.00148, 2016. [Online]. Available: http://arxiv.org/abs/1607.00148.
[25] Long short term memory networks explanation. [Online]. Available: https://www.geeksforgeeks.org/ long-short-term-memory-networks-explanation/?ref=lbp
[26] Illustrated guide to lstm's and gru's: A step by stepexplanation. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
[27] Z. Qu, L. Su, X. Wang, S. Zheng, X. Song, and X. Song, "A unsupervised learning method of anomaly detection using GRU," in 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), 2018, pp. 685–688
[28] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," IEEE Communications Magazine, vol. 58, no. 6, pp. 46–51, 2020.
[29] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo, "Analyzing federated learning through an adversarial lens," CoRR, vol. abs/1811.12470, 2018. [Online]. Available: http://arxiv.org/abs/1811. 12470
[30] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Inference attacks against collaborative learning," CoRR, vol. abs/1805.04049, 2018. [Online]. Available: http://arxiv.org/abs/1805.04049

[31] C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” CoRR, vol. abs/1903.03934, 2019. [Online]. Available: http://arxiv.org/ abs/1903.03934
[32] Z. Chai, Y. Chen, L. Zhao, Y. Cheng, and H. Rangwala, “Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data,” CoRR, vol. abs/2010.05958, 2020. [Online]. Available: https://arxiv.org/abs/2010.05958
[33] M. van Dijk, N. V. Nguyen, T. N. Nguyen, L. M. Nguyen, Q. Tran-Dinh, and P. H. Nguyen, “Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise,” CoRR, vol. abs/2007.09208, 2020. [Online]. Available: https://arxiv.org/abs/2007.09208
[34] Y. Chen, Y. Ning, and H. Rangwala, “Asynchronous online federated learning for edge devices,” CoRR, vol. abs/1911.02134, 2019. [Online]. Available: http://arxiv.org/abs/1911.02134
[35] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, “Communicationefficient edge AI: algorithms and systems,” CoRR, vol. abs/2002.09668, 2020. [Online]. Available: https://arxiv.org/abs/2002.09668
[36] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” CoRR, vol. abs/1712.01887, 2017. [Online]. Available: http://arxiv.org/abs/1712.01887
[37] Y. Liu, J. Peng, J. Kang, A. M. Iliyasu, D. Niyato, and A. A. A. ElLatif, “A secure federated learning framework for 5g networks,” IEEE Wireless Communications, vol. 27, no. 4, pp. 24–31, 2020.
[38] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, “Reliable federated learning for mobile networks,” CoRR, vol. abs/1910.06837, 2019. [Online]. Available: http://arxiv.org/abs/1910. 06837
[39] Y. Huang, Y. Su, S. Ravi, Z. Song, S. Arora, and K. Li, “Privacy preserving learning via deep net pruning,” CoRR, vol. abs/2003.01876, 2020. [Online]. Available: https://arxiv.org/abs/2003.01876
[40] T. Cao, T. Truong-Huu, H. D. Tran, and K. Tran, “A federated learning framework for privacy-preserving and parallel training,” CoRR, vol. abs/2001.09782, 2020. [Online]. Available: https://arxiv.org/abs/2001. 09782
[41] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, “Federated learning over wireless networks: Optimization model design and analysis,” in IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1387–1395.
[42] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke and L. Shu, "Federated Deep Learning for Cyber Security in the Internet of Things: Concepts, Applications, and Experimental Analysis," in IEEE Access, vol. 9, pp. 138509-138542, 2021, doi: 10.1109/ACCESS.2021.3118642.
[43] X. Wang, S. Garg, H. Lin, J. Hu, G. Kaddoum, M. J. Piran, and M. S. Hossain, “Towards accurate anomaly detection in industrial internet-ofthings using hierarchical federated learning,” IEEE Internet of Things Journal, pp. 1–1, 2021.
[44] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, “D¨Iot: A federated self-learning anomaly detection system for iot,” in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 756–767.
[45] K. Yadav, B. Gupta, C.-H. Hsu, and K. T. Chui, “Unsupervised federated learning based iot intrusion detection,” in 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE), 2021, pp. 298–301.

