Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.051307

ARTICLE

Federated Network Intelligence Orchestration for Scalable and Automated FL-Based Anomaly Detection in B5G Networks

Pablo Fernández Saura[1,*], José M. Bernabé Murcia[1], Emilio García de la Calera Molina[1], Alejandro Molina Zarca[2], Jorge Bernal Bernabé[1] and Antonio F. Skarmeta Gómez[1]

[1]Department of Information and Communications Engineering, University of Murcia, Murcia, 30100, Spain

[2]University Center of Defense, Spanish Air Force Academy, San Javier, 30720, Spain

*Corresponding Author: Pablo Fernández Saura. Email: pablofs@um.es

Received: 01 March 2024 Accepted: 20 May 2024 Published: 18 July 2024

ABSTRACT

The management of network intelligence in Beyond 5G (B5G) networks encompasses the complex challenges of scalability, dynamicity, interoperability, privacy, and security. These are essential steps towards achieving the realization of truly ubiquitous Artificial Intelligence (AI)-based analytics, empowering seamless integration across the entire Continuum (Edge, Fog, Core, Cloud). This paper introduces a Federated Network Intelligence Orchestration approach aimed at scalable and automated Federated Learning (FL)-based anomaly detection in B5G networks. By leveraging a horizontal Federated learning approach based on the FedAvg aggregation algorithm, which employs a deep autoencoder model trained on non-anomalous traffic samples to recognize normal behavior, the system orchestrates network intelligence to detect and prevent cyber-attacks. Integrated into a B5G Zero-touch Service Management (ZSM) aligned Security Framework, the proposal utilizes multi-domain and multi-tenant orchestration to automate and scale the deployment of FL-agents and AI-based anomaly detectors, enhancing reaction capabilities against cyber-attacks. The proposed FL architecture can be dynamically deployed across the B5G Continuum, utilizing a hierarchy of Network Intelligence orchestrators for real-time anomaly and security threat handling. Implementation includes FL enforcement operations for interoperability and extensibility, enabling dynamic deployment, configuration, and reconfiguration on demand. Performance validation of the proposed solution was conducted through dynamic orchestration, FL, and real-time anomaly detection processes using a practical test environment. Analysis of key performance metrics, leveraging the 5G-NIDD dataset, demonstrates the system's capability for automatic and near real-time handling of anomalies and attacks, including real-time network monitoring and countermeasure implementation for mitigation.

KEYWORDS

Federated learning; 6G; orchestration; anomaly detection; security policy

1 Introduction

In recent years, there has been growing research interest in the field of Network Intelligence (NI) orchestration and Federated Learning (FL) for anomaly detection in Beyond-5G/6G networks. The real-time data analytics required by critical applications are moving AI-based processes to

edge devices, which are closer to the data sources. These edge devices run intelligence processes that must be orchestrated seamlessly across the Continuum (Internet-of-Things (IoT), Edge, Fog, Core, Cloud). In this sense, edge intelligence, which refers to bringing training data and low-level learning tasks to the network's edge, can be realized through an FL approach. The move towards decentralized learning paradigms sets the main motivation for this research, focusing on addressing the shortcomings inherent in traditional centralized and signature-based anomaly/intrusion detection systems—a critical examination that we expand upon in Section 2.

1.1 Federated Learning: An Emerging Paradigm

FL was proposed by McMahan et al. [1] as an alternative to centralized machine learning in which a global model can be trained from decentralized data, thus preserving its privacy during the learning process. Over the last few years, it has been used to design intrusion detection frameworks due to its efficiency and scalability, among other aspects. FL brings several benefits to these networks, such as leveraging network control, improving behavior prediction, and achieving accurate anomaly detection, since all these processes require ubiquitous AI spanned across the Continuum. Yet, the novelty of our approach lies in harnessing these benefits within a specifically designed policy-based dynamic orchestration within a B5G security management framework, setting our research apart from other works already published in the context of FL-based anomaly/intrusion detection.

1.2 Challenges

The integration of FL into B5G networks introduces several challenges, including orchestration and management of the FL federation, scalability, automation, and management of non-independent and identically distributed data (non-IID), interoperability across B5G tenants/domains, and extensibility, among others.

To address these challenges, future architectures will need Artificial Intelligence (AI) processes to be deployed and coordinated in a distributed way, configured, and managed dynamically and on demand. The policy-based security orchestration paradigm can be a powerful ally as it can decide and manage how the security tools (security enablers) should be enforced. Besides, the policy-based approach allows users to define intents/policies that can be transformed into real security deployments/configurations across the infrastructure without requiring knowledge about underlying technologies.

The combination of these two approaches enables the creation of an efficient and scalable system on demand for deploying and decommissioning real-time agent-based FL-driven intrusion detection systems across network segments. On the one hand, FL allows the training of a centralized global model from decentralized data while preserving its privacy and improving efficiency and scalability by bringing the training computation closer to the edge of the network, where the data is stored.

FL-based anomaly detection allows for converging on global and leveraged AI models that can be used to detect, predict, and alert anomalies and intrusions, performing timely mitigation actions that minimize the consequences of a potential attack. On the other hand, policy-based orchestration ensures that the infrastructure is interoperable, extensible, and well-managed across heterogeneous domains, governing Software-Defined Networking (SDN) or Network Function Virtualization (NFV) technologies to provide key network properties such as scalability, reliability, and dynamicity to network intelligence management.

1.3 Main Contributions

This subsection highlights the main contributions of our work, detailed as follows:

- Design of a B5G security management framework, enhanced with decentralized anomaly detection capabilities, conceived to strengthen security, reliability, and privacy across the Continuum. This design leverages the unique advantages of distributed computing to enhance real-time security measures without compromising data privacy or system performance.
- Proposal of model-driven proactive and reactive Network Intelligence Security Orchestration mechanisms aimed at managing and choreographing the FL processes. Such orchestration mechanisms are key in dynamically adapting to evolving threats, ensuring that security measures are both proactive and adaptable to real-time incidents.
- Creation of a novel FL-policy model to achieve interoperability and extensibility, while the proposed orchestration system ensures desirable features in terms of scalability, context awareness, dynamicity, and reaction capabilities to prevent and counter cyberattacks. The FL-policy model introduces a framework for seamless policy integration, facilitating the agile adaptation of security strategies to protect against novel or unseen threats that may occur.
- Exhaustive performance evaluation of the orchestration, learning, and real-time detection processes, considering different scenarios, configurations, and a wide range of performance metrics, and making use of the recent 5G-NIDD for model training and detection performance assessment. This comprehensive evaluation not only validates the effectiveness of our framework but also provides critical insights into optimizing performance across diverse network environments.

In this study, we aim to extend the discourse established by existing literature, specifically targeting the nuances and challenges underscored within FL applications in B5G/6G networks. Our efforts are directed towards proposing methodologies and solutions that not only bridge identified gaps but also innovate within the realm of network intelligence and anomaly detection. Detailed discussions and comparisons related to this progression are presented in the following sections.

1.4 Paper Organization

The rest of the paper is organized as follows: Section 2 reviews the existing literature and research related to the FL approach and network intelligence. Section 3 introduces the framework and describes the design of orchestration, decision-making, and detection, showing its behavior in different stages. Section 4 describes some implementation details about the systems proposed. Section 5 provides an in-depth analysis of the results obtained from the experimentation. Finally, Section 6 outlines the conclusions.

2 Related Works

Generally, signature-based intrusion detection systems have proven to be inefficient in detecting zero-day attacks [2], which are a high-probability threat in next-gen mobile networks. Therefore, machine or deep learning-based intrusion detection systems have better applicability in this type of environment and combined with training based on FL could offer an enhanced and optimal solution. Although several approaches have been presented over the last few years in this line [3], some of them adapted even to work over 5G networks [4], none of them leverages FL for model training as comprehensively as in our proposal. This work integrates a Zero-touch Service Management (ZSM)

aligned Security Framework with multi-domain orchestration, enhancing real-time anomaly detection and security threat handling in B5G networks.

Thantharate [5] offers insights into adaptive network management, which could inform the flexible orchestration capabilities needed to manage network slices dynamically, enhancing our understanding of network scalability and resource optimization. However, while this study provides valuable adaptive methodologies for network management, it lacks the comprehensive integration of real-time security management and multi-domain orchestration that is central to our approach. Similarly, Li et al. [6] explore the enhancement of data privacy and learning efficiency through a novel relay-based FL approach, suggesting complementary strategies for edge intelligence integration in our framework. Despite its innovative approach to data privacy and edge processing, this study does not address the orchestration of these capabilities across a multi-domain environment, nor does it provide a mechanism for dynamic policy-based management, both of which are fundamental components of our FL-based network intrusion detection system, ensuring a scalable, private, and fully customizable framework that adapts to real-time network conditions and threat landscapes.

Asensio-Garriga et al. [7] address the importance of ZSM for managing security slices effectively in highly dynamic Mobile Edge Computing (MEC)-enabled Vehicle-2X (V2X) scenarios, directly supporting the mitigation of Distributed Denial-of-Service (DDoS) attacks through end-to-end service management. This approach emphasizes advanced security orchestration within diverse network contexts, illustrating how ZSM can be instrumental in achieving robust, scalable security solutions tailored to the specific needs of B5G networks.

In the realm of policy-based security orchestration, a significant contribution is made by Hermosilla et al. [8]. This article discusses the essential role of advanced orchestration mechanisms in Unmanned Aerial Vehicle (UAV) networks, employing NFV and SDN technologies to dynamically manage security functions, and aligns with our study by emphasizing the importance of robust, multi-domain orchestration frameworks that integrate security management seamlessly, thus enhancing both the security and efficiency of network operations.

FL applied to secure various types of network environments has gained a lot of interest over the last few years due to its privacy and efficiency, compared to centralized learning settings [9]. In this line, Idrissi et al. [10] assess the power of autoencoders for anomaly detection by comparing them to some widely used and proven mechanisms such as Generative Adversarial Networks (GANs), concluding that they perform better in distributed network intrusion detection, which is the scope of our proposal. Wahab et al. [11] present an exhaustive literature review on how FL can be applied to the field of communication and networking and provide some directives for researchers to overcome key state-of-the-art gaps in the context of communication and networking systems. Another recent study, published by Alsamhi et al. [12], highlights FL's impact on B5G indoor navigation and smart device networks, showing its effectiveness in localization, privacy, and security. FL's decentralized data analysis supports precise navigation and aligns with our FL use in anomaly detection, emphasizing privacy and network optimization.

Nevertheless, other approaches try to mitigate some of the main B5G FL challenges, to reduce communication latency or energy consumption, while improving the model's accuracy. Zhao et al. [13] present a hierarchical methodology that involves the participation of devices at the edge of the network, such as MEC servers. The study formulates a multi-objective optimization problem to balance the trade-off between the model's accuracy, latency, and energy consumption, which is solved by leveraging Deep Reinforcement Learning (DRL) to fine-tune resource allocation and device orchestration

dynamically. This optimization of the FL processes aligns with our goals of using orchestrated FL for anomaly detection in B5G networks.

In addition, several works have been also presented in the line of not only FL orchestration but general NI orchestration. Camelo et al. [14] present an architectural model to cover the requirements and specifications of NI instances in next-gen mobile networks. More specifically, D'Oro et al. [15] propose *OrchestRAN*, a prototype orchestration framework built upon the Open Radio Access Network (RAN) to address the orchestration challenges of a given network optimization/automation objective. Also, Bega et al. [16] apply AI techniques to build a framework that extends the relevant standard specifications with some building blocks based on this paradigm, to control, manage, and orchestrate a B5G network environment. However, these existing works, although address some of the key challenges of NI orchestration and management, do not provide solutions to deploy, configure, and manage in real-time FL-based network intrusion detection systems achieving the desired level of interoperability and scalability across the Continuum. Our approach covers this problem by a policy-based network functions orchestration that can deploy, in real-time and based on the infrastructure needs, several FL agents and the corresponding detection engines to ensure a private, efficient, scalable, and fully customizable detection framework updated to the current threat status of the network.

3 Network Intelligence Orchestration

The incorporation of Artificial Intelligence and Machine Learning into 5G/B5G networks has caused a quantitative and qualitative leap since these technologies, applied to dynamic orchestration, allow intelligent analysis, detection, and decision-making on-demand, deploying in any network segment in the continuum of management and control planes, improving security, resource management, and services which lead to better Quality of Service (QoS), lower latency, and greater energy efficiency, among others.

This section describes the designed architecture, which leverages the Inspire-5GPlus project reference architecture, extending it with different components such as monitoring agents, FL agents, aggregators, and AI-based Anomaly Detection Engines. All these modules would be part of the entire management system, capable of monitoring, learning, detecting, and alerting about potential attacks in a closed loop. At the same time, the learning process should resist possible attacks such as model poisoning or membership inference attacks, for which Differential Privacy (DP) techniques that add noise to exchanged model updates, as described by Ruzafa-Alcázar et al. [17], can be used, and trust scores can be leveraged in an agent selection process for each federated training to minimize poisoning risks. Moreover, the architecture should make it possible to re-train implied detection models, if necessary, i.e., when a threat is detected, and be extensible, scalable, and reliable.

