Federated Learning in the Wild: A Comparative Study for Cybersecurity under Non-IID and Unbalanced Settings

Roberto Doriguzzi-Corin^a, Petr Sabel^b, Silvio Cretti^a, Silvio Ranise^{a,b}

^aCenter for Cybersecurity, Fondazione Bruno Kessler, Italy ^bUniversity of Trento, Italy

Abstract

Machine Learning (ML) techniques have shown strong potential for network traffic analysis; however, their effectiveness depends on access to representative, up-to-date datasets, which is limited in cybersecurity due to privacy and data-sharing restrictions. To address this challenge, Federated Learning (FL) has recently emerged as a novel paradigm that enables collaborative training of ML models across multiple clients while ensuring that sensitive data remains local.

Nevertheless, Federated Averaging (FedAvg), the canonical FL algorithm, has proven poor convergence in heterogeneous environments where data distributions are non-independent and identically distributed (i.i.d.) and client datasets are unbalanced, conditions frequently observed in cybersecurity contexts. To overcome these challenges, several alternative FL strategies have been developed, yet their applicability to network intrusion detection remains insufficiently explored.

This study systematically reviews and evaluates a range of FL methods in the context of intrusion detection for DDoS attacks. Using a dataset of network attacks within a Kubernetes-based testbed, we assess convergence efficiency, computational overhead, bandwidth consumption, and model accuracy. To the best of our knowledge, this is the first comparative analysis of FL algorithms for intrusion detection under realistic non-i.i.d. and unbalanced settings, providing new insights for the design of robust, privacy-preserving network security solutions.

Keywords: Federated Learning, Cybersecurity, Heterogeneous Data, Unbalanced Datasets

1. Introduction

As reliance on digital infrastructure grows, so does the scale and complexity of cybersecurity threats. Attacks such as data breaches, Distributed Denial of Service (DDoS) incidents, phishing campaigns, and financial fraud are becoming increasingly sophisticated, leveraging the expanded attack surface that now surrounds every connected device and user. With cybercriminals constantly evolving their techniques, traditional rule-based security systems often suffer in identifying and mitigating novel or stealthy attack patterns.

To address these challenges, cybersecurity experts must engage in continuous research and threat analysis. This includes studying detailed attack behaviours, developing detection signatures, and maintaining extensive documentation to support the identification and mitigation of emerging threats. In addition, real-time monitoring of modern infrastructures is essential for detecting anomalies that may indicate malicious activity. However, such tasks are complex and resource-intensive, which has driven significant interest in automated solutions to improve efficiency. In this context, Machine Learning (ML) has shown strong effectiveness across diverse application domains, including autonomous vehicles [1], healthcare [2], Industrial Control Systems (ICSs) [3, 4] and maritime systems [5] among others, making it a promising tool for cybersecurity. Nevertheless, its adoption poses significant challenges, particularly due to strict privacy concerns and data-sharing limitations.

Federated Learning (FL) [6] has emerged as a promising approach for collaborative model training while preserving data privacy. In this framework, sensitive information stays on the local devices (clients) and is not directly shared. Instead, clients iteratively train on their private datasets and exchange only model updates with a central server. These updates are then aggregated to create a global model, which is progressively refined over multiple communication rounds. The ultimate goal is to enable clients to learn a common ML model that benefits from the diversity and scale of distributed data, all while respecting the confidentiality of each client's data.

Despite its potential, FL, particularly with algorithms like Federated Averaging (FedAvg), faces significant challenges related to model performance and training convergence. These issues are primarily a result of the heterogeneous and diverse nature of client data. In real-world scenarios, data across devices is often non-independent and identically distributed (i.i.d.), meaning local datasets can have significant differences in feature distributions, label proportions, or sampling biases (in contrast to centralised learning, where data is assumed to be i.i.d., meaning it's all from the same distribution). For instance, in a security application, some clients might primarily encounter specific types of attacks, while others see entirely different patterns.

Regarding training convergence, the heterogeneity of client data can lead to model drift, where local models diverge as they adapt to their unique datasets, making it harder for the global aggregation step to converge to an optimal solution. In terms of performance, poor generalisation may arise when the global model becomes overly tailored to participants contributing large volumes of data, while those providing only limited samples are underrepresented. This imbalance introduces a bias toward data-rich clients and further undermines generalisation across the federation.

These problems are particularly pronounced in the security domain, where clients may observe highly diverse traffic patterns and feature distributions for both benign and malicious activities. For example, some clients may predominantly face specific types of attacks or variations of normal traffic, while others may record entirely different behaviours. This difference in feature distributions not only exacerbates the non-i.i.d. nature of the data but also makes it challenging for the global model to learn a representation that generalises well across all participants. In addition, the rarity of certain attack types can lead to severe class imbalance, further complicating the training process and increasing the risk that the global model underperforms on less frequently observed threats.

To address the problems related to model performance and training convergence, researchers have developed various techniques, each targeting a specific problem. For instance, Fed-Prox [7] introduces a proximal regularization term that constrains local models from diverging from the global model, directly tackling the model drift issue. SCAFFOLD [8] mitigates model drift by using control variates, where the server sends a correction value to each client to align their updates with the global objective before aggregation. Other approaches, such as FedALA [9], enhance global model generalisation on client-specific data through weighted aggregation of the local and global models. In the security domain, where data diversity is a major issue, methods like FedSBS [10], DAFL [11], and FLAD [12, 13] employ alternative client selection mechanisms to ensure that a more diverse and representative group of clients participates in each training round. This directly addresses the model performance problem and also helps mitigate client non-participation by ensuring a more robust and reliable training process. Despite these advancements, there's still no single consensus on the most effective strategies, highlighting a critical gap in current research.

This work reviews recent studies that implement various FL variants and evaluates their applicability in the context of cybersecurity network traffic analysis, specifically focusing on DDoS detection. While some of the algorithms were originally designed for tasks unrelated to cybersecurity, one of our primary objectives is to adapt these methods and satisfy the requirements of Network Intrusion Detection Systems (NIDSs). Our evaluation focuses on key metrics such as convergence time, resource consumption, and final model accuracy, enabling a comparative analysis to assess and rank the different algorithms. The temporal dimension of learning is especially critical in this context, as NIDSs must rapidly incorporate updates related to newly observed attack profiles in order to remain effective against evolving threats. To ensure reproducibility and scalability, we design a virtualised experimental environment based on a Kubernetes cluster, which simulates a realistic distributed

setting for FL. Each node in the cluster acts as an independent client, ensuring proper data and resource isolation across participants. This configuration mimics the deployment of FL systems in a real-world scenario where clients operate on separate devices or infrastructures. As a data source, we utilise the CIC-DDoS2019 dataset [14], a recent benchmark that includes both benign traffic and 13 distinct types of DDoS attack patterns. To evaluate the algorithms under challenging conditions, we design a scenario in which each client is exposed to a single DDoS attack type and an unbalanced volume of data. This is called a pathological setting [6] and it represents a worst-case scenario adopted to reflect the heterogeneity commonly observed in practical cybersecurity applications, where clients may have access to very different data distributions.

In summary, the contributions of this work are the following:

- Review and adaptation of FL methods: this paper reviews a set FL algorithms, some of them originally designed for other domains, and adapts them to the specific requirements of NIDSs for DDoS detection.
- Comparative evaluation of such algorithms in terms of overall duration of the FL process, local training time, network bandwidth overhead and final model accuracy.
- Realistic experimental environment: A virtualised FL testbed is implemented using a Kubernetes cluster, where each client runs in isolation, simulating realistic distributed deployments. The source code of the environment is publicly available for evaluation and testing [15].
- Challenging evaluation scenario: non-i.i.d. and unbalanced settings, where each client faces a unique attack profile and unbalanced data volumes, reflecting the heterogeneity typical in real-world cybersecurity contexts.