[46] Y. Liu, N. Kumar, Z. Xiong, W. Y. B. Lim, J. Kang, and D. Niyato, “Communication-efficient federated learning for anomaly detection in industrial internet of things,” in GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1–6.
[47] Y. Wei, S. Zhou, S. Leng, S. Maharjan, and Y. Zhang, “Federated learning empowered end-edge-cloud cooperation for 5g hetnet security,” IEEE Network, vol. 35, no. 2, pp. 88–94, 2021.
[48] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, “Deepfed: Federated deep learning for intrusion detection in industrial cyber–physical systems,” IEEE Transactions on Industrial Informatics, vol. 17, no. 8, pp. 5615–5624, 2021.
[49] L. Fernandez Maimo, A. L. Perales, Gomez, F. J. Garcia Clemente, M. Gil Perez, and G. Martinez Perez, “A self-adaptive deep learning based system for anomaly detection in 5g networks,” IEEE Access, vol. 6, pp. 7700–7712, 2018.
[50] A. Rizwan, M. Jaber, F. Filali, A. Imran, and A. Abu-Dayya, “A zero-touch network service management approach using ai-enabled cdr analysis,” IEEE Access, vol. 9, pp. 157 699–157 714, 2021
[51] (2021) The unsw-nb15 dataset. [Online]. Available: https://research. unsw.edu.au/projects/unsw-nb15-dataset

9 APPENDICES

Appendix 1 Code for simulating the system

Appendix 2 Code for Testing the system

Appendix 3 Code for Training stage 1

Appendix 4 Code for training stage 2

Appendix 5 Code for data processing

Appendix 6 List of packages installed in the simulation environment.

Appendix 1 Code for simulating the system

Importing required libraries

```
import collections
import numpy as np
import tensorflow as tf
import tensorflow_federated as tff
import pandas as pd
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
import csv
import random
import pickle
from numpy import random as rd
from tensorflow.keras import losses, metrics, optimizers
from tensorflow import reshape, nest, config
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPool2D,
Dropout, Flatten, Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.metrics import SparseCategoricalAccuracy
np.random.seed(0)
np.set_printoptions(precision=3, suppress=True)
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import nest_asyncio
nest_asyncio.apply()
```

Processing the data, the code for processing the data is included below

```
%run Data_Proccessing.ipynb
```

Defining the anomaly percnetages, resulst matrices and number of samples

```
percentagelist_stage2=[0.20,0.30,0.40,0.50,0.60,0.70] # anomaly ratios of
training data used in stage 2
percentagelist_stage1=[0.20,0.30,0.40,0.50,0.60,0.70] # anomaly ratios of
training data used in stage1

resul_matrix
=np.zeros((len(percentagelist_stage2),len(percentagelist_stage1))) # matrix
to store results
difference_matrix=
np.zeros((len(percentagelist_stage2),len(percentagelist_stage1))) # matrix
to store improvement from stage 1 to stage 2

stage1_no_of_samples = 8000 # Number of samples used for stage 1
```

```
stage_2_no_of_samples = 99200 # Number of samples used for stage 2
```

For loop for training the model and testing the model with generated data

```
for iter1 in range(0,len(percentagelist_stage2)): # for loop for
calculating the accuracy of stage 1 and stage 2
    Stage_2_anomaly_percentage=percentagelist_stage2[iter1]
    %run ST2.ipynb # train the model in stage 2 with the data  with a
anomaly percentage defined in Stage_2_anomaly_percentage
    for iter2 in range(0,len(percentagelist_stage1)):
        Stage_1_anomaly_percentage=percentagelist_stage1[iter2]
        %run ST1.ipynb # train the model in stage 1 with the data  with a
anomaly percentage defined in Stage_2_anomaly_percentage
        %run Test_set.ipynb # testing the generated model parameters
        resul_matrix[iter1,iter2] = round_2_nn[20] # storing the overall
system accuracy
        difference_matrix[iter1,iter2] = round_2_nn[20]-round_1_nn[20] #
storing the differenc ebetween stage 1 and stage 2
```