3.1 Architecture

Fig. 1 shows the proposed distributed architecture, composed of three main planes: (i) End-to-end (E2E) Management Domain, which oversees governing all the lower-level management domains, (ii) Management Domains, with one Security Management Domain (SMD) per network segment, e.g., Radio-Access Network (RAN), Transport Network and Core Network Domain, and (iii) Infrastructure Resources, where network segments and their physical and virtual functions are present. Each Management Domain is self-governed, can belong to a different provider, and can be replicated multiple times thanks to virtualization. In addition, each domain would follow a Monitor-Analyze-Plan-Execute (MAPE) loop and would be capable of orchestrating the Network Intelligence across its local domain resources, from the UE or final devices to the components located at the core segment

of the network. They can deploy a FL architecture, composed of several Aggregators and multiple Agents connected to them, serving as a decentralized Intrusion Detection System (IDS) that alerts of security threats potentially taking place in the infrastructure in real-time. Although in Fig. 1 FL actors are only included in the RAN Management and E2E domains for simplicity, they can also be within the Security Analytics Engine (SAE) of the other domains, i.e., Transport and Core. This enables federated training since multiple agents should be involved.

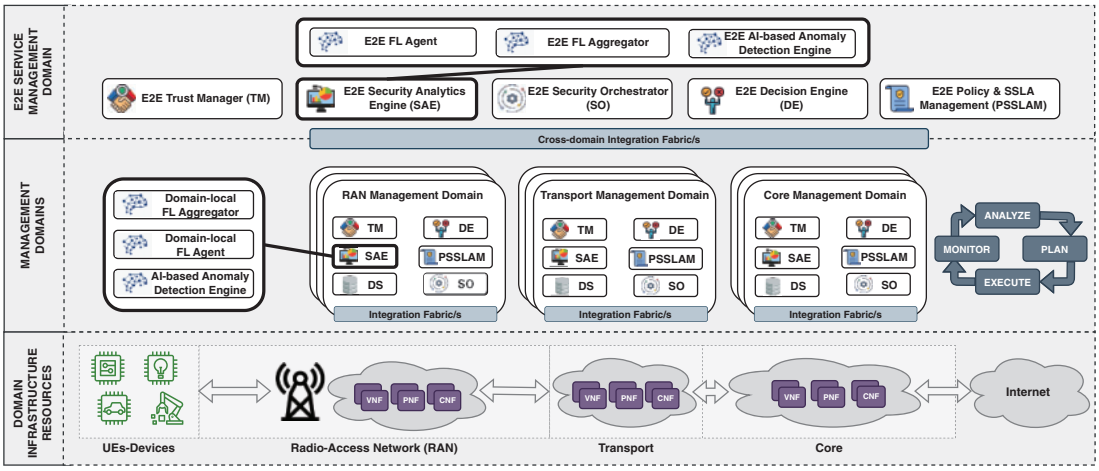

Figure 1: High-level vision of network intelligence orchestration for B5G networks

One of the main reasons for adapting and extending the Inspire-5Gplus reference architecture is that it implements the ZSM approach provided by the European Telecommunications Standards Institute (ETSI), providing essential capabilities such as automation, self-managing, and self-healing. This represents a suitable starting point for building future and next-gen functionalities, so it has been used to show a high-level vision of Network Intelligence orchestration for beyond-5G networks. Following, the different modules that compose each SMD are briefly described:

- **Security Analytics Engine (SAE).** This module analyzes the information/alerts provided by the probes distributed across the infrastructure. The same module at the E2E level focuses on analyzing and correlating events and security issues from different domains to detect issues that could be affecting different domains. In our proposal, this block will be extended to include FL Aggregation, FL agent, and Anomaly Detection features. Those features, as well as their related models, operations, and workflow, will be detailed later.
- **Decision Engine (DE).** It provides dynamic decisions about required countermeasures according to the alerts received from the Security Analytics Engine. At the E2E level, these decisions also consider multiple domains to provide E2E countermeasures.
- **Security Orchestration (SO).** This module oversees orchestrating and enforcing proactive and reactive security policies. It especially decides how the requested security policies will be enforced by selecting the most suitable enforcement point and managing dynamic deployments if required. At the E2E level, the behavior is similar, but orchestration processes consider whole domains instead of low-level infrastructure as enforcement points.
- **Policy & SSLA Management (PSSLAM).** This module manages policy operations, transforming high-level policies into actionable configurations and maintaining a policy repository for traceability and automation support.

- **Trust Management (TM).** This module collects data to calculate trust metrics for orchestration and FL agent selection, aiming to enhance security by identifying the most trustworthy domains.
- **Data Services (DS).** Support other modules by providing essential infrastructure and security data.
- **Integration Fabric.** Using ZSM methodology, it facilitates secure, efficient communication and service management across domains, crucial for FL agent authentication and data exchange.
- **Infrastructure Resources.** Comprises the domain's hardware and software, serving as points for dynamically applying security configurations based on policy decisions.

3.2 Policy-Based FL

To deploy the required entities for FL in the scenario described in the previous section, we devised an interoperable novel FL policy model containing all the necessary information. To this aim, we extended the Medium-level Security Policy Language for Orchestration (MSPL-OP) since it provides multiple security capabilities, extends concepts from Interface to Network Security Functions (I2NSF) of the Internet Engineering Task Force (IETF), is easily extensible, and has been used in relevant and related projects such as ANASTACIA or INSPIRE-5Gplus. Fig. 2 shows the new proposed data model elements aimed at driving the FL policy intents. It will be used to deploy an FL agent or an FL aggregator. The colors used for the boxes in the diagram indicate whether a class is simple, i.e., it directly contains the value (lighter colors), or complex (darker colors), i.e., it contains other classes, either simple or complex.

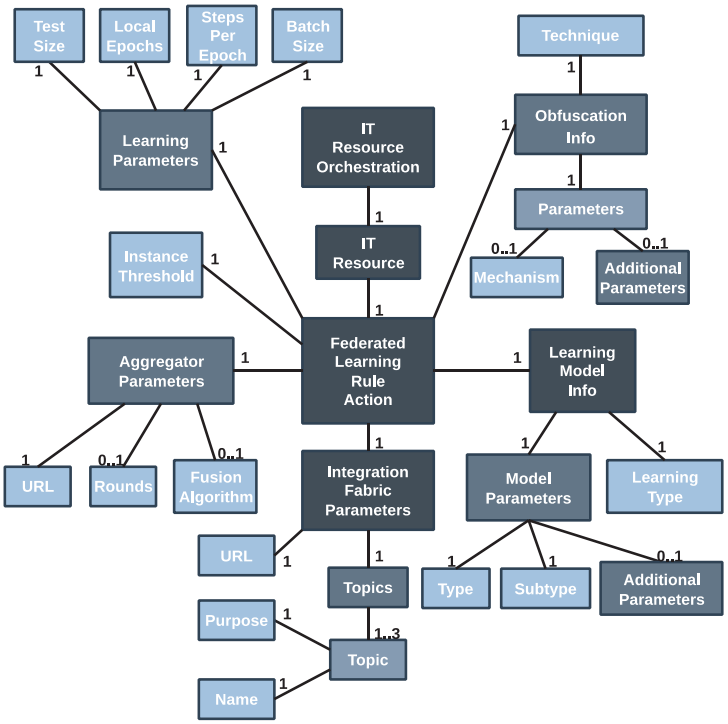

Figure 2: FL policy model for network Intelligence management

As shown in Fig. 1, there are two upper-level classes, *ITResourceOrchestration* and *ITResource* that indicate a resource will be deployed and configured in the infrastructure. In addition to an FL agent and FL aggregator, a resource could be another security entity such as the Anomaly Detection Engine or a monitoring probe. For the FL scenario, the *ITResource* is a *FederatedLearningRuleAction*, that will contain the necessary information to deploy either an FL agent or an FL aggregator. Inside this class, we can find blocks such as the *AggregatorParameters*, that contain the necessary information to deploy an aggregator (Uniform Resource Locator (URL), FL rounds, and fusion algorithm to be used), or an agent, that would need to know where the aggregator is located, but not the rounds or the fusion algorithm.

Moreover, there are blocks to indicate the Integration Fabric parameters (*IntegrationFabricParameters*), which in this example is assumed to work as a publish/subscribe platform where information is stored and retrieved from topics or to specify the details of the Machine Learning (ML) model to be trained, as part of the class *LearningModelInfo*. In the case of a Deep Autoencoder for anomaly detection, we should set "unsupervised" in attribute *LearningType*, "neural network" in *Type*, and "autoencoder" in *Subtype*. The *AdditionalParameters* attribute can be used to set more details about the model, such as the number of layers, the number of neurons in each layer, or the loss function to be used.

Next, common learning parameters such as the test size, local epochs, steps per epoch, and batch size can be set inside *LearningParameters*, and obfuscation mechanisms to elude membership inference attacks can be set inside *ObfuscationInfo*. For instance, setting "differential-privacy" in *Technique*, "Laplace Bounded Domain" in *Mechanism*, and epsilon and delta values inside *AdditionalParameters*. Lastly, the *InstanceThreshold* attribute represents the number of traffic samples, e.g., traffic flows, that must be reached to register against the aggregator and start the federated training.

In Fig. 3, an example of a policy to deploy an FL agent is shown. It is set to work with an instance threshold of 100,000, assuming each instance represents a network traffic flow. The FL process is configured with 100 training rounds and *FedAvg* as the fusion algorithm. Furthermore, it is configured to use an autoencoder, i.e., an unsupervised learning algorithm based on a neural network for anomaly detection, and its parameters are also specified, such as the number of encoder and decoder layers, the number of neurons in each one, and the number of neurons in the code layer, as well as the loss function to be used. Additionally, local training parameters are set: 0.33 as the test size, 5 local epochs, 32 instances as batch size, and 1 step per epoch. Lastly, it is required to use a DP mechanism to obfuscate the exchanged model updates and prevent inference and/or model poisoning attacks. Specifically, it is a Laplace Truncated mechanism configured with an epsilon value of 1 and a delta of 0.

3.3 FL Orchestration Process

The FL Orchestration process in a large-scale distributed B5G network is a crucial task that enables seamless coordination and collaboration among various entities involved, helping to configure, deploy, and manage FL features. Fig. 4 shows the proposed FL Orchestration process, involving several SMDs at different levels (Core, Transport, RAN).

The orchestration process can be triggered in proactive or reactive ways depending on the source of the request. The proactive workflow starts with a user requesting FL features and capabilities to be deployed across the multi-domain infrastructure. The E2E Security Orchestrator analyzes the requested capabilities and computes an orchestration plan to distribute the FL and anomaly detection tasks among the underlying domains. To this aim, it generates as many FL security policies as needed, modeled in the Medium-level Security Policy Language (MSPL) according to the FL model defined

in the previous section, and commands them to the different domains (Fig. 4, steps 1 and 16). SMD orchestrators receive the MSPL-OP policy, composed of as many FL agent policies as required and one FL aggregator policy. Then, it retrieves the available security enablers; these are pieces of software or hardware able to implement the required capabilities, FL agent, and FL Aggregation (Fig. 4, step 2). When the Security Orchestrator has a list of enabler candidates, it runs the allocation algorithm, choosing the best suitable candidate and the best place to deploy the FL agents and the FL aggregator considering the context, environment, and constraints (Fig. 4, step 3). To perform the enforcement, the Security Orchestrator relies on the NFV Management and Orchestration (NFV-MANO) component to deploy the agents and the aggregator in the required place (Fig. 4, steps 4, 5, and 6). In the next step, the Security Orchestrator configures the enablers (FL agents and FL aggregator) with the parameters that have been requested (Fig. 4, steps 7 and 8). The agents will register with the aggregator once they reach the instance threshold. The aggregator, in collaboration with these registered agents, will generate a new global model through the number of training rounds previously specified in the aggregator configuration policy (Fig. 4, steps 9 and 10).

```
<federatedLearningRuleAction>
    <instanceThreshold>100000</instanceThreshold>
    <aggregatorParameters>
        <URL>localhost:3333</URL>
        <rounds>100</rounds>
        <fusionAlgorithm>fed-avg</fusionAlgorithm>
    </aggregatorParameters>
    <integrationFabricParameters>
        <URL>localhost:29032</URL>
        <topics>
            <topic>
                <purpose>training-traffic-topic</purpose>
                <value>benign-flows-info</value>
            </topic>
            <topic>
                <purpose>anomaly-alerts-topic</purpose>
                <value>anomaly-alerts</value>
            </topic>
        </topics>
    </integrationFabricParameters>
    <learningModelInfo>
        <learningType>unsupervised</learningType>
        <modelParameters>
            <type>neural-network</type>
            <subtype>autoencoder</subtype>
            <additionalParameters>
                <encoderLayers>2</encoderLayers>
                <decoderLayers>2</decoderLayers>
                <encoderNeurons>32-16</encoderNeurons>
                <decoderNeurons>16-32</decoderNeurons>
                <codeNeurons>8</codeNeurons>
                <lossFunction>mean-squared-error</lossFunction>
            </additionalParameters>
        </modelParameters>
    </learningModelInfo>
    <learningParameters>
        <testSize>0.33</testSize>
        <localEpochs>5</localEpochs>
        <batchSize>32</batchSize>
        <stepsPerEpoch>1</stepsPerEpoch>
    </learningParameters>
    <obfuscationInfo>
        <technique>differential-privacy</technique>
        <parameters>
            <mechanism>laplace-truncated</mechanism>
            <additionalParameters>
                <epsilon>1.0</epsilon>
                <delta>0.0</delta>
            </additionalParameters>
        </parameters>
    </obfuscationInfo>
</federatedLearningRuleAction>
```