The remainder of this paper is organised as follows. Section 2 analyses related work. Section 3 outlines the limitations of FedAvg in heterogeneous FL cybersecurity scenarios, while Section 4 reviews the main properties of the other algorithms benchmarked in this study. Section 5.2 details the dataset employed to emulate a non-i.i.d. and unbalanced FL scenario. Section 5 provides a comprehensive description of the experimental setup, including the evaluation environment and the hyperparameters used to configure the algorithms. Section 6 presents the results of the comparative evaluation. Finally, Section 7 summarises the findings and concludes the study.

2. Related work

Although a few studies have compared FL algorithms in the computer vision domain, our work provides additional insights by focusing on a security-specific context. Most existing research evaluates new algorithms against standard baselines, such as FedAvg, with some studies extending the comparison to a few additional methods [16, 17]. In contrast, we broaden this scope by benchmarking a wider set of FL algorithms within the NIDS domain, with the goal of enhancing malicious traffic detection. Notably, our evaluation also includes methods specifically designed for NIDS, such as DAFL [11], FedSBS [10], and FLAD [12].

Recent literature often uses the final model accuracy after a fixed number of communication rounds as the primary performance metric [18] [19] [20] [21]. However, as discussed in [17], relying solely on final accuracy can be misleading, especially under highly imbalanced data distributions where some clients have significantly more training data than others. To address this limitation, we also report the total training duration as a measure of computational demand, following the approaches in [17] [22] [23]. We also evaluate the communication overhead introduced by each method, an important metric in resource-constrained environments like Internet of Things (IoT), as emphasised by [22] [17] [23].

Because our work targets lightweight models suitable for FL environments, we restrict training to CPU-only resources, avoiding the need for GPU support. This constraint contrasts with works such as [17] and [20], which rely on GPU-accelerated training for larger models.

The selected methods include both server-side and client-side strategies for optimising the FL training process. On the server side, optimisation is particularly important in FL [17] [22], with strategies related to participant selection and workload distribution being the key to improving efficiency. Client-side optimisation can introduce additional computational overhead, but there is a practical limit to how many computations can be offloaded to clients to enhance local training [17].

Most prior works [24] [9] [16] [19] and benchmark frameworks [22] assess FL algorithm performance using datasets like CIFAR-10 or MNIST, which differ substantially from NIDS data in both structure and semantics. Other studies [25] distribute equal-sized datasets with differing labels among clients. Instead, we vary both label distribution and dataset size across clients.

In contrast to prior studies, our work conducts an evaluation of a broader set of FL algorithms within a more realistic experimental environment. Existing research commonly simulates clients by executing multiple processes on a single machine [25] [20]. This practice, however, has been criticised by the community [22], with recommendations favouring the use of Docker containers as a more reliable alternative. With this insight, we simulate each client as an independent Virtual Machine (VM), uniformly provisioned with an equal number of CPU cores and memory resources. This design yields a testing environment that more closely approximates real-world conditions, thereby enabling performance assessments that are both more accurate and generalizable.

While there is a growing body of research dedicated to assessing the privacy implications of FL methods [22, 26], we deliberately exclude this dimension from our study, as our primary objective is to analyse performance and efficiency rather than privacy-preserving guarantees. Similarly, although several studies provide a baseline comparison with centralised training approaches [21], we maintain a clear focus on the comparative evaluation of decentralised learning strategies. Our analysis is therefore entirely centered on benchmarking a diverse set of FL algorithms with respect to their capacity to produce a global model that generalises effectively across heterogeneous client datasets. To this end, we adopt the so-called pathological set-

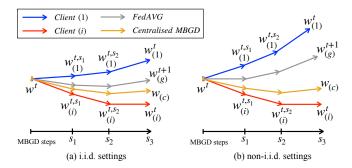


Figure 1: Weight divergence of FedAvg in non-i.i.d. settings.

tings [6], a highly challenging scenario in which each attack type is assigned to a single participant. This setup allows us to stress-test the algorithms under extreme non-i.i.d. conditions, thereby highlighting their relative strengths and limitations in handling realistic adversarial distributions.

3. Limitations of Federated Averaging

Recent research [27, 28] has underscored the limitations of FedAvg when applied to non-i.i.d. and unbalanced datasets distributed across participating clients. In particular, the presence of non-i.i.d. data leads to significant divergence in the local model updates, as illustrated in Figure 1, which contrasts the i.i.d. and non-i.i.d. scenarios. Under non-i.i.d. conditions, the local training performed by clients (e.g., Client 1 and Client i) introduces a pronounced drift in the model parameters, causing the aggregated global model weights $w_{(g)}$ to deviate substantially from the optimal weights $w_{(c)}$ that would have been obtained under centralised training.

With FedAvg, each client independently minimises its local loss function L (formulated in Equation (1)) through several steps of Mini-Batch Gradient Descent (MBGD), starting from the shared global model \mathbf{w}^t distributed at the beginning of each communication round t. MBGD optimises the global model parameters \mathbf{w}^t by computing gradients on small subsets of a client's local data (batches). During training, the client performs several epochs, where one epoch corresponds to a complete pass over its entire local dataset, ensuring that the model parameters are updated multiple times before returning the local model to the server.

Each update moves parameters in the direction that minimises the loss function:

$$L(\mathbf{w}^t, B) = \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} l(y, h_{\mathbf{w}^t}(\mathbf{x}))$$
 (1)

where, l is the individual loss computed with the shared global model \mathbf{w}^t on the labelled sample (\mathbf{x}, y) , which belongs to the batch of local training samples B, while the term $h_{\mathbf{w}^t}(\mathbf{x})$ is the prediction of the model on sample \mathbf{x} . The full list of symbols used in this paper is provided in Table 1.

One step of MBGD is defined as follows:

$$\mathbf{w}_{(i)}^{t,s+1} = \mathbf{w}_{(i)}^{t,s} - \eta_l \nabla L(\mathbf{w}_{(i)}^{t,s}, B)$$
 (2)

Table 1: Mathematical notation used in the paper.

Symbol	Description
\mathbf{w}^t	the global model at round t
$\mathbf{w}_{(i)}^{t,s}$	the local model of client i at MBGD step s of round t
$\mathbf{w}_{(i)}$ $\mathbf{w}_{(i)}^t$	the local model of client i at end of round t
$\Delta \mathbf{w}_{(i)}^t = \mathbf{w}^t - \mathbf{w}_{(i)}^t$	model update of client i at round t
$D_{(i)}$	the local training set of the <i>i</i> -th client
$B \subset D_{(i)}$	batch of samples
$(\mathbf{x}, y) \in D_{(i)}$	training sample \mathbf{x} and its label y
$h_{\mathbf{w}^t}(\mathbf{x})$	prediction of the model \mathbf{w}^t on sample \mathbf{x}
$L(\mathbf{w}_{(i)}^t, B)$	loss function on mini batch B
$l(y, h_{\mathbf{w}_{(i)}^t}(\mathbf{x}))$	loss function on individual sample (\mathbf{x}, y)
μ_i	number of local training updates performed by client i
S^t	subset of clients chosen at round t
N	the total number of clients in the federation
η_l	the local learning rate
η_g	the global learning rate
λ	regularisation weight

where the weights at step s of model $\mathbf{w}_{(i)}^{t,s}$ are updated using the gradients of the loss function L (with $\mathbf{w}_{(i)}^{t,0} = \mathbf{w}^t$ the global model), scaled by the client's local learning rate η_l . As shown in Figure 1, at the end of round t of local training, each client produces a local model $\mathbf{w}_{(i)}^t$ which is sent to the server for aggregation.

While this procedure is effective under the assumption of i.i.d. data, in practical scenarios where clients possess non-i.i.d. datasets, the optimisation trajectories of the local models can diverge substantially, despite being initialised with the same parameters. This divergence arises because each client's updates reflect the biases of its own data distribution, which may not be representative of the overall population. As a result, the aggregation step performed by FedAvg produces a global model that drifts away from the optimal solution that would be obtained if the loss function were minimised centrally on a unified dataset.