Appendix 2 Code for testing the system

Uploading the generated model parameters

```
a_file = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/modelparameters/NN.pkl", "rb") #NN2
params=pickle.load(a_file)

a_file_1 = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/modelparameters/NN_1.pkl", "rb")
params_1=pickle.load(a_file_1)
```

Starting the test for loop

```
round_1_nn={}
round_2_nn={}
wrong_pred={}
for num in range(1,21):

    K.clear_session() #clear data
    #model 1 detecting  anomalies - stage
1*****************************************************************

    #model1
    kernel_initializer_a1= tf.keras.initializers.constant(params_1[num][0])
    bias_initializer_a1=tf.keras.initializers.constant(params_1[num][1])
    kernel_initializer_b1= tf.keras.initializers.constant(params_1[num][2])
    bias_initializer_b1=tf.keras.initializers.constant(params_1[num][3])
    kernel_initializer_c1= tf.keras.initializers.constant(params_1[num][4])
    bias_initializer_c1=tf.keras.initializers.constant(params_1[num][5])
```

```
    model1 = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(42, )),

tf.keras.layers.Dense(30,activation='relu',kernel_initializer=kernel_initia
lizer_a1,bias_initializer=bias_initializer_a1),

tf.keras.layers.Dense(10,activation='relu',kernel_initializer=kernel_initia
lizer_b1,bias_initializer=bias_initializer_b1),

tf.keras.layers.Dense(2,kernel_initializer=kernel_initializer_c1,bias_initi
alizer=bias_initializer_c1),
    tf.keras.layers.Softmax()
    ])

    #model1.trainable = False

    #model 2
*************************************************************************************
**********************************

    kernel_initializer_a= tf.keras.initializers.constant(params[num][0])
    bias_initializer_a=tf.keras.initializers.constant(params[num][1])
    kernel_initializer_b=tf.keras.initializers.constant(params[num][2])
    bias_initializer_b=tf.keras.initializers.constant(params[num][3])
    kernel_initializer_c=tf.keras.initializers.constant(params[num][4])
    bias_initializer_c=tf.keras.initializers.constant(params[num][5])
    kernel_initializer_d=tf.keras.initializers.constant(params[num][6])
    bias_initializer_d=tf.keras.initializers.constant(params[num][7])

    model2 = tf.keras.Sequential([
  tf.keras.layers.InputLayer(input_shape=(42, )),

tf.keras.layers.Dense(60,activation='relu',kernel_initializer=kernel_initia
lizer_a,bias_initializer=bias_initializer_a),

tf.keras.layers.Dense(30,activation='relu',kernel_initializer=kernel_initia
lizer_b,bias_initializer=bias_initializer_b),

tf.keras.layers.Dense(10,activation='relu',kernel_initializer=kernel_initia
lizer_c,bias_initializer=bias_initializer_c),

tf.keras.layers.Dense(2,kernel_initializer=kernel_initializer_d,bias_initia
lizer=bias_initializer_d),
  tf.keras.layers.Softmax()
])
```

```
model1.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCross
entropy(from_logits=True),metrics=['accuracy'])
    probability_model = tf.keras.Sequential([model1])
    predictions = probability_model.predict(x_test)


    #collecting all normal data as per model
1*************************************************************************
******

    new_dict_1=[]
    new_dict_label_1 =[]
    anomaly_list_1=[]
    False_negatives =list()
    for x in range(len(predictions)):
        if round(np.argmax(predictions[x]))==0:
            anomaly_list_1.append(x) #all non-anomalies

    #created data for model
2*************************************************************************
**********************
    for index in anomaly_list_1:
        m=x_test[index]#.tolist()
        new_dict_1.append(m)
        new_dict_label_1.append(y_test[index])
    new_dict_1 = np.array(new_dict_1)
    new_dict_label_1=np.array(new_dict_label_1)

    #Compiling model
2*************************************************************************
******************************



model2.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCross
entropy(from_logits=True),metrics=['accuracy'])
    probability_model_2 = tf.keras.Sequential([model2])
    predictions2 = probability_model_2.predict(new_dict_1)

    #calculating accuracy of model
1*************************************************************************
****************

    #error list in model 1
    Error_list = list()
    for x in range(len(predictions)):
        if (round(np.argmax(predictions[x]))-y_test[x])!=0:
```

```
            Error_list.append(x)

    round_1_nn[num] = 1-len(Error_list)/len(x_test)

    #calculating the accuracy after the model 2

    Wrong_pred = list()
    for x in range(len(predictions2)):
        if (np.argmax(predictions2[x])-new_dict_label_1[x])!=0:
            Wrong_pred.append(x)


    for i in Error_list:
        if i not in anomaly_list_1:
            Wrong_pred.append(i)


    round_2_nn[num]=1-(len(Wrong_pred)/len(x_test))
```