Figure 3: FL agent orchestration policy implementation example

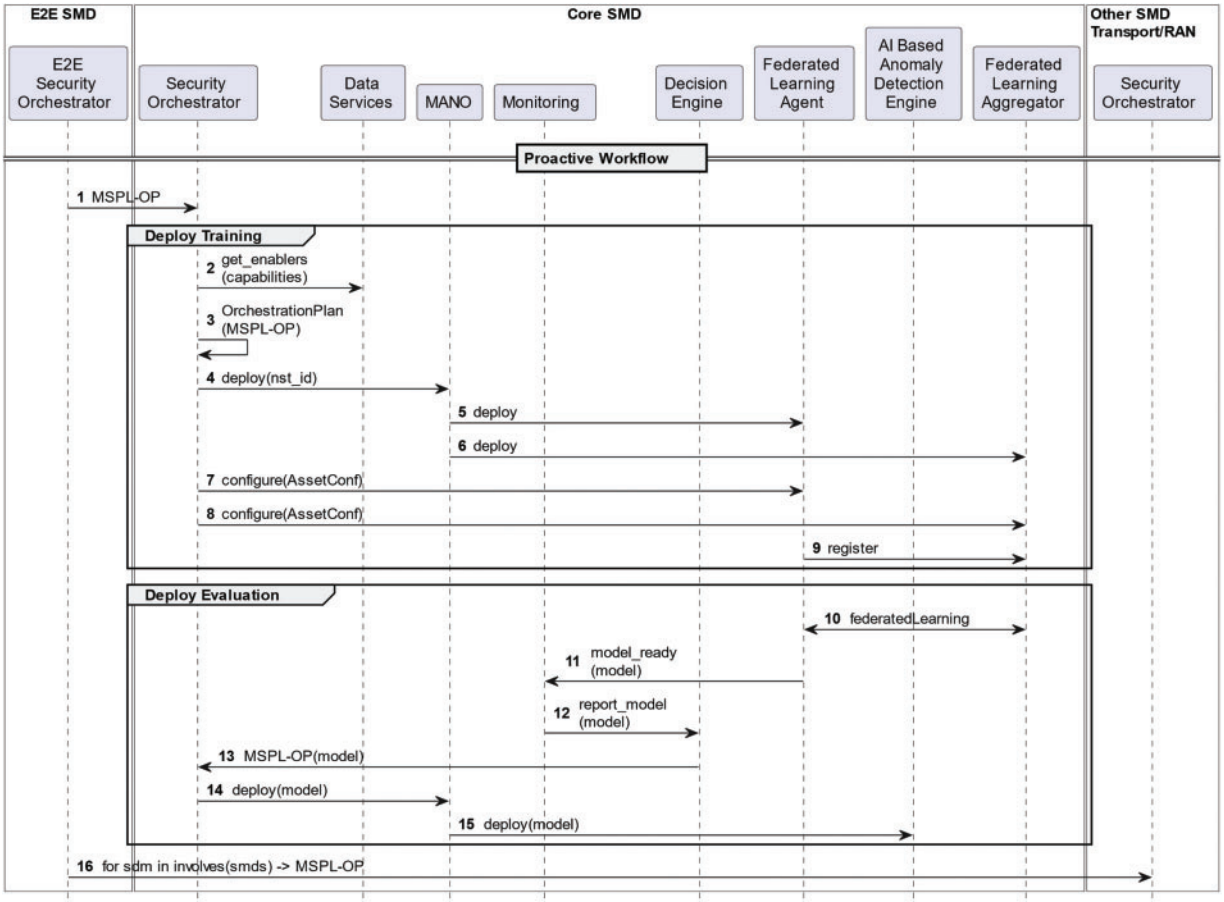

Figure 4: FL orchestration proactive workflow

When the final model is ready, each domain-local FL agent notifies its monitoring module (Fig. 4, step 11), which will trigger the Decision Engine (Fig. 4, step 12). The Decision Engine will generate a new MSPL policy, indicating FL Detection as the main capability, as well as the final model provided (Fig. 4, step 13). The Security Orchestrator receives the request and performs the orchestration process again, which in this case ends with the deployment of the AI-based Anomaly Detection Engine according to the trained model (Fig. 4, steps 14, 15).

A general allocation algorithm, adapted from the one proposed by Hermosilla et al. [8], that considers hard and soft constraints to orchestrate FL agents is depicted in Algorithm 1. First, the key variables are initialized. These are: the MSPLs containing the FL agents configuration details (F), the available Virtual Network Functions (VNFs) in the network, retrieved from the Data Services (V), the list of hard requirements to be met by each FL agent considered for deployment (R), the same list but for soft requirements (S), and the list of available FL agents that meet the requirements (AF), to be filled once both hard and soft requirements are checked.

Algorithm 1: General security enabler orchestration algorithm for FL agents

1: F ← {FL Agents MSPLs}
2: V ← DS.RetrVNFs()

(Continued)

Algorithm 1 (continued)

R ← {Hard Requirements}
S ← {Soft Constraints}
AF ← {}
for $f \in F$ **do**
  for $r \in R$ **do**
    for $v \in V$ **do**
      if $C_r(v) \leq (\sum_{v \in V_f} C_r^a(f_v)) - C_r^n(v')$ **then**
        $AF|f| \leftarrow$ f
      end if
    end for
  end for
end for
for $f \in AF$ **do**
  for $s \in S$ **do**
    for $v \in V$ **do**
      $sum \leftarrow$ sum $+ ((\sum_{v \in V_f} SC_s^a(f_v)) - SC_s^n(v')) * w_s$
    end for
  end for
  $gain_{AF}|f| \leftarrow$ sum
  $sum \leftarrow 0$
end for
$AF \leftarrow$ sort($gain_{AF}$[])
for $f \in AF$ **do**
  Deploy f to its designated VNF
  Configure f with parameters extracted from the MSPL
end for
Algorithm 2 (R, AF, E)

Some of the hard requirements that could be considered are: minimum computational capacity (sufficient CPU and GPU power to handle FL agent operations), available memory (adequate Random Access Memory (RAM) to support the agent's runtime environment and data processing), storage space (necessary disk space for data retention and model parameters), network bandwidth (sufficient throughput for communication between FL agents and servers), latency constraints (requirements to ensure real-time or near-real-time data processing), energy consumption limits (restrictions on power usage, especially critical for edge devices), hardware compatibility (ensuring FL agents are compatible with device architecture), reliability and availability (ensuring minimal downtime and continuous operation), and regulatory compliance (adherence to legal standards like General Data Protection Regulation (GDPR) or Health Insurance Portability Accountability Act (HIPAA), depending on the application domain). On the other side, some of the soft requirements that could be considered are: scalability (ability to adjust resource allocation dynamically based on workload changes), response time (reducing the time it takes for FL agents to respond to data inputs and queries), resource utilization (maximizing the efficiency of resource use like Central Processing Unit (CPU) and memory), or quality of service (maintaining high standards of service performance, reliability, and uptime).

After the initialization, the hard constraints are checked. This is done through a triple-nested loop that iterates over each FL agent, each hard requirement, and each VNF. $C_r(v)$ refers to the constraint function that obtains the normalized value of a requirement r, for a given VNF v. $C_r^a(d_v)$ is the constraint function that gets the actual normalized value of a particular type of requirement r currently available in a given FL agent f for a given v. For instance, available storage in a certain compute node in the network. This information can be retrieved from the system model that would be available in the local-domain Data Services. $C_r^n(v')$, similarly, is the constraint function that calculates the value for the requirement r, needed (n) by a virtual enabler to be allocated, as demanded by the orchestrator, for a given new v' VNF to be allocated. So, the algorithm is checking that the VNF v has sufficient resources, after considering existing allocations, to meet the requirement, and thus f is added to the list of acceptable agents AF.

Next, the same process but for the soft requirements takes place. This checking is addressed through a different approach. Since soft requirements are not a must but a should, the gain that could be obtained by deploying the VNF in such an FL agent is calculated. First, the constraint function $SC_s^a(d_v)$ gets the normalized value of a soft constraint s available a for a given VNF in a device d_v, for instance, available memory. The summarization for all $v \in V_d$ will calculate the total overall available value for the FL agent being considered. Then, the second constraint function $SC_s^n(d_v)$ gets the normalized value of a given soft constraint s needed (n) by the orchestrator for deployment. The difference between those functions, i.e., what is needed and what is available, is finally weighted by the importance given by the administrator to that soft constraint (w_s) and added to the gain of that specific FL agent. As a result, a list of the available FL agents, i.e., the ones that meet the hard requirement specified, sorted by the gain of deploying each one of them in the network, is obtained.

Lastly, the agents in the AF list are iterated in order, and each one of them is both deployed and configured by the orchestrator, based on the specific MSPL policies. After that, the FL training process, depicted in Algorithm 2, would take place, with the number of rounds (R), available FL agents (AF) and local epochs to be executed per round (E) as inputs. R is obtained from the aggregator's configuration policy, while E is retrieved from each FL agent's policy. This orchestration algorithm can be used as well to orchestrate FL aggregators. Different hard and soft requirements from those specified for agents based on the aggregators' needs should be specified.

The state-of-the-art Federated Learning process based on FedAvg aggregation technique that would trigger after the orchestration process has finished and once there are sufficient agents registered against a certain aggregator is shown in Algorithm 2. The rest of the learning parameters, such as the batch size or the test size, are removed for simplicity. On the other hand, the output is the global model W^G. Firstly, the initial global model W_0^G is initialized by the aggregator. Secondly, for each training round, a subset of A is selected based on a specific client-selection procedure and stored in S_r. Next, for each selected agent i, the local model weights Δw_r^i for the current round r (model update) are generated fitting the received global model from the aggregator for that specific round (W_r^G) to the local training data. Finally, the aggregator mixes all the received model updates based on the FedAvg algorithm, generating the global model W_{r+1}^G to be used in the next training round, or in case the current round is the last, the final model to be used for anomaly detection. This algorithm calculates the summation of all model updates received from the agents (Δw_r^i), for each agent i in S_r, and divides it by the number of agents $|S_r|$, obtaining the average of all the received model updates for a certain round r, which is added to the weights of the model from the previous round W_r^G.

Algorithm 2: State-of-the-art FedAvg-based Federated Learning algorithm

Input: *Number of rounds* R, *available agents* A, *number of local epochs* E
Output: *Global model* W^G

Initialize W_0^G
for $r = 0$ *to* $R - 1$ **do**
  $S_r = Choose\ a\ agents\ out\ of\ A$
  for *each agent* $i \in S_r$ **do**
    $\Delta w_r^i = LocalUpdate\ (i, W_r^G)$
  end for
  Aggregate updates using FedAvg:
  $W_{r+1}^G \leftarrow W_r^G + \sum_{i \in S_r} \left(\frac{\Delta w_r^i}{|S_r|} \right)$
end for

3.4 FL-Based Anomaly Detection

As stated in the previous section, the proposal considers, for the FL scenario, at least one aggregator and multiple agents distributed across domains.

The model proposed for real-time anomaly detection tasks after training is a Deep Autoencoder. This type of model aims to reconstruct the input in the output. The idea is to feed the Autoencoder with flow-based non-anomalous traffic samples extracted from the network as training data. Then, the autoencoder will become adept at reconstructing these samples with low error. Since the error can be measured dynamically, an error threshold can be defined for evaluating new traffic so that traffic flows exceeding the threshold will be classified as anomalies, as they significantly differ from the non-anomalous traffic dataset the model has been trained on.

As shown in Fig. 5, once anomalies are detected, the framework can perform reactive actions in real time to apply proper countermeasures. This is when an anomaly is detected by the AI-based Anomaly Detection Engine, which informs the Decision Engine, which comes up with a set of mitigation policies to be orchestrated and enforced to mitigate the attack. As part of this reaction phase, the Decision Engine could command a new learning process so that the agents can offer the AI-based Anomaly Detection Engine a fresher model, updated with the latest status of the network, i.e., including the latest monitored data. Additional actions can also be taken, such as deploying more FL agents and AI-based Anomaly Detection Engines across the infrastructure to protect some domains/subdomains that were not initially considered for the analysis.