Moreover, when datasets are unbalanced across clients, the weighted averaging scheme of FedAvg, where updates are scaled by the number of local samples, amplifies the influence of clients with larger datasets. This weighting can exacerbate the bias introduced by non-i.i.d. distributions, effectively skewing the global model toward the statistical properties of dominant clients while underrepresenting minority distributions. Consequently, FedAvg may suffer from slower convergence, reduced model generalisation, and even convergence to suboptimal local minima. In extreme cases, such as when data heterogeneity is severe, the algorithm may fail to converge altogether, highlighting the challenges posed by non-i.i.d. and unbalanced data in FL.

These limitations represent a fundamental drawback of FedAvg [27]. While prior works have primarily studied its effects in general-purpose machine learning tasks, in this work we specifically investigate its implications in the cybersecurity domain. In particular, we focus on the problem of network intrusion detection, where heterogeneous and unbalanced distributions of benign and malicious network traffic across clients may critically impair the performance and reliability of FL models.

4. Compared algorithms

In this study, we consider seven FL algorithms, including the standard Fedavg algorithm [6] and six advanced methods designed to improve training in non-i.i.d. and unbalanced settings. Three of these algorithms were specifically proposed in the context of network intrusion detection (namely, DAFL, FedSBS and FLAD), while the others were developed for more general purposes (FedProx, SCAFFOLD and FedALA). The selection of these algorithms is motivated by the goal of evaluating how different approaches address the limitations of FedAvg across three key aspects of the FL process: client selection, local training, and model aggregation.

Here we provide a brief overview of each algorithm, highlighting their key features (summarised in Table 2 at the end of this Section).

For the sake of clarity and consistency, we use a uniform notation to describe the methods (Table 1), although it may differ from the original papers.

4.1. FedAvg

FL was introduced in 2017 by McMahan et al. [6] as a communication-efficient method for training neural networks on decentralised data. At its core lies the FedAvg algorithm, which was designed to enable FL to remain effective even in challenging settings with non-i.i.d. and unbalanced datasets. The FL process, common to all the algorithms considered in this study, is iterated over several communication rounds, until the global model converges to a satisfactory solution (as verified by the server on a dedicated test set).

Each round consists of four main phases:

- 1. **Model initialisation**: the server initialises the global model, which is shared with the selected participants.
- 2. **Client selection**: the central server selects a subset of participants to locally train the model.
- 3. **Local training**: each selected participant trains the global model \mathbf{w}^t on its own local data, performing several MBGD steps (Equations 1 and 2). The output of this phase is a local model $\mathbf{w}^t_{(i)}$.
- 4. **Aggregation**: all participant clients send their local updates $\mathbf{w}_{(i)}^t$ to the server for aggregation to obtain a new global model \mathbf{w}^{t+1} .

At each round, the participants are randomly sampled using a uniform distribution, ensuring that each client has the same probability of being selected. The aggregation step involves a weighted averaging of model parameters:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \sum_{i \in S^t} \frac{|D_{(i)}|}{\sum_{i \in S^t} |D_{(i)}|} \Delta \mathbf{w}_{(i)}^t$$
(3)

As shown in Equation (3), the participants with larger dataset size $|D_{(i)}|$ are assigned greater weights.

The steps outlined above are common to all FL algorithms, although each algorithm may implement these steps differently to address data heterogeneity among clients and enhance the convergence of the global model. In this work, we consider the FedAvg algorithm as the baseline method for comparison

with the other algorithms and to highlight the improvements introduced by each of them.

4.2. General purpose algorithms

The three algorithms presented in this section, namely Fed-Prox, SCAFFOLD, and FedALA, were originally developed and evaluated to address the limitations of FedAvg in the context of image classification. Nevertheless, their methodologies can also be adapted to cybersecurity applications, such as collaborative training of a NIDS.

4.2.1. FedProx

FedProx [7] is an extension of the FedAvg algorithm that introduces a regularisation term to the local loss function, allowing for more controlled updates from clients. The regularisation term penalises the local model updates that deviate significantly from the global model. This encourages the clients to stay close to the global model during local training. FedProx was proposed to address the challenges of heterogeneous settings, where clients have different computing capabilities and data distributions.

The local training loss L introduced in Equation (1) is modified to include a regularisation term as follows:

$$L(\mathbf{w}_{(i)}^{t,s}, B) = \frac{1}{|B|} \sum_{(\mathbf{x}, \mathbf{v}) \in B} l(y, h_{\mathbf{w}_{(i)}^{t,s}}(\mathbf{x})) + \frac{\lambda}{2} ||\mathbf{w}_{(i)}^{t,s} - \mathbf{w}^{t}||^{2}$$

In this formulation, i denotes the client index and λ is the regularisation weight, which controls the strength of the regularisation. At the server side, the aggregation mechanism is identical to that of FedAvg (Equation (3)). FedProx does not introduce additional communication costs. However, the incorporation of the regularisation term requires clients to perform extra local computations during training.

4.2.2. SCAFFOLD

SCAFFOLD [8] introduces a correction term to the local updates, which helps to align the local models with the global model. The main idea is to correct the local updates by using a global variate c^t that is shared among all clients:

$$\mathbf{w}_{(i)}^{t,s+1} = \mathbf{w}_{(i)}^{t,s} - \eta_l(\nabla L(\mathbf{w}_{(i)}^{t,s}) + c^t - c_{(i)}^t)$$
(4)

where η_l is the local learning rate and $c_{(i)}^t$ is the local variate computed by client i in the previous round.

Next to the model update, the local variate $c_{(i)}^t$ is updated as the gradient of the local loss function with respect to the shared global model \mathbf{w}^t :

$$c_{(i)}^{t+1} = \nabla L(\mathbf{w}^t, B)$$

or as follows:

$$c_{(i)}^{t+1} = c_{(i)}^t - c^t + \frac{1}{\mu_i \eta_l} (\mathbf{w}^t - \mathbf{w}_{(i)}^t)$$

where $\mathbf{w}_{(i)}^t$ is the local model of client i at the end of local training at round t, η_l is the local learning rate and μ_i is the number

of local updates performed by client i at round t (computed as the number of training epochs multiplied by the number of mini batches B).

On the server side, the global variate is updated by incorporating the drift of the local control variates, defined as $\Delta c_{(i)} = c^{t+1}(i) - c^t(i)$, for the random subset of clients S_t selected for training in round t. The average of the values across these clients is used to update the global control variate:

$$c^{t+1} = c^t + \frac{1}{N} \sum_{i \in S^t} \Delta c_{(i)}$$

At the beginning of round t+1, the global control variate c^{t+1} is communicated to the subset S_{t+1} and used to correct the local training as shown in Equation (4).

The aggregation of the clients' local models is performed as a weighted average, as in FeDAvg, and scaled by the global learning rate η_g :

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta_{g} \sum_{i \in S^{t}} \frac{|D_{i}|}{\sum_{i \in S^{t}} |D^{i}|} \Delta \mathbf{w}_{(i)}^{t}$$

In our validation of the SCAFFOLD algorithm, we use the first option for computing the local variate $c_{(i)}^{t+1} = \nabla L(\mathbf{w}^t, B)$, as it provides stable training with minimal computational overhead. In terms of communication overhead, SCAFFOLD requires the transmission of the global variate c^t to all clients at each round, plus the transmission of the local variates $c_{(i)}^t$ from each client to the server.

4.2.3. FedALA

FedALA (Federated Learning for Adaptive Local Aggregation) [9] combines the parameters of the global model with those of each client's local model, assigning different weights to each parameter according to its relevance for the local optimisation objective.