Storing test results

```
a_file = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/parameters/rnd2" + str(stage1_no_of_samples) + "A" +
str(int(Stage_1_anomaly_percentage*100))+"B"+
str(int(Stage_2_anomaly_percentage*100))+".pkl", "wb")
pickle.dump(round_2_nn, a_file)
a_file.close()

a_file1 = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/parameters/rnd1" + str(stage1_no_of_samples) + "A" +
str(int(Stage_1_anomaly_percentage*100))+"B"+
str(int(Stage_2_anomaly_percentage*100))+".pkl", "wb")
pickle.dump(round_1_nn, a_file1)
a_file.close()
```

Appendix 3 Code for training stage 2

```
dftrain2, ytrain2 =
Create_data_set(stage_2_no_of_samples,Stage_2_anomaly_percentage)

#preparing training data
no_of_clients2=10
index_list2= list(range(0,len(dftrain2)))
no_of_clients_list2= list(range(0,no_of_clients2))
sample_size2=len(dftrain2)//no_of_clients2
client_train_dataset2 = collections.OrderedDict()
```

```
for index in no_of_clients_list2:
    client_name2 = "client_" + str(index)
    client_data_ind2 = random.sample(index_list2,sample_size2)
    index_list2 = list(set(index_list2)-set(client_data_ind2))
    new_dict2 =[dftrain2[y] for y in client_data_ind2]
    new_dict_label2= [ytrain2[y] for y in client_data_ind2]
    client_train_dataset2[client_name2] =collections.OrderedDict((('y',
new_dict_label2), ('x', new_dict2)))

train_dataset2 =
tff.simulation.FromTensorSlicesClientData(client_train_dataset2)

SHUFFLE_BUFFER = sample_size2
PREFETCH_BUFFER = 10
NUM_EPOCHS = 5
BATCH_SIZE = 32
def preprocess(dataset):
  return
dataset.repeat(NUM_EPOCHS).shuffle(SHUFFLE_BUFFER).batch(BATCH_SIZE).prefet
ch(PREFETCH_BUFFER)

def make_federated_data(client_data, client_ids):
    return [preprocess(client_data.create_tf_dataset_for_client(x)) for x
in client_ids]

federated_train_data2 = make_federated_data(train_dataset2,
train_dataset2.client_ids)

sample_dataset2 =
train_dataset2.create_tf_dataset_for_client(train_dataset2.client_ids[0])
preprocessed_sample_dataset2 = preprocess(sample_dataset2)
```