More in detail, the process starts with the monitoring of traffic flow data, which is forwarded in real-time to the AI-based Anomaly Detection Engine (Fig. 5, step 1). For each received traffic flow, the engine evaluates it against the model and, in case it is detected as abnormal, an alert is created and forwarded to the Decision Engine to decide which countermeasures should be taken to mitigate the attack (Fig. 5, step 2). These countermeasures are modeled in a policy and sent to the Security Orchestrator for enforcement (Fig. 5, step 3). Once this information is received, the Security Orchestrator consults the Data Services to retrieve the available or suitable enablers in the infrastructure according to the requested detection capability, computes the orchestration plan, and communicates with the NFV-MANO to deploy the corresponding countermeasures (Fig. 5, steps 4 to 6). Assuming filtering the attacker's traffic has been selected as the countermeasure, a Filtering Agent is deployed by the NFV-MANO and configured by the Security Orchestrator (Fig. 5, steps 7 and 8). In addition, based on the infrastructure needs, the Security Orchestrator can command a re-training

process to the corresponding FL aggregator, which will perform, along with the agents, a new FL process, resulting in a new model, updated with the latest threat status of the infrastructure, that will be deployed in the AI-based Anomaly Detection Engine at the end (Fig. 5, steps 9 to 15).

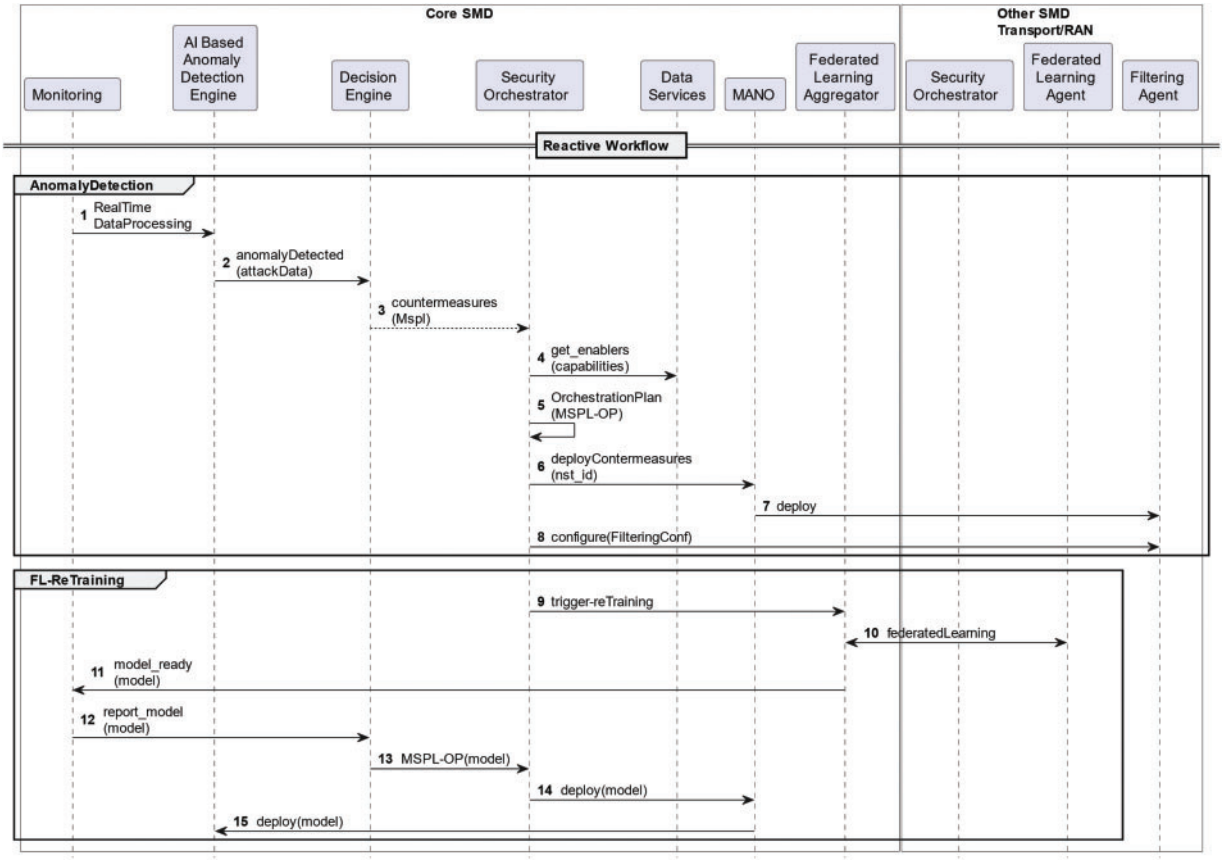

Figure 5: FL orchestration reactive workflow

It is important to highlight that the reactive stage of the framework is not fixed to improve FL features but provides a whole set of countermeasures. The decision module might infer different mitigation actions depending on the detected threat. For instance, it could generate a filtering policy to drop the malicious traffic as close as possible to the source, request the enforcement of a security slice to protect communications, deploy a honeynet, or assign the attacker to a data-plane slice with minimal QoS.

Trust management evaluates the reliability of incoming data and the credibility of FL agents, prioritizing inputs based on their trustworthiness to prevent compromised data from undermining the learning process. Obfuscation, on the other hand, protects sensitive information by disguising its true characteristics, thereby complicating attackers' efforts to exploit the data. These strategies not only mitigate immediate vulnerabilities but also adaptively recalibrate in response to emerging threats, by adjusting obfuscation levels to thwart reverse-engineering attempts or by modifying trust score thresholds to isolate suspicious activities. Additionally, the architecture can deploy targeted countermeasures such as isolating compromised nodes, enhancing honeynet deployments for deeper threat analysis, or reconfiguring network topologies to safeguard critical data paths. Together, these

advanced techniques ensure a comprehensive and resilient defense mechanism, capable of dynamically responding to a broad spectrum of security challenges within the FL ecosystem.

4 Implementation

To attain the desired functionalities, we implemented a proof of concept involving the Security Orchestrator, FL aggregator, FL agent, and AI-based Anomaly Detection Engine. The Security Orchestrator has been developed in Python 3, including specific plugins and drivers for generating and enforcing FL capabilities across the infrastructure. The FL aggregator and FL agents have also been implemented in Python 3 using the Flower framework for FL. For both, a Representational State Transfer Application Programming Interface (REST API) has been implemented so they can be easily configured, and they have been packaged using Docker for easy deployment. The dynamic configuration based on the set of aspects specified in the policy (see model in Fig. 2 and example instantiation in Fig. 3) is fully supported, so both the FL aggregator and FL agents are highly customizable based on the needs. More details about the detection model used and how data is preprocessed in the FL agents can be found in Sections 5.2 and 5.3. Different orchestration agents have also been developed to allow the orchestrator drivers to interact with the aggregator and the agents on demand. The implementation has also been performed in Python 3.

The control plane was deployed within a Kubernetes cluster virtualized on an AMD EPYC 730P 16-Core processor, boasting 32 threads and 128 GB of RAM. On the other hand, the data plane was deployed across another multi-node Kubernetes infrastructure. Nodes were virtualized on an Intel Xeon Gold 6138 processor, each equipped with 40 cores, 80 threads, and 256 GB of RAM. The agents and aggregators were deployed using Helm Chart in Kubernetes. The Radio Access Network (RAN) was deployed using Advanced Wireless Solutions and Services (AW2S) running the Amarisoft Callbox Radio stack, on top of an Intel(R) Core (TM) i7-9700K CPU @ 3.60 GHz with 8 cores, 8 threads, and 8 GB of DDR4 RAM. However, the FL and real-time anomaly detection processes performance assessment has been done using a single machine equipped with a Ryzen 7 3700X CPU @ 3.6 GHz with 8 cores and 16 threads, and 16 GB of DDR4 RAM.

5 Experimentation

To validate the proposed solution, multiple experiments were performed to cover different aspects of the design such training process, real-time anomaly detection as well as on-demand deployment and configuration of the FL components. Below, these aspects to be tested are described:

- **The training process.** For this part, a centralized scenario is compared against different FL settings, varying the number of agents participating and/or the total number of local epochs executed per round, and considering model performance evaluation metrics such as the recall, specificity, balanced accuracy, and F1 score. Also, the evolution of the model loss using the Mean-Squared Error (MSE) is analyzed under different conditions, i.e., incremental number of agents and different local epochs executed per agent and training round.
- **The real-time anomaly detection process.** Here, the minimum, maximum, and average time (and the standard deviation) is spent by the real-time anomaly detection engine from the moment it receives a sample to the moment it alerts the anomaly, in case it predicts it as an anomalous flow. This time is measured for each evaluation phase executed per flow, including the decoding, the preprocessing, and the prediction against the model.

- **The security orchestration process.** FL aggregator and multiple FL agents are deployed and configured on demand according to the security policies received. In this regard, the orchestration time is measured to determine the time the orchestration decision takes depending on the amount of FL agents to be deployed and configured.
- **The enforcement process.** The time the system requires for deploying and configuring FL entities (aggregator and agents) across the infrastructure.

5.1 Training and Detection

Below, the formulas for the model performance metrics considered for the first point can be consulted:

1. Recall: $\frac{\text{TP}}{\text{TP} + \text{FN}}$
2. Specificity: $\frac{\text{TN}}{\text{TN} + \text{FP}}$
3. Balanced accuracy: $\frac{\text{Recall} + \text{Specificity}}{2}$
4. F1 score: $\frac{\text{TP}}{\text{TP} + \frac{1}{2} * (\text{FP} + \text{FN})}$

where: TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives

Recall becomes more relevant than specificity in our context because our primary interest lies in accurately capturing true positives, which represent anomalies. In scenarios like ours, where detecting these anomalies is critical, the ability to identify all relevant instances (high recall) outweighs the importance of correctly identifying all negative cases (high specificity). However, with imbalanced datasets, it's crucial to employ a metric that can provide a fair evaluation of the model's performance across both classes. Therefore, we have chosen balanced accuracy as the principal evaluation metric, which offers a comprehensive measure by considering the model's effectiveness in identifying both classes equally, regardless of their distribution in the dataset. This ensures that the model evaluation reflects its ability to perform well across different conditions, making balanced accuracy an ideal choice for assessing overall performance.

5.2 Data Preparation

For the experimentation part, the 5G-NIDD dataset, presented by Samarakoon et al. [18], has been chosen to test, on one hand, the FL process under different conditions and, on the other hand, the real-time anomaly detection process. This dataset is generated from a realistic 5G testbed deployed in the 5G Test Network at the University of Oulu, Finland. The data is collected from two different *gNodeB* base stations, each one having an attacker node and several benign end-users. On the victim side, there is a server deployed in the 5GTN MEC environment. From this architecture, several attack scenarios have been simulated, including different types of Denial-of-Service (DoS) attacks and port scans.

There are different versions of this dataset publicly available, depending on how the data has been processed. For this study, we selected a version in which the data from both base stations has been merged. In total, there are 1,215,890 samples, each one representing a single traffic flow. Among them, we found 738,153 malicious samples, containing all the attacks launched, i.e., Internet Control Message Protocol (ICMP) Flood, User Datagram Protocol (UDP) Flood, SYN Flood, Hypertext

Transfer Protocol (HTTP) Flood, Slowrate DoS, SYN Scan, Transmission Control Protocol (TCP) Connect Scan, and UDP Scan, and 477,737 benign samples. Each sample contains 48 features, representing flow traffic statistics, and 3 label columns, indicating the binary label, i.e., benign, or malicious, the specific attack type, and the tool used to perform the attack.

To prepare the data for our experiments, we first dropped the label columns indicating the attack type and the attack tool. Then, we balanced the dataset under-sampling, by random drop, the majority class, i.e., the malicious class. As a result, we obtained a fully balanced dataset containing 477,737 samples for each class, while preserving samples for all the attack types previously mentioned in the malicious subset.

Next, we replaced the infinite values for null values, dropped the columns containing more than 30% null values, and imputed the rest of the null values using a k-nearest-neighbors imputer with 3 neighbors and uniform weights for the categorical columns, and a simple imputer for the numerical columns, imputing by the most frequent value for each considered column. Moreover, we calculated the correlation matrix and dropped columns that are highly correlated with others (correlation percentage exceeding 90%). Finally, to split the dataset into training and testing sets, considering that the training set must contain only benign samples, we randomly selected 400,000 benign samples from the original data after the preprocessing steps just mentioned for the training set. For the test set, we randomly picked 66,000 benign samples from the rest, and 66,000 by the same method from the malicious subset. The result of the whole process is a training dataset with 400,000 benign samples and 26 features including the label, and a testing dataset with 132,000 samples with an equal number of samples for the benign and malign classes, being a third part of the training dataset. Both include a total of 25 features, excluding the label, representing 52% of the original features, i.e., before preprocessing.