The central idea of FedALA is therefore to preserve part of the information contained in the client's local model from the previous round (t-1), rather than completely overwriting it with the global model received from the server in round t. Specifically, because the lower layers of a neural network capture more generic representations [29], while the higher layers learn task-specific features, the ALA aggregation is applied only to p higher layers to preserve local information, whereas the generic knowledge from the global model is retained in the lower layers. This is achieved by combining the current global model \mathbf{w}^{t-1} with the previous local model \mathbf{w}^{t-1} as formulated as follows:

$$\hat{\mathbf{w}}^{t} = \mathbf{w}_{(i)}^{t-1} + (\mathbf{w}^{t} - \mathbf{w}_{(i)}^{t-1}) \odot [1^{|\mathbf{w}_{(i)}^{t-1}|-p}; \theta_{(i)}^{p}]$$
 (5)

where \odot is the Hadamard product, $\theta_{(i)}^p$ denotes the aggregation weights for the higher layers, whereas $1^{|\mathbf{w}^{i-1}(i)|-p}$ represents the identity matrix applied to the remaining $|\mathbf{w}^{t-1}(i)| - p$ lower layers, ensuring they are overwritten with the corresponding weights of the global model.

The resulting model $\hat{\mathbf{w}}^t$ is trained with MBGD with the weights $\theta^p_{(i)}$ frozen:

$$\mathbf{w}_{(i)}^{t,s+1} = \mathbf{\hat{w}}_{(i)}^{t,s} - \eta_{l_1} \nabla_{\mathbf{\hat{w}}_{(i)}^{t,s}} L(\mathbf{\hat{w}}_{(i)}^{t,s}, B)$$

Similarly, at each round the aggregation weights $\theta_{(i)}^p$ are optimised using MBGD, while the parameters of both the global and local models remain frozen.

$$\theta_{(i)}^p = \theta_{(i)}^p - \eta_{l_2} \nabla_{\theta_{(i)}^p} L(\hat{\mathbf{w}}_{(i)}^{t,s}, B)$$

where η_{l_1} and η_{l_2} are the local learning rates for the model weights and the aggregation weights, respectively.

The aggregation weights are trained until convergence during the first two rounds, and in subsequent rounds they are updated for a single epoch to reduce computational overhead.

Although FedALA does not incur additional communication costs, it introduces extra computational overhead for the clients due to the training of the ALA parameters and the aggregation of the global and local models.

4.3. Algorithms for network intrusion detection

DAFL, FedSBS, and FLAD were specifically designed to address FedAvg's convergence issues within the network security domain. Each of these algorithms introduces modifications at every stage of the FL process, namely: local training, client selection, and model aggregation.

4.3.1. DAFL

DAFL (Disparity-Aware Federated Learning) [11] is an FL algorithm designed for collaborative training of NIDSs. Its key idea is to aggregate updates only from clients that achieve sufficiently high accuracy on their local data. The filtering is performed client-side: if a client's local model accuracy is below a predefined threshold hyperparameter β , its local model is not sent to the server. By excluding poorly performing local models, DAFL aims to accelerate convergence and improve the quality of the global model. In addition to the filtering procedure, DAFL also modifies the aggregation strategy by giving more weight to the better-performing local models as follows:

$$\mathbf{w}^{t+1} = \sum_{i \in S^t} \frac{\rho_{(i)} a_{(i)}}{\sum_{j \in S^t} \rho_{(j)} a_{(j)}} \mathbf{w}_{(i)}^t \quad \text{where:}$$

$$\rho_{(i)} = \frac{|D_{(i)}|}{\sum_{i \in S^t} |D_{(i)}|} \quad a_{(i)} = \frac{e^{A_{(i)}}}{\sum_{i \in S^t} e^{A_{(j)}}}$$

where S^t is the subset of clients that sent back their local model at round t, $\rho_{(i)}$ and $a_{(i)}$ are the weights assigned to each client based on the size of their local dataset $|D_{(i)}|$ and their local accuracy $A_{(i)}$, respectively.

The DAFL algorithm is designed to improve the convergence speed of the global model by focusing on the clients that contribute positively to the training process. However, it may lead to a situation where some clients are never selected for aggregation, which can result in missing out some attack patterns present in their local datasets.

It should be noted that DAFL may increase communication overhead, as the server must transmit the global model to all clients at every round. Furthermore, DAFL requires active clients, i.e., those whose validation accuracy exceeds the threshold β , to send their accuracy scores and the number of training samples to the server in addition to their local model parameters. Finally, since all clients must perform local training at each round before computing their validation accuracy, DAFL also introduces additional computational overhead.

4.3.2. FedSBS

The key aspect of FedSBS [10] is a client selection strategy based on a greedy algorithm that selects clients by estimating their expected improvement in the global model performance with the Information Gain (IG) score. The potential improvement is computed as the difference between the global and local losses, taking into account the imbalance φ_i of dataset classes for each client:

$$I_i \leftarrow -\ln L(\mathbf{w}^t) + \varphi_i \ln L(\mathbf{w}_{(i)}^t)$$

The local dataset imbalance coefficient φ_i is computed by using the Shannon entropy of the class distribution in the local dataset:

$$\varphi_i = \begin{cases} 1 + \sum p_c \log_2(p_c), & \text{if } \ln L(w_i^t) < 0 \\ -\sum p_c \log_2(p_c), & \text{if } \ln L(w_i^t) \ge 0 \end{cases}$$

where p_c is the proportion of class c in client's i dataset. The higher the value of I_i , the more likely the client i is to be selected for the next round.

To promote the participation of all clients, the method implements a so-called ϵ -greedy policy, which selects a participant for the next round using the IG score with a probability of $1-\epsilon$, or in a random fashion with probability ϵ , with $0<\epsilon<1$. In addition, FedSBS rejects the clients that are chosen too frequently. This is achieved by computing the Boltzmann distribution to determine the likelihood of a participant being selected for the next FL round. Specifically, once a participant has been selected, the probability of its exclusion from subsequent selections increases with each additional selection. This approach reduces the over-participation of certain clients, thereby producing a less biased final model.

Clients of FedSBS apply a regularisation term $r(\mathbf{w}_{(i)}^{t,s})$ to the local loss function at each MBGD step s:

$$L(\mathbf{w}_{(i)}^{t,s}, B) = \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} l(y, h_{\mathbf{w}_{(i)}^{t,s}}(\mathbf{x})) + \lambda r(\mathbf{w}_{(i)}^{t,s})$$

where the regularisation term $r(\mathbf{w}_{(i)}^{t,s}) = \mathbf{w}_{(i)}^{t,s} + \mathbf{w}^t - \mathbf{w}^{t-1}$ incorporates the gradient information of the global model $\mathbf{w}^t - \mathbf{w}^{t-1}$ between two consecutive rounds, and λ controls the strength of the regularisation.

The gradient information is also used by the server to implement a momentum-based method for the aggregation of clients' updates:

$$\mathbf{w}^{t+1} = \sum_{i \in S^t} \frac{|D_{(i)}|}{\sum_{i \in S^t} |D_{(i)}|} (\mathbf{w}^t - \eta_g(\mathbf{w}_{(i)}^t - r(\mathbf{w}_{(i)}^t)))$$

Table 2: Summary of the main features of the algorithms evaluated in this work. Differences with FedAvg	are highlighted in bold.
---	--------------------------

Method	Local Training	Client Selection	Aggregation	Communication
FEDAVG	MBGD	Random subset	Weighted Averaging	Global model and client updates
FedProx	MBGD with regularisation	Random subset	Weighted Averaging	Global model and client updates
SCAFFOLD	MBGD with control variates	Random subset	Weighted average with global learning rate	Global model, client updates and Control variates
FedALA	MBGD on aggregated local and global models; MBGD on aggregation weights	Random subset	Weighted averaging	Global model and client updates
DAFL	MBGD; Model validation	Accuracy threshold on client side	Customised weighted averag- ing based on number of training samples and local accuracy	Global model, client updates and local accuracy
FedSBS	MBGD with regularisation	Information Gain client scoring and random	Weighted averaging with momentum	Global model, client updates and loss values
FLAD	MBGD with configurable epochs and steps; Model validation	Accuracy-based	Arithmetic mean	Global model, client updates, local accuracy and number of epochs and steps

The momentum term serves to preserve knowledge accumulated from previous aggregation rounds, ensuring that valuable information from earlier updates is not lost. By doing so, it enhances the stability of the model during the FL process.