defining the model for training

```
def create_keras_model2():
  return tf.keras.models.Sequential([
  tf.keras.layers.InputLayer(input_shape=(42, )),
  tf.keras.layers.Dense(60,activation='relu'),
  tf.keras.layers.Dense(30,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(2),
  tf.keras.layers.Softmax(),
  ])

def model_fn2():

  keras_model2 = create_keras_model2()
  return tff.learning.from_keras_model(
      keras_model2,
```

```
      input_spec=federated_train_data2[0].element_spec,
      loss=losses.SparseCategoricalCrossentropy(),
  )

client_lr2 =0.0001
server_lr2 =1
mean2 = tff.aggregators.MeanFactory()
iterative_process2 = tff.learning.build_federated_averaging_process(
    model_fn2,
    model_update_aggregation_factory=mean2,
    client_optimizer_fn=lambda: optimizers.Adam( ))
```

Initializing the process

```
state2 = iterative_process2.initialize()

NUM_ROUNDS2=21
parameter_dict2 ={}
for round_num in range(1, NUM_ROUNDS2):
    state2, metrics2 = iterative_process2.next(state2,
federated_train_data2)
    parameter_dict2 [round_num]= state2.model.trainable
a_file = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/modelparameters/NN.pkl", "wb")
pickle.dump(parameter_dict2, a_file)
a_file.close()
```

Appendix 4 Code for training stage 1

preparing training data

```
dftrain, ytrain =  Create_data_set(stage1_no_of_samples,
Stage_1_anomaly_percentage) # create data set with given ratios

no_of_clients=10
index_list= list(range(0,len(dftrain)))
no_of_clients_list= list(range(0,no_of_clients))
sample_size=len(dftrain)//no_of_clients
client_train_dataset = collections.OrderedDict()

for index in no_of_clients_list:
    client_name = "client_" + str(index)
    client_data_ind = random.sample(index_list,sample_size)
    index_list = list(set(index_list)-set(client_data_ind))
    new_dict =[dftrain[y] for y in client_data_ind]
    new_dict_label= [ytrain[y] for y in client_data_ind]
    client_train_dataset[client_name] =collections.OrderedDict((('y',
new_dict_label), ('x', new_dict)))
```

```
train_dataset =
tff.simulation.FromTensorSlicesClientData(client_train_dataset)

SHUFFLE_BUFFER = sample_size
PREFETCH_BUFFER = 10
NUM_EPOCHS = 5
BATCH_SIZE = 32
def preprocess(dataset):
  return
dataset.repeat(NUM_EPOCHS).shuffle(SHUFFLE_BUFFER).batch(BATCH_SIZE).prefet
ch(PREFETCH_BUFFER)

def make_federated_data(client_data, client_ids):
    return [preprocess(client_data.create_tf_dataset_for_client(x)) for x
in client_ids]

federated_train_data = make_federated_data(train_dataset,
train_dataset.client_ids)

sample_dataset =
train_dataset.create_tf_dataset_for_client(train_dataset.client_ids[0])
preprocessed_sample_dataset = preprocess(sample_dataset)
```

defining the function for generating the model

```
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(),
  ])

weights= [1,2]
def model_fn():
  keras_model = create_keras_model()
  return tff.learning.from_keras_model(
      keras_model,
      input_spec=federated_train_data[0].element_spec,
      loss=losses.SparseCategoricalCrossentropy())

client_lr =0.0001#1e-2
server_lr =1#1e-2 #change per client
mean = tff.aggregators.MeanFactory()
iterative_process = tff.learning.build_federated_averaging_process(
    model_fn,
    model_update_aggregation_factory=mean,
```

```
    client_optimizer_fn=lambda: optimizers.Adam( ))
```