The resulting training and testing sets will be used to evaluate the centralized approach. For the FL setting, each set is divided into 10 equal and non-overlapping portions, one for each agent, simulating the traffic data that would be monitored in real-time from the real infrastructure. This is, for each agent, a training dataset containing approximately 40,000 benign samples, and a testing dataset containing approximately 13,200 samples with equal samples for both classes.

5.3 Model Definition

The model used for anomaly detection, as previously mentioned in Section 3.4, will be a Deep Autoencoder that will be able to reconstruct benign samples with low error, being trained to do so by using a training dataset only composed of samples from this class, and then detect a malicious class sample since it exceeds an error threshold set at training phase from benign samples.

The architecture of the model can be consulted in Fig. 6. As can be seen, it is composed of, firstly, an encoding part, containing three dense, i.e., fully connected, layers, each one being half the size of the previous layer, being n the number of neurons in the first layer, that correspond to the number of input features, and a drop-out layer between the second and the third, with a drop rate of 20%, meaning that one in five inputs will be randomly excluded from each update cycle, to prevent overfitting. Also, a regularizer with learning rate 1e−07 is set between the first and the second layer to improve performance and contribute to reducing overfitting. Secondly, the code part contains only one layer one-eighth the size of the input layer. Here, the data is compressed at its maximum. Finally, the decoder part, is a mirror of the encoding part. At the output layer, the reconstructed data can be consulted, and the model loss can be calculated by comparing it to the original data in the input.

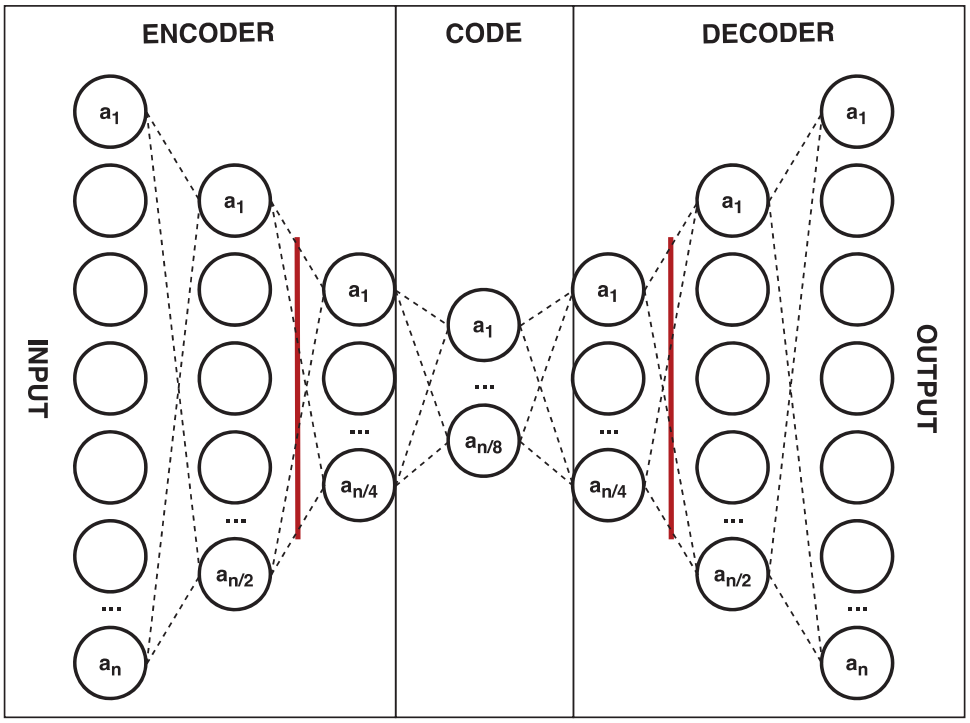

Figure 6: Architecture of deep autoencoder for anomaly detection. *n* represents the activation value for the neuron *n*, being *n* the number of features in the input data. The vertical red bars represent the dropout layers

Between layers, different activation functions are used. For the input and output layers, the *tahn* function is used, while *ReLU* is used for the rest of the layers. The hyperparameters used have been fine-tuned and the best ones have been selected. For instance, several values in the range of 10% to 30%, which are considered optimal values, have been evaluated using balanced accuracy as a metric to set the best drop rate, and the results show that 20% stands as the most accurate one. A similar process to fine-tune the learning rate has been carried out, being 1e−07 the optimal value under the conditions of this experiment. Lastly, the model is compiled using the optimizer *Adam*, and the MSE is selected as the loss function.

5.4 Anomaly Threshold Setting

As commented in previous sections, an anomaly threshold calculated from model loss during the training phase must be used in the real-time evaluation process to discern between benign/normal samples and malicious/abnormal samples. To do so, we propose taking the *n-th* percentile of the MSE values obtained during the training phase, meaning that below that percentile, the n percent of training samples' MSE values are found. So, samples being reconstructed whose MSE exceeds that value will be categorized as anomalous and will be alerted accordingly.

To select the optimal percentile, we employ *Cohen's Kappa* index, which is a statistical value that measures the agreement between two raters (in the context of machine/deep learning, between an algorithm's predictions and the true labels), each one classifying items into categories. It is especially useful to be used as a model performance metric for selecting the optimal anomaly threshold percentile

since the percentile maximizing the Kappa value will maximize the agreement between the original and the predicted labels. Below, the formula to obtain the Cohen's Kappa index can be consulted (Eq. (1)):

$$Kappa\ (\kappa) = \frac{p_o - p_e}{1 - p_e} \tag{1}$$

where:

- p_o is the relative observed agreement among raters (i.e., the proportion of items that the raters agree upon). In other words, it is the accuracy of the model, i.e., the number of correct predictions divided by the number of total predictions made.
- p_e is the hypothetical probability of chance agreement, calculated using the observed data to determine the probabilities of each observer randomly seeing each category. It is equivalent to the formula shown in Eq. (2):

$$p_e = \frac{(TP + FN) \times (TP + FP) + (FP + TN) \times (FN + TN)}{(TP + FP + FN + TN)^2} \tag{2}$$

where: TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

The Cohen's Kappa index has been selected for the following reasons:

- Finding a balance between sensitivity and specificity: By maximizing the kappa coefficient, a balance is found between sensitivity and specificity, which is critical in binary (anomaly) classification,
- Provide robustness against class imbalance: In anomaly classification, the classes tend to be highly imbalanced (few anomalies *vs*. many normal cases). The Kappa index takes this imbalance into account, providing a more reliable measure of the model's ability to distinguish between classes compared to other metrics such as precision.
- Facilitate comparison between models: By offering a single value that summarizes the model's performance, the Kappa index facilitates comparison between different models or configurations, helping to choose the best threshold for classification.

In Fig. 7, the *Cohen's Kappa* index obtained for each considered percentile is shown. As can be concluded, *Kappa* is maximized at the percentile of 84%, so this is the value that will be picked for the rest of the experiments described in the following sections. To extract these results, we have evaluated the model, using a centralized setting with 10 epochs executed, 32 as batch size, and using the preprocessed training and testing datasets (before splitting) described in Section 5.2. The percentiles below 70% are not shown since they are not considered relevant.

Assuming B is the set of MSE values obtained from the benign samples in the testing dataset, composed only of benign samples, A is the set of MSE values obtained from all samples, i.e., benign, and malign, from the testing dataset, and L their true labels, let's define:

- Let $p(i)$ be the i-th percentile of B.
- Let $K(t, L)$ be the Cohen's Kappa Index when the MSE predictions in A are thresholded at t, compared to the true labels L.
- Let $T(j)$ be the threshold corresponding to the j-th percentile.

We are trying to find the threshold $T(j)$ that maximizes $K\left(A > T(j), L\right)$. See Eq. (3):

$$T(j) = \max_i K\left(A > p\left(i\right), L\right) \quad \text{for} \quad 1 \leq i \leq 100 \tag{3}$$

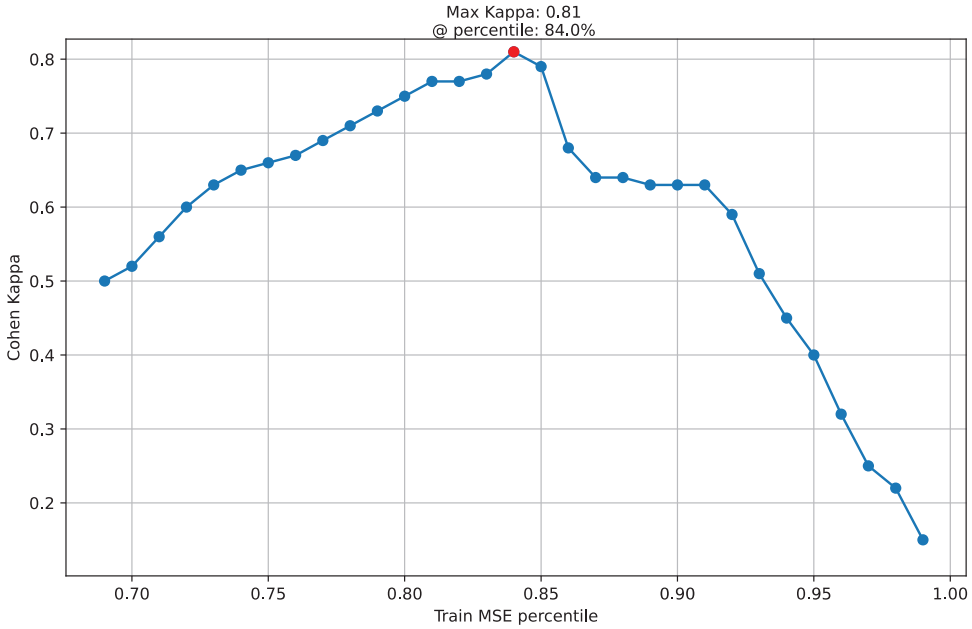

Figure 7: Optimizing Cohen's Kappa coefficient relative to training data MSE percentiles

To derive these results, we identified the optimal parameters for evaluation as follows: the model was evaluated using a centralized setting with 10 epochs, a batch size of 32, and a validation set comprising 5% of the original training dataset's size.

5.5 FL Process Analysis

For the FL process evaluation, 10 agents will be considered, each one containing a portion of one-tenth the size of the preprocessed training and testing datasets, as described in Section 5.2. The step-by-step methodology for setting up the FL process is as follows:

1. Data Distribution: Each of the 10 agents is assigned an equal portion of the preprocessed dataset. This ensures a fair and balanced distribution of data across all participating nodes.
2. Baseline Setup: A centralized learning scenario is established as described in Section 5.4, serving as a baseline for comparison. This involves aggregating the entire dataset at a central location and training a global model.
3. FL Configuration: For all FL executions, we employ the *FedAvg* aggregation algorithm. Parameters such as the learning rate and batch size are standardized across all agents.
4. Execution Rounds: The FL process is configured to execute 100 rounds, with each agent performing one local epoch per round in one configuration and five local epochs per round in another. The process is repeated three times to ensure the reliability of the results.

In Figs. 8 and 9, the recall, specificity, balanced accuracy, and F1 score for an increasing number of agents (from 2 to 10) are shown. In Fig. 8, the results correspond to the FL process being configured to execute 100 rounds and 1 local epoch per agent and round. In Fig. 9, the results correspond to the same number of rounds but in this case, 5 local epochs are executed. From the results, on one hand, it can be concluded that it is not too much increasing the number of agents that matters, even though new data is added with each new agent participating, but rather elevating the number of local epochs executed per

round. The model performance metrics for the latter scenario, i.e., 5 local epochs, are closer to the ones achieved in the baseline, i.e., centralized, scenario. However, in terms of specificity, which measures the ability of the model to correctly identify negative instances among all the actual negative instances, i.e., benign samples, we find that the federated scenario (for both 1 and 5 local epochs settings) improves the baseline centralized scenario, however, it is not either impacted by raising the number of agents in any case.