FedSBS implements a mixed greedy/random client selection strategy. Similar to FedAvg, the number of clients selected per round can be set through a specific hyperparameter (denoted as *s* in the FedSBS paper). Consequently, the only additional network overhead compared to FedAvg arises from transmitting the losses of the global and local models on the clients' validation sets, which the server requires to compute the IG. Moreover, the computation of these losses, along with the regularisation term, introduces extra load on the client side.

4.3.3. FLAD

FLAD (adaptive Federated Learning Approach to DDoS attack detection) [12] implements a custom client selection mechanism, with the aim to overcome the limitations of FedAvg on unbalanced and out-of-distribution data across the clients. The method, which was evaluated on an unbalanced and non-i.i.d. dataset of DDoS attacks, is designed to improve the performance of the global model by focusing on clients that are underperforming in terms of accuracy. This is achieved by adapting the number of local training steps performed by each client based on their local accuracy scores. Such accuracy scores are computed by the clients at each round by evaluating the global model on their local validation set and are communicated to the server. With this information, the server determines each client's local training steps for the next round.

This approach aims to assign more training to the clients with lower accuracy, so they can get closer to the optimal performance, while clients with high accuracy receive less training steps, possibly zero, because their performance is already close to the maximum.

The communication of the accuracy scores is essential for the FLAD algorithm, as it allows the server to determine which clients need more training steps, which ones can be skipped and when to stop the training process (early-stopping with *patience*). However, along with the communication of the number of training steps from the server to the clients, it produces additional communication overhead compared to the standard FedAvg algorithm.

The computation of accuracy scores on the local validation set introduces additional computational overhead for the clients. Unlike FedAvg, where the local training load is constant across clients, the load in this case is dynamically adjusted at each round based on the client's local accuracy. As a result, the computational load may vary significantly between clients, depending on their individual performance.

5. Experimental setting

The experimental environment is deployed on a server-class machine equipped with two 16-core Intel Xeon Silver 4110 @ 2.1 GHz CPUs and 64 GB of RAM. The system is configured as a Kubernetes cluster, where Virtual Machines (VMs) serve as nodes. Each node corresponds to a VM provisioned with 3.8 GB of RAM and 2 CPU cores. This setup provides strong isolation within the federation and ensures proper separation of client resources, thereby enabling reliable evaluation of algorithm performance.

Communication between clients and the server is handled via a Mosquitto message broker [30], deployed as a Kubernetes pod on the server node. Mosquitto is a lightweight, open-source implementation of the Message Queuing Telemetry Transport (MQTT) protocol. Messages exchanged through this publish-subscribe system include model weights and method-specific parameters. The source code and the configuration files of the experimental environment are publicly available for reproducibility and further research [15].

5.1. Algorithms and hyperparameters

The FL algorithms and the global ML model are implemented in Python, using the Tensorflow framework v2.12 for the model and the FL algorithms.

5.1.1. FL algorithms

For each FL method, the hyperparameters are set to align with the original papers and are summarised in Table 3. Please note that 5 methods use the same client selection ratio (0.5), meaning that 6 out of 13 clients are selected at each round. The only exceptions are FLAD and DAFL, which select clients based on their local accuracy scores. With FLAD the clients are selected by the server before a FL round starts, while in DAFL, the selection is performed by the clients themselves based on their local accuracy scores.

Table 3: Hyperparameters used for each algorithm.

Algorithm	Hyperparameters
	Selection ratio: 0.5
FedAvg	Local epochs: 1
	Local steps/epoch: training_samples
	Regularization weight λ : 1
FedProx	Selection ratio: 0.5
reuriox	Local epochs: 1
	Local steps/epochs: training_samples 1024
	Global learning rate η_g : 1
SCAFFOLD	Selection ratio: 0.5
SCHIOLD	Local epochs: 1
	Local steps/epoch: training_samples 1024
	Higher layers <i>p</i> : 1
	Selection ratio: 0.5
FedALA	Local epochs: 1
	Local steps/epoch: training_samples 1024
	Threshold accuracy β : 0.6
DAFL	Selection ratio: accuracy-based
	Local epochs: 1
	Local steps/epoch: $\frac{ \text{training_samples} }{1024}$
	Regularization weight <i>λ</i> : 1
	ε min: 0.1
	Initial ε : 1.0
E IGEG	Clients/round s: 6
FedSBS	Local epochs: 1
	Local steps/epocn: 1024
	Patience value: 25 rounds
	Selection ratio: accuracy-based
FLAD	Min local epochs: 1
	Max local epochs: 5
	Min local steps/epoch: 1
	Max local steps/epoch: 1000

As summarised in Table 3, the number of local epochs is set to 1, meaning that each client performs a single pass over its local dataset before sending the updated model weights to the server. The number of local MBGD steps/epoch is computed as the size of the local dataset divided by the batch size (1024 training samples), or set to 1 for training sets smaller than the batch size (e.g., the client with the *WebDDoS* attack). Note that FLAD is the only method that assigns both local steps and epochs dynamically, meaning the number of epochs and batch sizes vary across different clients and rounds.

To allow a fair comparison, all algorithms execute the same number of rounds. The actual number of rounds depends on the runtime of the FLAD method, as it is the only method that implements a stopping criterion based on the average clients' validation accuracy. FLAD executes first, and its total number of rounds is used for all other algorithms.

5.1.2. Global model

The global model is an Multi-Layer Perceptron (MLP) consisting of two hidden layers with 32 neurons each, using the ReLU activation function. ReLU is a widely adopted activation function in neural networks due to its low computational cost and its ability to mitigate the vanishing gradient problem [31].

The input to the model is an encoded representation of a network flow that aggregates packet-level features [32]. Thus, a single sample is a two dimensional array of shape (10, 11), where 10 is the number packets extracted from the traffic traces for each flow and 11 is the number of packet-level features. These features include packet length, IP and TCP flags, and other relevant statistics that characterize a traffic flow. The neural network is trained to classify the network traffic either as benign or DDoS. Thereby, the output layer uses a sigmoid activation function to produce a probability score for each traffic flow being DDoS. The loss function is the binary cross-entropy, a common choice for binary classification tasks. Model parameters are optimised using MBGD with learning rate set to 0.1.

Model weights are randomly initialised by the server using a different seed for each test run and are communicated to all clients before starting the FL. To ensure a fair comparison and reproducible results, the initialised MLP model is kept the same across all FL methods.

5.2. The Dataset

The evaluation is performed with a recent dataset of DDoS attacks, CIC-DDoS2019 [33], provided by the Canadian Institute of Cybersecurity of the University of New Brunswick. CIC-DDoS2019 consists of several days of network activity, and includes both benign traffic and 13 different types of DDoS attacks.

The benign traffic in the dataset was generated using the B-profile [34], which models the distribution of typical applications such as web browsing (HTTP/S), remote shell access (SSH), file transfers (FTP), and email communications (SMTP). The malicious traffic was produced using third-party tools and consists of 13 types of DDoS attacks:

- 1. **WebDDoS**: A web-based DDoS attack that targets web servers, aiming to exhaust resources and disrupt service availability.
- 2. **LDAP**: Uses the Lightweight Directory Access Protocol (LDAP) for reflection, increasing traffic volume directed at the victim.
- 3. **PortMap**: A reflection-based DDoS attack leveraging the Port Mapper service to amplify traffic and overwhelm the target.
- 4. **DNS**: Uses the Domain Name System (DNS) for reflection, amplifying traffic directed at the victim.
- UDPLag: A variation of the UDP attack, specifically designed to disrupt online gaming by introducing lag through bandwidth consumption.