Starting the training process

```
state = iterative_process.initialize()

NUM_ROUNDS=21
Num_Of_Layers= 4+1
parameter_dict_stage1 ={}
for round_num in range(1, NUM_ROUNDS):
    state, metrics = iterative_process.next(state, federated_train_data)
    parameter_dict_stage1[round_num]= state.model.trainable
a_file = open("/Users/Suwani Jayaweera/project_v1/final
scripts/phase_1/modelparameters/NN_1.pkl", "wb")
pickle.dump(parameter_dict_stage1, a_file)
a_file.close()
```

Appendix 5 Code for processing data

This file is for processing data

```
import nest_asyncio
nest_asyncio.apply()

import numpy as np
import csv
import random
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

np.random.seed(0)

a = pd.read_csv("A.csv")
b = pd.read_csv("B.csv")

Data_set_combined = pd.concat([a,b], axis=0)
Label = Data_set_combined.pop('label')
Length_of_dataset = len(Data_set_combined)
scaler = MinMaxScaler()
Data_set= scaler.fit_transform(Data_set_combined)
Data_set = pd.DataFrame(Data_set, columns=Data_set_combined.columns)
Data_set = np.array(Data_set)
Label = np.array(Label)
```

Create test data set

```
number_of_test_data=10000;
index_list= list(range(0,len(Label)))
Test_index = random.sample(index_list,number_of_test_data)
```

```
index_list = list(set(index_list)-set(Test_index))
x_test =[]
y_test =[]
for i in Test_index:
    m=Data_set[i].tolist()
    x_test.append(m)
    y_test.append(Label[i])

print('There are %d samples of data for training.' % (Length_of_dataset-
number_of_test_data))
```

Create train data

```
List_of_anomalies= list()
List_of_normal_flow = list()
Length_of_dataset = len(index_list) # take the length of the remaining data
list
for i in index_list:
    if Label[i] ==1:
        List_of_anomalies.append(i)
    else:
        List_of_normal_flow.append(i)

print("Number of Anomalies should be less than %d"
%(len(List_of_anomalies)))
print("Number of Normal flow should be less than %d"
%(len(List_of_normal_flow)))

def Create_data_set(num_of_data, Anomaly_data_ratio):
    x_train=[]
    y_train =[]
    Number_of_Anomaly_data_rows = round(num_of_data*Anomaly_data_ratio)#
create the number of anomaly data rows
    Number_of_Normal_data_rows = num_of_data - Number_of_Anomaly_data_rows#
deducted from the total number of rows so there are no erros de to rounding
off
    Anomaly_index =
random.sample(List_of_anomalies,Number_of_Anomaly_data_rows)#get a random
sample of number of anomaly data rows from the anomaly list
    Normal_index =
random.sample(List_of_normal_flow,Number_of_Normal_data_rows)#get a random
sample of number of anomaly data rows from the normal list
    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])

    return x_train, y_train
```

Appendix 7 List of packages installed in the simulation environment.

Name	Version	Build	Channel
_tflow_select	2.3.0	mkl	
abseil-cpp	20210324	hd77b12b_0	
absl-py	0.9.0	pypi_0	pypi
aiohttp	3.7.4.post0	py38h2bbff1b_2	
argon2-cffi	21.1.0	pypi_0	pypi
astor	0.8.1	py38haa95532_0	
astunparse	1.6.3	py_0	
async-timeout	3.0.1	py38haa95532_0	
attrs	19.3.0	pypi_0	pypi
backcall	0.2.0	pypi_0	pypi
blas	1	mkl	
bleach	4.1.0	pypi_0	pypi
blinker	1.4	py38haa95532_0	
brotlipy	0.7.0	py38h2bbff1b_1003	
ca-certificates	2021.10.26	haa95532_2	
cachetools	3.1.1	pypi_0	pypi
certifi	2021.10.8	py38haa95532_0	
cffi	1.15.0	py38h2bbff1b_0	
chardet	4.0.0	py38haa95532_1003	
charset-normalizer	2.0.4	pyhd3eb1b0_0	
click	8.0.3	pyhd3eb1b0_0	
colorama	0.4.4	pypi_0	pypi
cryptography	3.4.8	py38h71e12ea_0	
cycler	0.11.0	pypi_0	pypi
dataclasses	0.8	pyh6d0b6a4_7	
debugpy	1.5.1	pypi_0	pypi
decorator	5.1.0	pypi_0	pypi
defusedxml	0.7.1	pypi_0	pypi
dm-tree	0.1.6	pypi_0	pypi
entrypoints	0.3	pypi_0	pypi
flatbuffers	2	pypi_0	pypi
fonttools	4.28.2	pypi_0	pypi
gast	0.3.3	pypi_0	pypi
giflib	5.2.1	h62dcd97_0	
google-auth	1.33.0	pyhd3eb1b0_0	
google-auth-oauthlib	0.4.1	py_2	
google-pasta	0.2.0	pyhd3eb1b0_0	
grpcio	1.29.0	pypi_0	pypi
h5py	2.10.0	pypi_0	pypi
hdf5	1.10.6	h7ebc959_0	