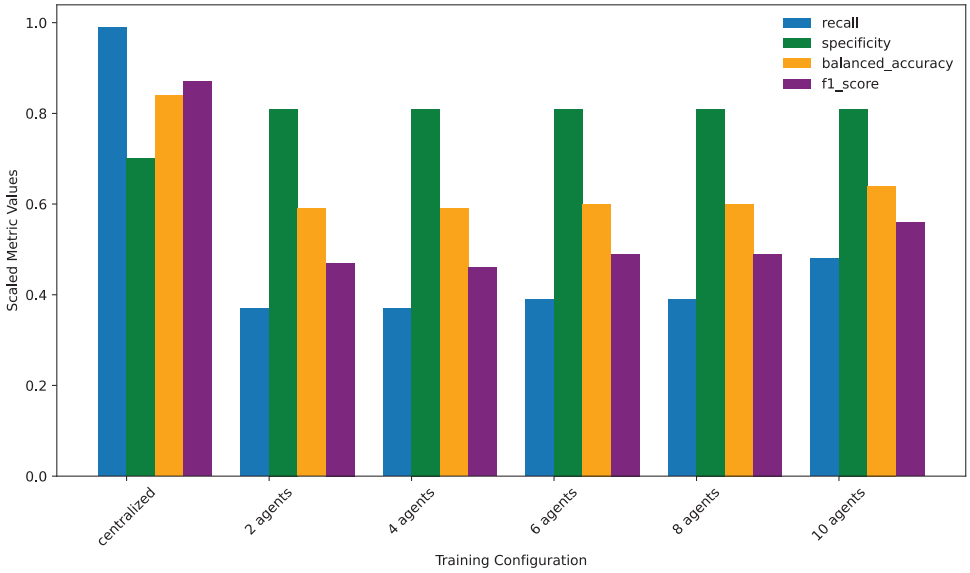

Figure 8: Comparison of evaluation metrics for different FL training configurations, with 100 FL rounds executed and 1 local epoch per round

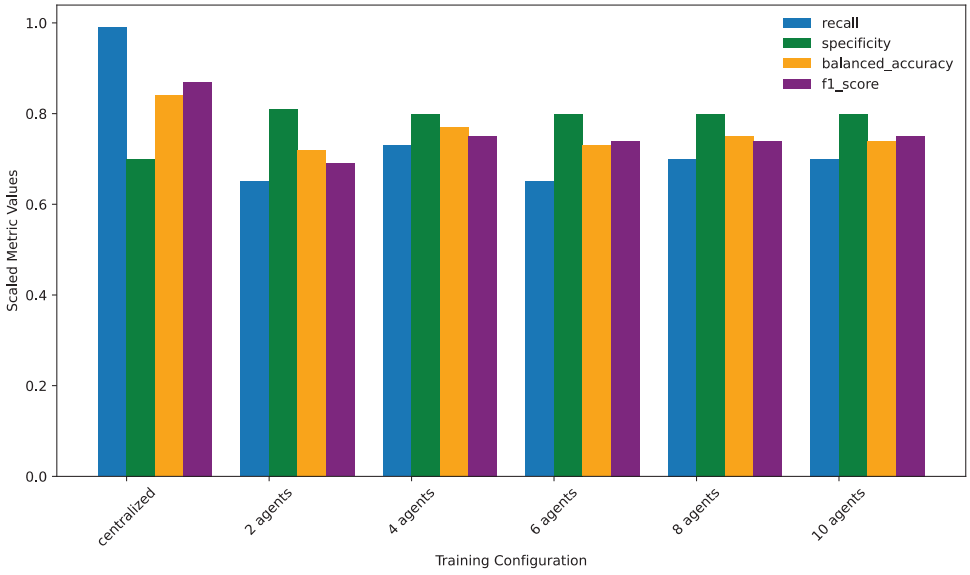

Figure 9: Comparison of evaluation metrics for different FL training configurations, with 100 FL rounds executed and 5 local epochs per round

Moreover, from another perspective, Fig. 10 shows perfectly that increasing the number of local epochs executed per round from 1 to 5 has a significant impact on the balanced accuracy metric, which is considered highly relevant as it is based on the mean of specificity and recall metrics. The difference is notable for all training configurations, with some ones, e.g., the 4-agent setting, benefiting slightly more than the others (from 0.59 with 1 epoch executed to 0.75 with 5 epochs executed).

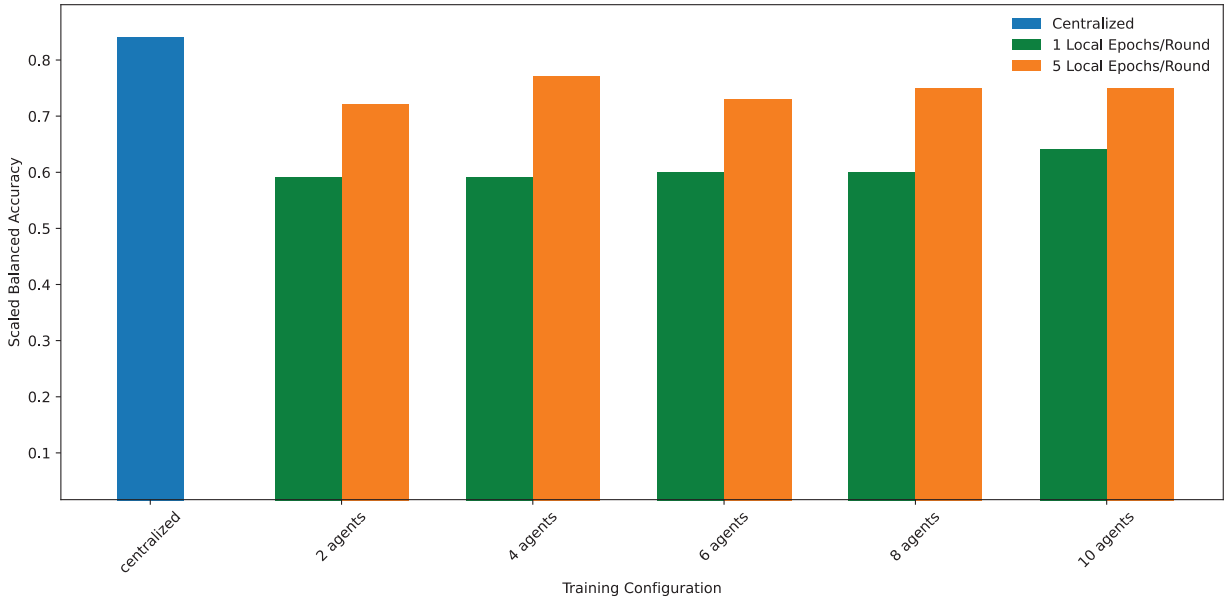

Figure 10: Comparison (using balanced accuracy) of different FL training configurations (increasing number of agents), and different number of local epochs executed per round (100 total rounds)

The balanced accuracy in Fig. 10 serves as a metric, once the ideal percentile maximizing the Cohen's Kappa has been obtained in Fig. 7, to evaluate the performance between the two approaches: centralized and federated. The best configuration for federated with a total of 5 local epochs is given by 4 agents, providing an accuracy of 0.77 *vs*. 0.84 for centralized, which is a decrease in performance of 8.33% in terms compared to the centralized approach. This indicates that, although the federated approach may offer advantages in terms of data privacy and decentralization, its accuracy in this configuration is slightly lower. However, this difference could be considered acceptable in contexts where the privacy and decentralization benefits outweigh the need for maximum accuracy.

Also, in terms of mean-squared-error evolution throughout FL rounds, Fig. 11 points out that, for a fixed number of agents (10), the model's convergence speed, i.e., reaching lower MSE values at earlier rounds, is higher on the 5 local epochs setting compared to the 1 local epoch setting. However, from Fig. 12, it can be concluded that for a fixed number of local epochs executed per round (5), there are no significant advantages in terms of speed convergence when adding more agents to the federated training process, although some configurations, i.e., the 6-agent configuration, seem to have better results compared to the rest. Still, all configurations (from using 2 agents to 10), appear to significantly evolve the same way, starting with around MSE = 0.975 at the first training round, ending with around MSE = 0.835 at the final round, and showing off a very similar pace (MSE being roughly 0.875 at the middle of the process, i.e., round 50), except for the 6-agent configuration.

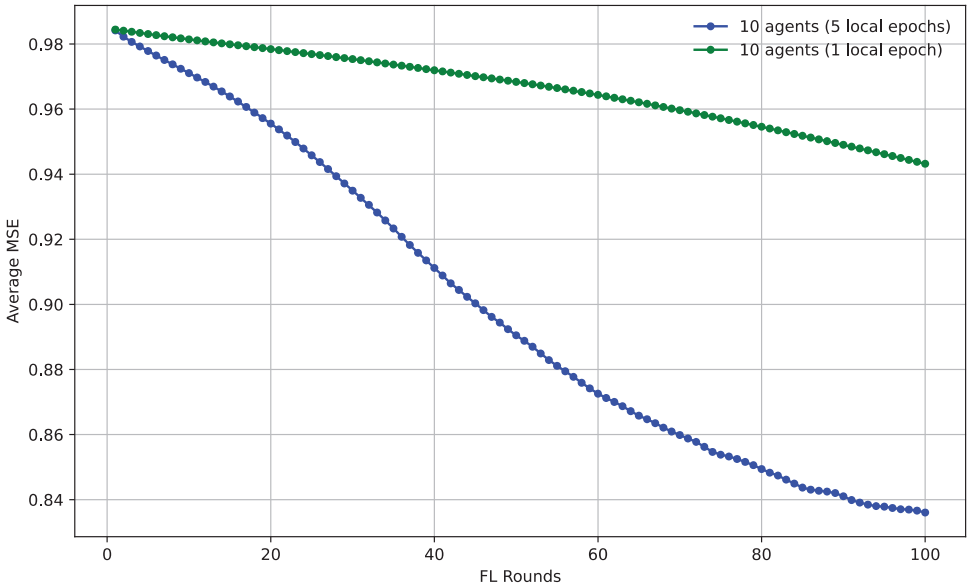

Figure 11: Evolution of model loss throughout 100 rounds of FL training with a fixed number of agents (10) but varying the local epochs executed (1 *vs.* 5), the model loss being the mean across agents for each round

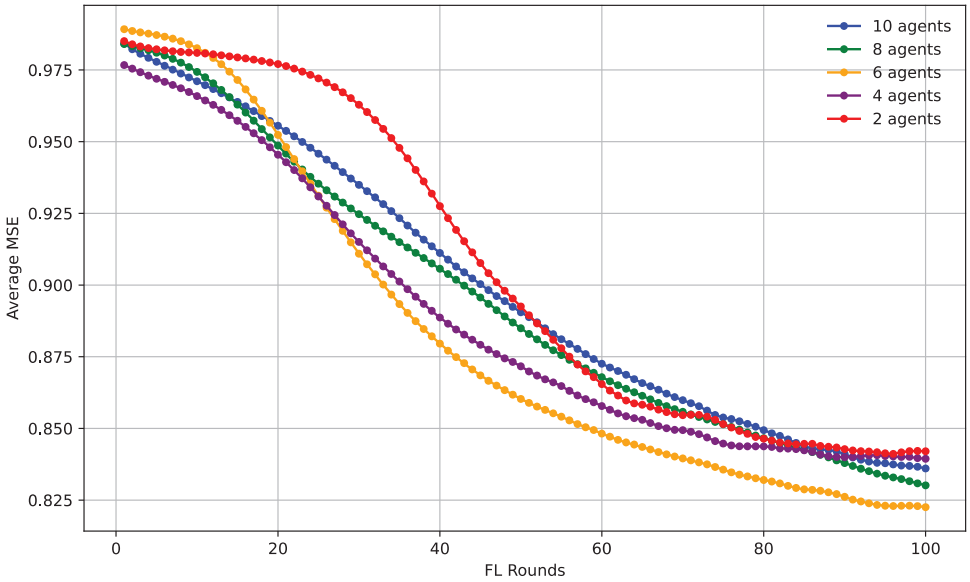

Figure 12: Evolution of model loss throughout 100 rounds of FL training (increasing number of agents) with a fixed number of local epochs per round (5), being the model loss the mean across agents for each round

Lastly, from the training time perspective, Fig. 13 shows the time taken, in seconds, by the centralized scenario (10 epochs), and by each FL setting varying the number of agents and the local epochs executed per round. As can be concluded, the FL scenario in any of its versions outperforms the centralized one, as it is logical since the training computation is distributed among the agents and not

centralized in a single point. Also, executing 5 local epochs in each agent per round in the FL scenario adds a bit of delay compared to the 1 local epoch setting, but it is considered not so significant, so we conclude that it is worth increasing the number of local epochs executed given the clear advantages in terms of almost all model performance metrics, as shown in Figs. 8 and 9.