- 6. **NTP**: Exploits the Network Time Protocol (NTP) for reflection, amplifying traffic to overwhelm the target system.
- 7. **SNMP**: Exploits the Simple Network Management Protocol (SNMP) for reflection, increasing traffic volume aimed at the target.
- 8. **SSDP**: Uses the Simple Service Discovery Protocol (SSDP) for reflection, amplifying traffic to flood the victim.
- 9. **Syn Flood**: A SYN flood attack that sends a series of SYN requests to a target's system, consuming resources and potentially causing a denial of service.
- 10. **TFTP**: Exploits the Trivial File Transfer Protocol (TFTP) for reflection, amplifying traffic to overwhelm the target system.
- 11. **UDP**: A generic DDoS attack that floods the target with User Datagram Protocol packets, consuming bandwidth and resources.
- 12. **NetBIOS**: Exploits the NetBIOS service for reflection, amplifying traffic to flood the target system.
- 13. **MSSQL**: Targets Microsoft SQL Server by exploiting its UDP port for reflection, amplifying traffic to the victim.

As described in previous works [12], these attacks present disjoint feature distributions and different traffic volumes, enabling the configuration of non-i.i.d. and unbalanced FL scenarios.

5.3. Clients

The evaluation environment consists of 13 clients and a single central server that communicates with all clients (Figure 2). As in previous research [12], the dataset is distributed among the clients in a non-i.i.d. and unbalanced manner: each attack type is assigned to a different client, and the amount of data varies across clients as presented in the Figure. This scenario simulates a realistic FL environment where clients possess heterogeneous data both in terms of distribution and quantity.

Each client's dataset is balanced to contain approximately the same amount of benign and attack samples. Moreover, each dataset is split into training (90%) and test (10%) sets, with 10% of the training set kept as validation data. It is worth noting that the test set is never used during the FL process and is only employed to evaluate the final global model performance.

6. Evaluation results

As anticipated in the previous sections, the experiments are designed to evaluate the performance of FL algorithms in settings with non-i.i.d. and unbalanced data distributions. We compare the performance of the algorithms in terms of network overhead, client selection, training time, model aggregation, and accuracy and analyse the impact of their customisations with respect to the standard FedAvg algorithm. For each metric, we perform 10 runs with different random seeds to improve the reliability of the results.

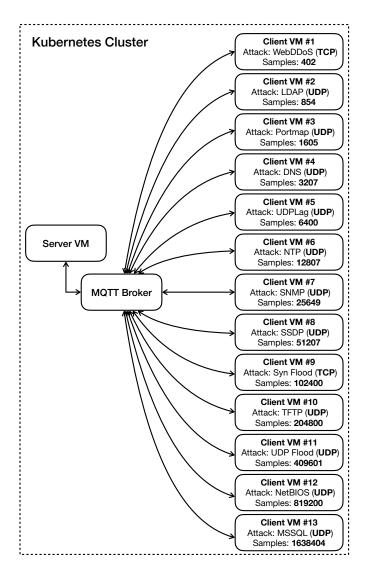


Figure 2: Overview of the experimental environment. Each client is assigned a dataset containing benign traffic and a single DDoS attack type. The number of samples refers to the overall dataset size, encompassing training, validation, and test sets.

6.1. Network overhead

Network overhead is a crucial aspect of FL algorithms, especially in environments with limited, unstable or expensive connections. In this section, we analyse the network traffic generated by each approach, focusing on the amount of data exchanged between clients and the server during training, including global and local models and any additional information required for managing the FL process. In this way, we also highlight the different approaches to client selection and model aggregation. Our experiments are conducted with the MLP model described in Section 5.1.2, which consists of two hidden layers with 32 neurons each (37.5 KBytes).

Table 4 presents the experimental results about network traffic consumption, with FedAvg as the baseline. The network consumption data for FedProx and FedALA is similar to that of FedAvg, as they use the same client selection strategy and do not require additional data exchange. On the other hand,

Table 4: Average network bandwidth (MB) consumed by each client across all 10 experime	Table 4: Average network bandwidth	(B) consumed by each client	across all 10 experiments.
--	------------------------------------	---	----------------------------

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	Sum
FEDAVG	2.78	2.94	2.80	2.55	2.55	2.50	2.64	2.65	2.65	2.77	2.40	2.69	2.66	34.59
FedProx	2.78	2.94	2.81	2.55	2.55	2.50	2.64	2.65	2.65	2.77	2.40	2.69	2.66	34.59
SCAFFOLD	4.84	5.12	4.89	4.45	4.45	4.36	4.60	4.62	4.62	4.82	4.18	4.68	4.63	60.25
FedALA	2.78	2.94	2.80	2.55	2.55	2.50	2.64	2.65	2.65	2.77	2.40	2.69	2.66	34.59
DAFL	3.03	5.73	5.75	5.59	5.76	5.76	5.76	5.76	5.76	5.76	5.76	5.76	5.76	71.98
FedSBS	5.38	5.61	5.55	5.59	5.58	5.54	5.63	5.56	5.66	5.59	5.36	5.67	5.34	72.06
FLAD	4.79	3.73	4.05	3.42	3.39	3.90	4.06	3.93	4.44	4.14	3.91	4.20	4.25	52.18

DAFL, FedSBS, FLAD, and SCAFFOLD introduce additional overhead due to their specific design choices:

- DAFL always requires all clients to train, although only those that meet a predefined accuracy threshold send their local model to the server.
- The clients of **FedSBS** are required to send the validation loss of both global and local models to the server, which use these values for computing the Information Gain.
- FLAD evaluates the performance of the global model on local datasets and require the clients to send their local accuracy values to the server. For this reason, the server sends the global model to all clients in every round for evaluation.
- SCAFFOLD requires exchanging arrays of control variates, which have the same number of parameters as the model itself.

As already mentioned in Section 4, FLAD and FedSBS implement an evaluation phase, which is necessary to compute the performance of the global model on each client's local dataset. This evaluation is crucial for both methods, as it allows them to adaptively select clients based on their local performance. However, there is a key difference between the two methods: while FedSBS always selects a constant number of clients per round (partly at random and partly based on the IG score), FLAD adaptively adjust the number of clients selected in each round as training progresses (see Figure 3), focusing on those clients that require more training to improve the global model's performance. This dynamic selection strategy helps FLAD to reduce network overhead caused by local updates transmitted by the clients, as it progressively excludes clients that have already achieved good performance. This can be observed in Table 4, where FLAD shows higher network consumption for out-of-distribution (o.o.d.) clients such as WebDDoS and Syn, which require more training and, consequently, more data exchange. In contrast, clients like DNS, UDPLag, and LDAP show lower network traffic, indicating that they are learned efficiently and require less communication.

The network overhead of both FedSBS and DAFL is significantly higher than that of FedAvg. Although their mechanisms differ, the amount of data exchanged in each round is very similar. With FedSBS, 25 models are exchanged between server and clients: the server sends 6 copies of the global model to the selected clients, which in turn send one local model each back to the server. In addition, the server distributes a copy of the global model to all 13 clients for the evaluation of global and

local losses on their validation sets. With DAFL, the server always sends a copy of the global model to all 13 clients. Only those clients whose local accuracy exceeds a predefined threshold β return their model to the server. In theory, the number of responding clients could range from 0 to 13. In practice, with our setup, after a few initial rounds this number stabilises at 12, as only the WebDDoS clients consistently fail to meet the accuracy requirement. This again results in 25 model exchanges per round, just like in FedSBS. In addition, both approaches require clients to send extra information to the server: FedSBS transmits the losses, while DAFL transmits the local accuracies used for aggregation. This further explains why the network overhead of the two methods is so similar.