Name	Version	Build	Channel
icc_rt	2019.0.0	h0cc432a_1	
icu	68.1	h6c2663c_0	
idna	3.2	pyhd3eb1b0_0	
importlib-resources	5.4.0	pypi_0	pypi
intel-openmp	2021.4.0	haa95532_3556	
ipykernel	6.5.1	pypi_0	pypi
ipython	7.29.0	pypi_0	pypi
ipython-genutils	0.2.0	pypi_0	pypi
jedi	0.18.1	pypi_0	pypi
jinja2	3.0.3	pypi_0	pypi
joblib	1.1.0	pypi_0	pypi
jpeg	9d	h2bbff1b_0	
jsonschema	4.2.1	pypi_0	pypi
jupyter-client	7.0.6	pypi_0	pypi
jupyter-core	4.9.1	pypi_0	pypi
jupyterlab-pygments	0.1.2	pypi_0	pypi
keras	2.7.0	pypi_0	pypi
keras-nightly	2.8.0.dev2021112208	pypi_0	pypi
keras-preprocessing	1.1.2	pyhd3eb1b0_0	
kiwisolver	1.3.2	pypi_0	pypi
libclang	12.0.0	pypi_0	pypi
libcurl	7.78.0	h86230a5_0	
libpng	1.6.37	h2a8f88b_0	
libprotobuf	3.14.0	h23ce68f_0	
libssh2	1.9.0	h7a1dbc1_1	
markdown	3.1.1	py38_0	
markupsafe	2.0.1	pypi_0	pypi
matplotlib	3.5.0	pypi_0	pypi
matplotlib-inline	0.1.3	pypi_0	pypi
mistune	0.8.4	pypi_0	pypi
mkl	2021.4.0	haa95532_640	
mkl-service	2.4.0	py38h2bbff1b_0	
mkl_fft	1.3.1	py38h277e83a_0	
mkl_random	1.2.2	py38hf11a4ad_0	
mpmath	1.2.1	pypi_0	pypi
multidict	5.1.0	py38h2bbff1b_2	
nbclient	0.5.9	pypi_0	pypi
nbconvert	6.3.0	pypi_0	pypi
nbformat	5.1.3	pypi_0	pypi
nest-asyncio	1.5.1	pypi_0	pypi
notebook	6.4.6	pypi_0	pypi
numpy	1.18.5	pypi_0	pypi
oauthlib	3.1.1	pyhd3eb1b0_0	
openssl	1.1.1l	h2bbff1b_0	
opt_einsum	3.3.0	pyhd3eb1b0_1	