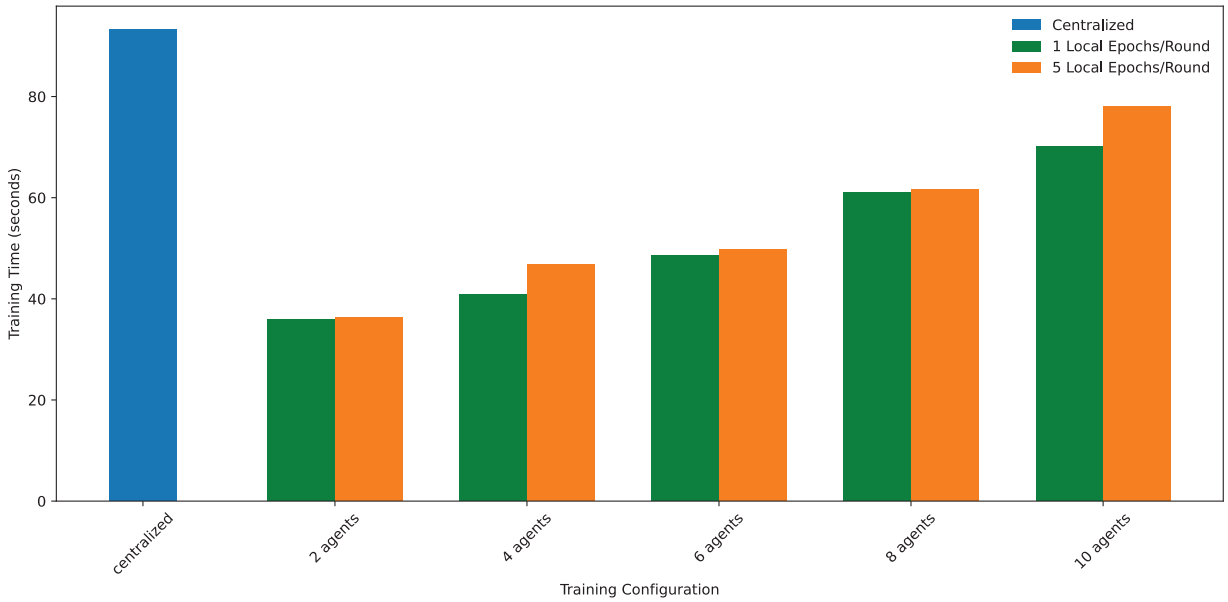

Figure 13: Comparison of total time spent in the training phase using different FL training configurations (increasing number of agents), and different numbers of local epochs executed per round (100 total rounds)

To compare with existing works in this regard, we analyzed the approaches and methodologies of three key papers in the field of anomaly detection. Kim et al. [19] focus on the application of a one-class Support Vector Machine (SVM) model, Seth et al. [20] evaluate the use of a Light Gradient-Boosting Machine (LightGBM) model, and Ferrag et al. [21] discuss the implementation of a Deep Autoencoder model. Each study employs these techniques on large and complex datasets—NSL-KDD, CSE-CIC-IDS2018, and CSE-CIC-IDS2018, respectively—offering a broad perspective on the performance and adaptability of different models in detecting network anomalies. These references provide a foundational context for assessing the efficacy of various computational approaches to anomaly detection model training. A comparison of these works and ours is shown in Table 1, considering different aspects such as the model used, dataset, dataset size, and processing time per sample, was obtained by dividing the total training time by the dataset size. Since not all studies detail the specific number of samples used for training, the size of the original dataset is considered.

The current configuration using a Deep Autoencoder on the 5G-NIDD dataset shows a considerable performance advantage in terms of training efficiency. With a time per sample of just 2.2 µs, it outperforms the other referenced configurations, omitting the details of the underlying hardware used, since they are not specified for most of the analyzed studies. This notably faster per-sample processing time can be attributed to both the efficiency of the Deep Autoencoder model and the effectiveness of its tuning for the dataset. While the One-Class SVM and LightGBM models from Kim et al. [19] and Seth et al. [20] show slower per-sample times of 11 and 4.5 µs, respectively, while close processing times are obtained by Ferrag et al. [21] since a very similar model is used. Therefore, it is evident that

the actual configuration not only handles a large dataset (∼1.2 M samples) efficiently but does so in a way that minimizes computational overhead. This could potentially translate to lower operational costs and faster turnaround times for tasks involving large-scale data processing, making it a superior choice in scenarios where time and resource efficiency are paramount.

Table 1: Comparison between our anomaly detection approach and other state-of-the-art approaches

Training time (avg. best config)	Model used	Dataset	Dataset size	Time per sample (seconds)
Our approach	Deep autoencoder	5G-NIDD	∼1.2 M samples	2.2 µs
Kim et al. [19]	One-class SVM	NSL-KDD	∼5 M samples	11 µs
Seth et al. [20]	LightGBM	CSE-CIC-IDS2018	∼10 M samples	4.5 µs
Ferrag et al. [21]	Deep autoencoder	CSE-CIC-IDS2018	∼10 M samples	2.8 µs

Moreover, while the proposed FL approach for anomaly detection in B5G networks demonstrates promising results, several limitations and challenges were encountered during the experimental validation process, such as uneven data distribution across agents, scalability concerns in larger networks, synchronization overhead, and the model's generalization ability to various network conditions and attack types. Addressing these considerations is essential for a holistic understanding of the solution's applicability and areas for future improvement.

5.6 Real-Time Anomaly Detection Process Analysis

For the real-time anomaly detection process analysis, we tested the real-time anomaly detection engine, once we received the model previously trained by the FL agents, by forwarding to it real-time samples, i.e., traffic flow features, for evaluation. Then, we measured how much it adds to the process in terms of latency for each stage of the evaluation of the new incoming flow. These stages comprise the following:

1. **Decoding stage**, in which the flow is received and decoded from the data services, simulated in this case using Apache Kafka and a topic for this purpose.
2. **Preprocessing stage**, where the sample is converted into a one-row data frame, the unnecessary features are removed so they are consistent with those used during training and normalized with a scaler previously fitted on the testing phase.
3. **Prediction stage**, where the sample is predicted against the model, the MSE is obtained and checked against the anomaly threshold, so it is decided whether it is an anomalous sample or not.

These methodical steps aim to ensure that the real-time anomaly detection process is both transparent and reproducible.

In Table 2, the time results, in milliseconds, for each stage described above are shown. As can be seen, a smaller part of the time is spent in the decoding stage, while the majority is spent in the prediction stage, which is logical since the interaction of the model should be the most time-consuming task. In total, as can be consulted in the last row of the table, the average time taken by the whole process is 50.792 milliseconds, which we consider to be a reasonable latency and would not impact the real-time requirement significantly.

Table 2: Time spent per flow (in milliseconds) in each evaluation stage (decoding, preprocessing, prediction), as well as the total time (sum of the previous stages)

Stage	Min. latency (ms)	Max. latency (ms)	Avg. latency (ms)	Std. latency (ms)
Decoding	0.022	0.724	0.044	0.049
Preprocessing	1.976	19.894	2.604	1.054
Prediction	35.844	229.133	48.144	12.486
Total	37.842	249.751	50.792	13.589

To the best of our knowledge, this work is the only one in the current state-of-the-art that provides this exhaustive analysis of the real-time anomaly detection phase processing times, considering that samples are being received in real-time from the monitoring agents in the network through a message broker like Kafka, although the monitoring part has been simulated for the sake of simplicity.

5.7 FL Security Orchestration and Enforcement

For validating FL security orchestration and enforcement features, multiple Kubernetes nodes were instantiated. To simulate a multi-domain environment, we deployed ten different far-edge nodes at different locations. The process involved in setting up the environment and conducting the experiments is outlined in the following steps:

1. **Requesting FL Features:** The experiments start by requesting FL features, comprising two security enablers—the FL agent and the FL aggregator—from the Security Orchestrator. The Orchestrator assesses and determines the most suitable allocation for these components based on security requirements.
2. **Integration of Security Protocols:** Security protocols are integrated and enforced by default through our security orchestrator to protect the deployment of agents and aggregators. This protection includes internal-only access within the Kubernetes cluster, blocking external access, and employing trust scoring algorithms to select deployment nodes, giving preference to those with the highest trust scores for enhanced system reliability and security.
3. **Validation of Extensibility and Scalability:** Different amounts of security policies are provided to validate the framework's extensibility and scalability. This involves the enforcement of 10 agents at different levels of the far edge, and one aggregator at the edge. This is: $\forall a \in \text{Agents}, \forall \text{fen} \in \text{Far} - \text{Edge} - \text{Nodes}, \exists$ *a such as a is deployed on fen*. Each experiment was repeated 30 times.
4. **Deployment and Configuration Process Assessment:** The process assesses deployment and configuration strategies within the management orchestration framework, differentiating between the deployment, configuration, and enforcement phases (a, b):
   a. Deployment Phase: Involves identifying a suitable NFV-MANO with adequate resources and triggering the NFV deployment. The update of the system model with information regarding the new deployment is also required.
   b. Configuration Phase: Gathers the required FL configurations according to security requirements. Then, a policy enabler chain that contains information regarding the required deployments and configurations is created to determine how the enforcement must proceed.

5. **Enforcement Phase:** Utilizing the knowledge acquired in the allocation deployment process, enforce the order into the NFV-MANO to deploy the enablers and/or use the communication driver of the different enablers for applying the final configurations if required.

In this proof of concept, we only deploy one aggregator, considering that it is enough for aggregating results with the maximum number of agents, each located at a different far-edge node in single and multi-domain environments. However, multiple aggregators could be deployed if required. In addition, the decision to deploy a single agent per far-edge, with a total of ten agents at ten far-edges, is considered to account for real-world applicability but could deploy as many agents as we have the capacity for.

Fig. 14 illustrates the resolution of the allocation problem, represented in seconds, for increasing the number of agents and using a single aggregator. This stage is critical as it encompasses the identification of viable NFV-MANO instances across the infrastructure, followed by the construction of a graph. This graph is then analyzed using an Integer Linear Programming (ILP) algorithm, which takes into consideration policy constraints related to the location of the different edge/far edge. Upon identifying a suitable NFV-MANO, the process ends by verifying that the NFV-MANO can deploy the enabler via Helm Charts. A notable distinction in the deployment *vs*. configuration in the latter phase requires only the identification of an enabler candidate, which streamlines the configuration process significantly. As seen in Fig. 14, the time required to solve the allocation placement for deployment is higher than that for configuration, and this is due to two reasons. The first one is that during the deployment phase, the focus is on identifying the most suitable site for deploying the enabler. The second one, in this same phase, is verified that the NFV MANO has an enabler in his Helm Chart repositories, either of the Aggregator or Agent type.

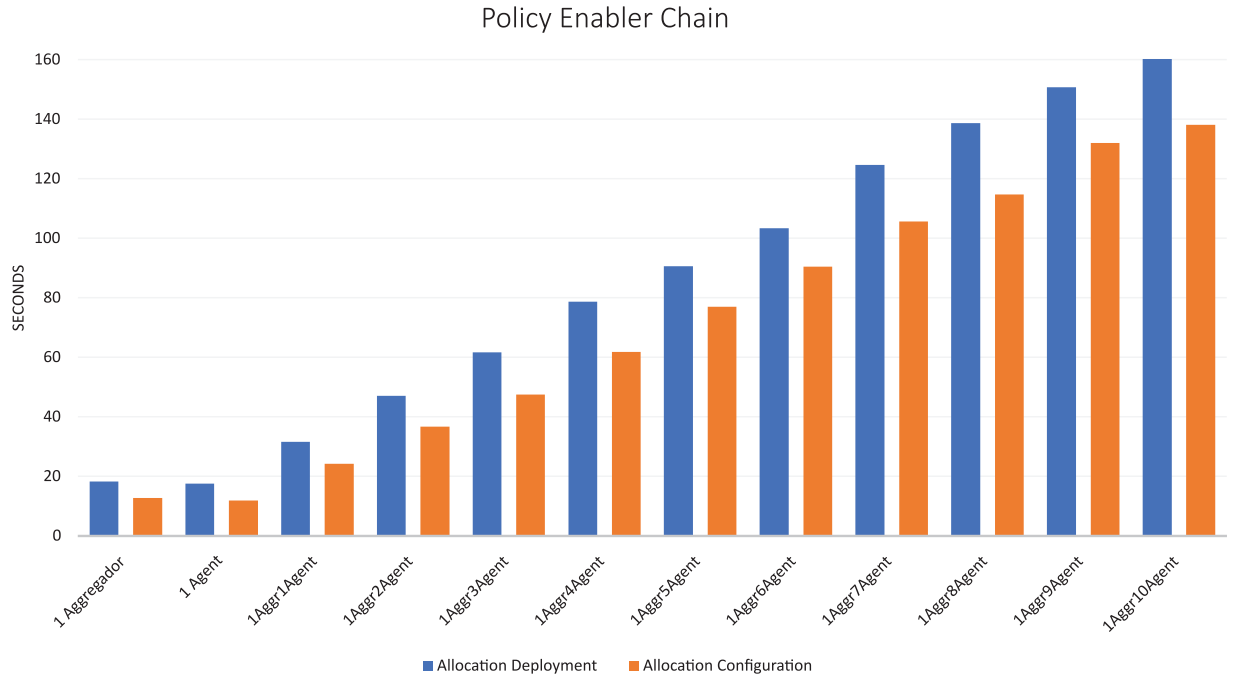

Figure 14: Allocation placement problem resolution times (in seconds). Blue bars represent the decision time for allocation deployment activities, while orange bars represent the time for allocation configuration activities

In the detailed analysis of the enforcement process, as illustrated in Fig. 15, a significant distinction is observed between deployment and configuration times. While configurations are completed in less than a second, deployment time increases from around 30 s for one deployment to several minutes when the number of required deployments increases. However, we consider the results promising compared to previous efforts, such as Asensio-Garriga et al. [7], where a single security enabler takes around 40 s to configure and more than a minute to deploy. It is important to note that within this context, the ILP algorithm used to solve the assignment problem demonstrates remarkable efficiency, taking an average of only 0.03 s to process decisions using 10 nodes. This means that for this use case most of the time spent does not depend on the decision algorithm to find the most suitable node, but on other processes such as: data extraction from databases, graph construction, selection of the appropriate algorithm type, verification of HELM repositories, and conflict detection, among others. In comparison, the ILP algorithm described by Kim et al. [22] requires approximately one second to handle a similar number of nodes, despite incorporating a smaller number of variables. Our algorithm not only considers standard variables such as CPU, memory, disk, and latency/network, but also integrates dynamic constraints and improves the connection between nodes. Although Sasabe et al. [23] pursues a different objective, focused on route optimization using ILP, the processing times are comparable. This underscores the robustness and efficiency of our ILP algorithm compared to similar methodologies in the field. Nevertheless, the significant time difference between configuration and deployment is largely due to the sequential methodology adopted for both deployments and configurations, meaning they are executed consecutively rather than simultaneously. This sequential approach emerges as a critical area for optimization. The ability to execute multiple deployments and configurations concurrently could revolutionize the process, reducing the time required for deployments to a few seconds. Another aspect identified during experimentation, which could enhance security, involves incorporating encryption protocols in communications between agents and the aggregator.