The network overhead of SCAFFOLD is influenced by the exchange of control variates between clients and the server. In our implementation, the control variates are computed as the gradient of the global model on the local dataset and stored in a NumPy array with the same number of elements as the model's weights. Since a TensorFlow model also includes additional metadata (e.g., model's architecture, optimizer state, training configuration, etc.), its size is slightly larger than that of the control variate array. At each round, the server sends the global model and the global control variates to six selected clients and receives six local models and six control variate arrays in return. In total, therefore, 24 data structures are exchanged between the server and clients at each round, close to the 25 exchanged by FedSBS and DAFL. However, 12 of them are control variates, whose size is smaller than a model. Moreover, FedSBS and DAFL also require the transmission of additional information such as losses and accuracies. Overall, this explains the observed differences in network overhead among the three approaches.

As a final consideration, the Deep Learning (DL) models used in these experiments are relatively small (37.5 KBytes) and introduce negligible network overhead in most scenarios. In practice, however, models can be much larger, for example, Large Language Models (LLMs) such as Bidirectional Encoder Representations from Transformers (BERT), which are also applied in cybersecurity [35]. These models contain hundreds of millions of trainable parameters and can exceed 1 GByte in size. In such cases, FedProx and FedALA would be the preferred approaches for training a NIDS in heterogeneous settings.

6.2. Client selection

A crucial aspect of FL in heterogeneous environments is the client selection strategy. The choice of which clients partici-

Table 5: Percentage of rounds in which a particular client is selected for local training (with respect to the total number of rounds). Approaches using FedAvg's random selection (FedProx, SCAFFOLD and FedALA) choose 6 out of 13 clients in each round, resulting in an average participation rate of about 46%.

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	Average
FedAvg-based	48	51	48	45	45	43	45	45	46	48	41	47	46	46
DAFL	100	100	100	100	100	100	100	100	100	100	100	100	100	100
FedSBS	44	47	46	47	46	46	48	47	49	47	43	48	43	46.2
FLAD	68	30	39	19	17	34	40	36	54	44	36	46	49	39.4

pate in each training round can significantly impact the convergence speed, final model accuracy and the computational overhead on clients. Different FL methods employ various strategies for client selection, ranging from random selection to more sophisticated approaches based on client performance or data characteristics. To evaluate the effectiveness of each selection strategy, we report the distribution of training rounds across all clients in Table 5. Most of the analysed methods incorporate the FedAvg's random selection, which assigns a similar number of rounds to each client. The analysis is more interesting for the DAFL, FedSBS, and FLAD algorithms.

DAFL performs client selection after the training procedure, forcing all clients to execute local training in every round. This results in a long training time because the slowest client always performs the local training. FedSBS selects clients partly at random and partly through a scoring mechanism based on the Information Gain metric. Similar to FedAvg, it also enforces a maximum client participation per round (50%). While its design explicitly aims to prioritise clients that maximise Information Gain, in practice the strategy proves less effective: under the default configuration, the resulting client selection distribution closely resembles that of FedAvg, showing near-uniform randomness.

FLAD, on the other hand, does not impose a fixed number of clients per round, but instead selects clients based on their local accuracy scores. As shown in Table 5, clients such as WebDDoS and Syn, with o.o.d. training data, are selected more frequently than others. A higher number of training rounds indicates that more computation is needed to build an effective model for that client.

Figure 3 shows the number of selected participants per round during the longest experiment run. In the figure, we can see that DAFL selects all clients in each round, while FLAD selects a varying number of participants over time, with an average of approximately 5.13 participants per round. All the other methods select 6 out of 13 clients, which is the result of a selection ratio of 0.5.

Although FLAD selects fewer clients per round on average, this does not implies that the duration of the whole FL is shorter. As discussed in Sections 6.3 and 6.4, FLAD's training time can be longer than that of other methods, as it dynamically assigns training load to each client. This can lead to very long local training sessions for some clients, hence increasing the overall FL duration.

6.3. Federated training time

The duration of FL is critical, as timely model updates are essential in cybersecurity to maintain system performance. In

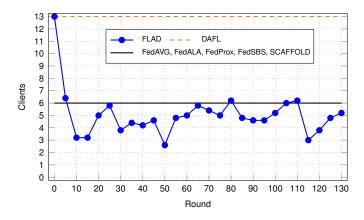


Figure 3: Number of participants/round selected by FLAD during the longest experiment run (133 rounds). The two horizontal lines represent the static number of participants/round selected by DAFL (13 clients) and the other methods (6 out of 13 clients).

this section, we analyse the duration of each method in terms of the total time taken to train the model and the time spent by each client during the training process.

In this experiment, all FL methods are trained for the same number of rounds. As discussed in Section 5.1, the number of rounds is determined by FLAD, which is the only method that applies an early stopping mechanism. FLAD is executed first, and the other methods are then run for the same number of rounds to ensure fairness.

The duration of the FL process is shown in Figure 4, which presents the distribution of execution durations for each

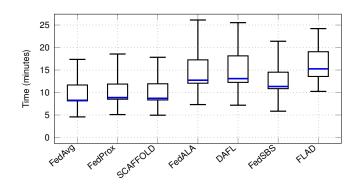


Figure 4: The figure depicts the distribution of execution durations for each method, reporting the minimum and maximum observed values. The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the median. The total duration of each round is determined by the slowest client, communication time and aggregation time.

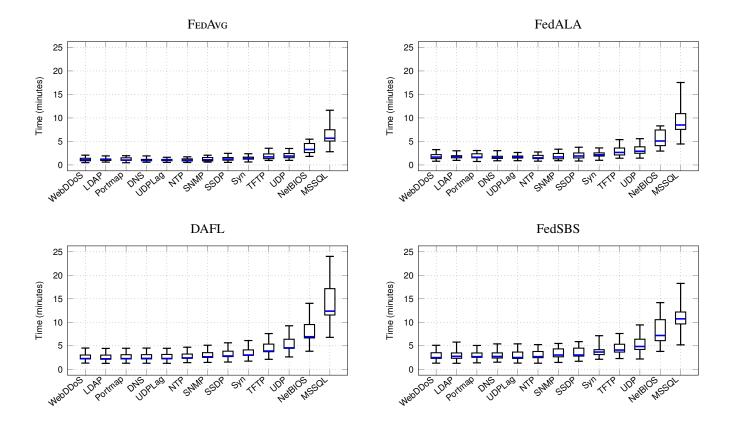


Figure 5: The figure depicts the distribution of local training times for each client with FeDAvg (top left), FedALA (top right), DAFL (bottom left) and FedSBS (bottom right). The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the median.

method, reporting the minimum and maximum observed values, the interquartile range and the median. The duration includes the clients' operations (e.g., model training and evaluation), data trasmission (model weights, control variates, accuracy scores, etc.) and global model aggregation.

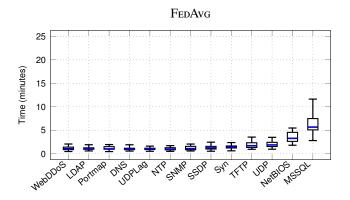
The figure shows that FLAD has a higher duration compared to other methods, although the clients are selected for a lower number of rounds on average, as discussed in Section 6.2. The reason is that FLAD dynamically assigns training epochs and steps to each client based on their performance, which can lead to very long local training sessions for some clients, significantly increasing the overall FL duration. Conversely, the other methods always assign one training epoch and a fixed number of training steps/epoch to each client (see Section 5.1).

Other relevant observations from Figure 4 are:

- The duration of the FL process with SCAFFOLD and FedProx aligns with that of FedAvg, as they require minimal extra coomputation to train the clients. FedProx introduces a small overhead due to the additional regularisation term, while SCAFFOLD requires the computation of control variates, which does not significantly affect the training time.
- **DAFL** requires all clients to train in each round, which leads to a longer duration compared to FedAvg. This is because the slowest client (that with the largest dataset *MSSQL*) determines the overall duration of each round, and with DAFL this client is always selected.