Name	Version	Build	Channel
packaging	21.3	pypi_0	pypi
pandas	1.3.4	pypi_0	pypi
pandocfilters	1.5.0	pypi_0	pypi
parso	0.8.2	pypi_0	pypi
pickleshare	0.7.5	pypi_0	pypi
pillow	8.4.0	pypi_0	pypi
pip	21.3.1	pypi_0	pypi
portpicker	1.3.9	pypi_0	pypi
prometheus-client	0.12.0	pypi_0	pypi
prompt-toolkit	3.0.22	pypi_0	pypi
protobuf	3.14.0	py38hd77b12b_1	
pyasn1	0.4.8	pyhd3eb1b0_0	
pyasn1-modules	0.2.8	py_0	
pycparser	2.21	pyhd3eb1b0_0	
pygments	2.10.0	pypi_0	pypi
pyjwt	2.1.0	py38haa95532_0	
pyopenssl	21.0.0	pyhd3eb1b0_1	
pyparsing	3.0.6	pypi_0	pypi
pyreadline	2.1	py38_1	
pyrsistent	0.18.0	pypi_0	pypi
pysocks	1.7.1	py38haa95532_0	
python	3.8.12	h6244533_0	
python-dateutil	2.8.2	pypi_0	pypi
pytz	2021.3	pypi_0	pypi
pywin32	302	pypi_0	pypi
pywinpty	1.1.6	pypi_0	pypi
pyzmq	22.3.0	pypi_0	pypi
requests	2.26.0	pyhd3eb1b0_0	
requests-oauthlib	1.3.0	py_0	
retrying	1.3.3	pypi_0	pypi
rsa	4.7.2	pyhd3eb1b0_1	
scikit-learn	1.0.1	pypi_0	pypi
scipy	1.7.2	pypi_0	pypi
semantic-version	2.8.5	pypi_0	pypi
send2trash	1.8.0	pypi_0	pypi
setuptools	59.2.0	pypi_0	pypi
setuptools-scm	6.3.2	pypi_0	pypi
six	1.16.0	pyhd3eb1b0_0	
snappy	1.1.8	h33f27b4_0	
sqlite	3.36.0	h2bbff1b_0	
tensorboard	2.7.0	pypi_0	pypi
tensorboard-data-server	0.6.0	py38haa95532_0	
tensorboard-plugin-wit	1.6.0	py_0	
tensorflow	2.3.4	pypi_0	pypi
tensorflow-addons	0.11.2	pypi_0	pypi

Name	Version	Build	Channel
tensorflow-estimator	2.3.0	pypi_0	pypi
tensorflow-federated	0.17.0	pypi_0	pypi
tensorflow-io-gcs-filesystem	0.22.0	pypi_0	pypi
tensorflow-model-optimization	0.4.1	pypi_0	pypi
tensorflow-privacy	0.5.2	pypi_0	pypi
termcolor	1.1.0	py38haa95532_1	
terminado	0.12.1	pypi_0	pypi
testpath	0.5.0	pypi_0	pypi
tf-estimator-nightly	2.8.0.dev2021112209	pypi_0	pypi
threadpoolctl	3.0.0	pypi_0	pypi
tomli	1.2.2	pypi_0	pypi
tornado	6.1	pypi_0	pypi
traitlets	5.1.1	pypi_0	pypi
typeguard	2.13.0	pypi_0	pypi
typing-extensions	3.10.0.2	hd3eb1b0_0	
typing_extensions	3.10.0.2	pyh06a4308_0	
tzdata	2021e	hda174b7_0	
urllib3	1.26.7	pyhd3eb1b0_0	
vc	14.2	h21ff451_1	
vs2015_runtime	14.27.29016	h5e58377_2	
wcwidth	0.2.5	pypi_0	pypi
webencodings	0.5.1	pypi_0	pypi
werkzeug	2.0.2	pyhd3eb1b0_0	
wheel	0.37.0	pypi_0	pypi
win_inet_pton	1.1.0	py38haa95532_0	
wincertstore	0.2	py38haa95532_2	
wrapt	1.12.1	py38he774522_1	
yarl	1.6.3	py38h2bbff1b_0	
zipp	3.6.0	pyhd3eb1b0_0	
zlib	1.2.11	h62dcd97_4	