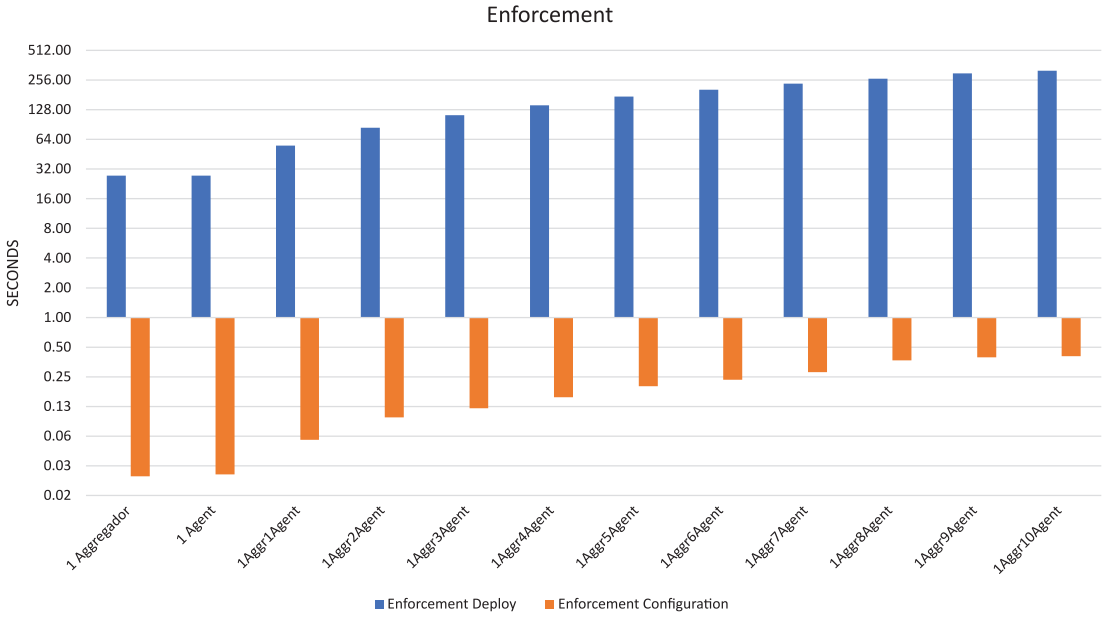

Figure 15: Enforcement process processing times (in seconds). Blue bars represent the processing time for deployments, while orange bars represent the processing time for configurations

6 Conclusions

This paper presented an approach for orchestrating Network Intelligence in B5G networks driven by an FL architecture, that overcomes current challenges of security orchestration, automation, management, and interoperability that undermine the applicability of FL in B5G Networks. The INSPIRE-5Gplus architecture has been leveraged to embrace the FL components aimed at realizing ubiquitous AI. In addition, the paper proposes a proactive and reactive scalable and dynamic orchestration process aimed at managing and deploying FL agents and AI-based Anomaly Detection engines that use the model generated previously to perform real-time detection. Moreover, an interoperable novel FL policy model, conceived to describe the behavior of the FL agent has been designed. The proposed FL architecture can be deployed automatically and efficiently in the B5G Continuum through a hierarchy of Network Intelligence orchestrators to handle anomaly and security threats in real-time. Lastly, the orchestration, FL, and real-time anomaly detection processes have been evaluated over a realistic testbed and using the recent (December 2022) 5G-NIDD dataset to emulate traffic monitoring for training and detection purposes, analyzing some key performance metrics considering different scenarios, configurations, and conditions. The results have shown that our proposals for either the orchestration, training, and detection introduce minimal latency to the normal network function while enabling an automatic and near real-time way to handle all the stages of an anomaly or attack taking place at a certain moment, from real-time network monitoring to the application of countermeasures for mitigation. As future work, it is envisaged that FL can be employed not only to detect anomalies/attacks in the network but also to infer real-time cognitive decisions about countermeasures to be taken at each moment to mitigate ongoing cyberattacks in the network. Likewise, substantial research work needs to be accomplished to reduce attack surface against the FL systems and deployments themselves, where certain FL agents deployed in potential untrusted network subdomains (e.g., volatile, or nomadic network sub-domains) might try to perpetrate poisoning attacks to degrade the FL model performance and hamper the FL model convergence.

Acknowledgement: This paper and the research behind it would not have been possible without the financial support of CERBERUS projects and the Department of Information and Communications Engineering (DIIC) of the University of Murcia.

Funding Statement: This work was supported by the grants: PID2020-112675RBC44 (ONOFRE-3), funded by MCIN/AEI/10.13039/501100011033; Horizon Project RIGOUROUS funded by European Commission, GA: 101095933; TSI-063000-2021-{36, 44, 45, 62} (Cerberus) funded by MAETD's 2021 UNICO I+D Program.

Author Contributions: Pablo Fernández Saura: analysis and interpretation of results, draft manuscript preparation. **José Manuel Bernabé Murcia:** analysis and interpretation of results, draft manuscript preparation. **Emilio García de la Calera Molina:** analysis and interpretation of results, draft manuscript preparation. **Alejandro Molina Zarca:** analysis and interpretation of results, draft manuscript preparation. **Jorge Bernal Bernabé:** analysis and interpretation of results, draft manuscript preparation. **Antonio Fernando Skarmeta Gómez:** analysis and interpretation of results, draft manuscript preparation. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: For this study, no proprietary datasets were created or analyzed. Instead, the analysis was conducted using the publicly available 5G-NIDD dataset, which has been duly cited in the references section of this document. This ensures transparency and enables other

researchers to validate the findings presented herein by accessing the same dataset. Details regarding the dataset can be found in the referenced literature.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat.*, Apr. 2017, vol. 54. pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf.

[2] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 20, Jul. 2019. doi: 10.1186/s42400-019-0038-7.

[3] M. Injadat, A. Moubayed, A. Nassif, and A. Shami, "Multi-stage optimized machine learning framework for network intrusion detection," *IEEE Trans. Netw. Serv. Manag.*, no. 2, pp. 1, Aug. 2020. doi: 10.1109/TNSM.2020.3014929.

[4] L. Fernández-Maimó, A. Gómez, F. Clemente, M. Pérez, and G. Pérez, "A self-adaptive deep learning-based system for anomaly detection in 5G networks," *IEEE Access*, vol. 6, pp. 7700–7712, 2018. doi: 10.1109/ACCESS.2018.2803446.

[5] A. Thantharate, "FED6G: Federated chameleon learning for network slice management in beyond 5G systems," presented at the Proc. 2022 IEEE 13th Ann. Inf. Technol. Electron. Mobile Commun. Conf. (IEMCON), Oct. 12–15, 2022, pp. 19–25. doi: 10.1109/IEMCON56893.2022.9946488.

[6] P. Li, Y. Zhong, C. Zhang, Y. Wu, and R. Yu, "FedRelay: Federated relay learning for 6G mobile edge intelligence," *IEEE Trans. Veh. Technol.*, vol. 72, no. 4, pp. 5125–5138, 2023. doi: 10.1109/TVT.2022.3225087.

[7] R. Asensio-Garriga *et al.*, "ZSM-based E2E security slice management for DDoS attack protection in MEC-enabled V2X environments," *IEEE Open J. Veh. Technol.*, no. 2, pp. 1–12, Jan. 2024. doi: 10.1109/OJVT.2024.3375448.

[8] A. Hermosilla, A. Molina-Zarca, J. Bernal-Bernabé, J. Ortiz, and A. Skarmeta, "Security orchestration and enforcement in NFV/SDN-aware UAV deployments," *IEEE Access*, vol. 8, pp. 131779–131795, 2020. doi: 10.1109/ACCESS.2020.3010209.

[9] E. Mármol-Campos *et al.*, "Evaluating FL for intrusion detection in Internet of Things: Review and challenges," *Comput. Netw.*, vol. 203, no. 3, pp. 108661, 2022. doi: 10.1016/j.comnet.2021.108661.

[10] M. J. Idrissi *et al.*, "Fed-ANIDS: FL for anomaly-based network intrusion detection systems," *Expert Syst. Appl.*, vol. 234, no. 1, pp. 121000, 2023. doi: 10.1016/j.eswa.2023.121000.

[11] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Commun. Surv. Tutor.*, vol. 23, no. 2, pp. 1342–1397, 2021. doi: 10.1109/COMST.2021.3058573.

[12] S. H. Alsamhi, A. V. Shvetsov, A. Hawbani, S. V. Shvetsova, S. Kumar and L. Zhao, "Survey on federated learning enabling indoor navigation for industry 4.0 in B5G," *Future Gener. Comput. Syst.*, vol. 148, no. 4, pp. 250–265, 2023. doi: 10.1016/j.future.2023.06.001.

[13] T. Zhao, F. Li, and L. He, "DRL-based joint resource allocation and device orchestration for hierarchical FL in NOMA-enabled industrial IoT," *IEEE Trans. Ind. Inform.*, vol. 19, no. 6, pp. 7468–7479, 2022. doi: 10.1109/TII.2022.3170900.

[14] M. Camelo *et al.*, "Requirements and specifications for the orchestration of network intelligence in 6G," presented at the 2022 IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC), Jan. 8–11, 2022, pp. 1–9. doi: 10.1109/CCNC49033.2022.9700729.

[15] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network automation through orchestrated intelligence in the open RAN," presented at the IEEE INFOCOM 2022—IEEE Conf. Comput. Commun., May 2–5, 2022, pp. 270–279. doi: 10.1109/INFOCOM48880.2022.9796744.

[16] D. Bega, M. Gramaglia, R. Perez, M. Fiore, A. Banchs and X. Costa-Pérez, "AI-based autonomous control, management, and orchestration in 5G: From standards to algorithms," *IEEE Netw.*, vol. 34, no. 6, pp. 14–20, 2020. doi: 10.1109/MNET.001.2000047.

[17] P. Ruzafa-Alcázar *et al.*, "Intrusion detection based on privacy-preserving FL for the industrial IoT," *IEEE Trans. Ind. Inform.*, vol. 19, no. 2, pp. 1145–1154, 2021. doi: 10.1109/TII.2021.3126728.

[18] S. Samarakoon *et al.*, "5G-NIDD: A comprehensive network intrusion detection dataset generated over 5G wireless network," *IEEE Dataport*, 2022. doi: 10.21227/xtep-hv36.

[19] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1690–1700, 2014. doi: 10.1016/j.eswa.2013.08.066.

[20] S. Seth, G. Singh, and K. Chahal, "A novel time efficient learning-based approach for smart intrusion detection system," *J. Big Data*, vol. 8, Aug. 2021. doi: 10.1186/s40537-021-00498-8.

[21] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *J. Inf. Secur. Appl.*, vol. 50, no. 1, pp. 102419, 2020. doi: 10.1016/j.jisa.2019.102419.

[22] S. I. Kim and H. S. Kim, "A VNF placement method based on VNF characteristics," presented at the 2021 Int. Conf. Inf. Netw. (ICOIN), Jeju Island, Republic of Korea, Jan. 13–16, 2021, pp. 864–869. doi: 10.1109/ICOIN50884.2021.

[23] M. Sasabe and T. Hara, "Capacitated shortest path tour problem-based integer linear programming for service chaining and function placement in NFV networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no.1, pp. 104–117, Mar. 2021. doi: 10.1109/TNSM.2020.3044329.