- The longer duration of FedALA is primarily due to the extra training phase required to estimate the aggregation weights used to combine the global model with the client's local model.
- The execution time of FedSBS exceeds that of FedAvG due to the additional evaluation phase, during which the global model is systematically assessed on each client's local dataset. An analogous evaluation step is also implemented by FLAD.

The overall duration of the FL process is determined by the time taken by the slowest client selected in each round to execute local operations and the time required to transmit the model weights and other data to the server. In our experimental setup, all the clients are running on machines with the same hardware specification and the same connection speed, so the impact of the slowest clients on the overall duration mostly depends on the size of its local training set and the computational complexity of the local functions, including the loss function, model aggregation and evaluation. Another factor that can influence the duration is the amount of data exchanged between the clients and the server, which influences the transmission time. FLAD, DAFL and FedSBS always send the global model to all clients (either for training or evaluation), while FEDAVG, FedProx and SCAFFOLD only send the model to the selected clients. In addition, FedSBS sends local evaluation results to the server, like FLAD, which also distributes the training epochs and steps assigned to each client. These additional



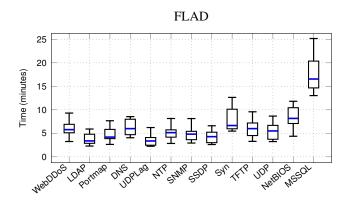


Figure 6: The figure depicts the distribution of local training times for each client with FEDAvg (left) and FLAD (right). The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the median.

interactions contribute to the overall duration of the FL process.

6.4. Local execution time

As mentioned above, the local execution time of each client is a key factor that influences the overall duration of the FL process. This time determines how quickly each round can be completed, as the server must wait for all selected clients to finish their local operations before proceeding to the next round. In addition, this value also indicates, along with the number of rounds a client is selected for training, the total computational load imposed on each client.

Figure 5 shows the distribution of local training times of each client for FedAvg, FedALA, DAFL, and FedSBS. FedProx and SCAFFOLD are not included in this analysis, as they show similar results to FedAvg (see Figure 4). The figure demonstrates the dependency of the local training time on the size of the local dataset, which is consistent across all methods. The clients with larger datasets, such as MSSQL and NetBIOS, require more time to train their local models compared to others. However, while with FedAvg, FedALA and FedSBS such clients are not always selected, with DAFL they are selected in every round, leading to a longer overall duration of the FL process. In addition, the figure shows the impact of the additional computations required by some methods. For example, FedALA requires an extra training phase to estimate the aggregation weights, while FedSBS requires an evaluation phase, which also adds to the local training time.

A different behaviour is observed with FLAD, as shown in Figure 6. The local training time of each client is influenced by the number of training epochs and steps assigned to it, which are dynamically determined based on the client's performance. This leads to a more varied distribution of local training times, with some clients requiring significantly more time than others. For example, clients with o.o.d. attack profiles *Syn* and *Web-DDoS* are selected frequently (see also Table 5) and assigned more training epochs and steps, since they are often misclassified by the global model, leading to longer local training times.

6.5. Aggregation analysis

In this section, we analyse the performance of the aggregation mechanisms implemented by the different methods. We focus on the validation accuracy of the global model and of the local models before aggregation, as well as on the impact of aggregation on these scores. This analysis allows us to understand how the different aggregation and client selection mechanisms (see Table 2) affect the accuracy of the global model in correctly classifying the attacks, particularly the o.o.d. ones.

The plots in Figure 7 compare the performance of the local models with that of the aggregated global model. The reported scores correspond to the training round in which the global model achieves its highest average performance. To ensure a fair assessment of the aggregation mechanisms, we evaluate the local models that contributed to the derivation of that global model. Moreover, we focus on a subset of representative clients, chosen to better highlight the impact of the aggregation mechanisms on the global model's performance. Performance is evaluated using the F1 score, the harmonic mean of precision (Pre) and recall (Rec), which provides a balanced measure of model accuracy, particularly in scenarios with imbalanced datasets. These metrics are formally defined as follows:

$$F1 = 2 \cdot \frac{Pre \cdot Rec}{Pre + Rec} \quad Pre = \frac{TP}{TP + FP} \quad Rec = \frac{TP}{TP + FN}$$

where *TP=True Positives*, *FP=False Positives* and *FN=False Negatives*.

As expected, local models generally perform better on their respective datasets. However, aggregation can significantly impact this performance: sometimes improving, but often degrading it. o.o.d. attacks, such as *WebDDoS* and *Syn*, typically show a notable drop in accuracy after aggregation, while the UDP-based ones, like *NetBIOS* and *MSSQL*, are less affected by this process.

FEDAVG, SCAFFOLD, and FedSBS show comparable performance on the WebDDoS and Syn datasets. While they achieve satisfactory accuracy after aggregation on the *Syn* attack, they completely fail on the *WebDDoS* attack. These findings are consistent with previous research [12], which highlighted the limitations of relying on random client selection and weighted

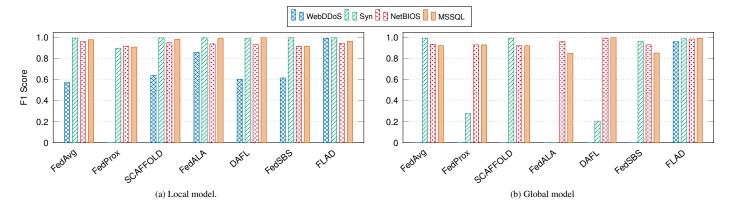


Figure 7: Accuracy scores on the validation datasets of the local models (left) and the global model (right) after aggregation. Only the clients that show significant changes in accuracy after aggregation are included.

averaging as aggregation strategies in such scenarios (though FedSBS does not rely on purely random client selection).

Two other methods, FedProx and FedALA, struggle to maintain high classification accuracy after aggregation on both the WebDDoS and Syn attacks. Compared to FeDAvg, both approaches focus on enhancing the local training algorithm while preserving the same aggregation and client selection strategies. FedProx introduces a regularisation term that penalises the divergence of local models from the global one. While this helps stabilise training, it also reduces the local models' ability to adapt to the specific characteristics of their datasets. By contrast, FedALA aggregates each client's local model with the global model received from the server, aiming to adapt the global model to the client's local objective. Although this approach improves performance on local datasets (see Figure 7a), once the server aggregates these adapted models, the resulting global model loses its ability to generalise to o.o.d. datasets such as WebDDoS and Syn (see Figure 7b).

Clearly, FLAD outperforms all other methods in producing a global model capable of effectively classifying o.o.d. attacks. This is achieved by leveraging the feedback from clients regarding the performance of the global model on their local datasets, enabling the server to dynamically adjust the training process and concentrate on the most challenging attacks. This is evident from Figure 8a, which illustrates the average validation F1 score of the global model over increasing rounds. The plot shows that FLAD consistently improves the global model's performance throughout the training process, ultimately achieving the highest F1 score among all methods. In the figure, we can observe that after ten rounds, FLAD outperforms all other methods and maintains this lead until the end of the training process. The initial performance boost is achieved by engaging all clients at the beginning of the FL process while, in later rounds, the method focuses on the most challenging clients, such as those with o.o.d. attacks.

DAFL could, in principle, achieve comparable results, since it also requires clients to report their local accuracy scores to the server. However, in DAFL this information is primarily used to exclude low-performing clients from the aggregation process, which prevents the global model from incorporating critical information needed to accurately classify o.o.d. attacks.

The limitations of DAFL and the other methods in handling o.o.d. attacks are evident in Figures 8b and 8c, which report the F1 score of the aggregated global model on the WebDDoS and Syn datasets across training rounds. The plots indicate that, with the exception of FLAD, all methods struggle to generate a global model capable of effectively classifying these attacks. The WebDDoS attack, in particular, proves especially challenging, likely due to the relatively small size of its dataset compared to the others. This limits aggregation mechanisms that weight contributions by dataset size, causing clients with smaller datasets to be underrepresented in the global model. The issue of dataset size does not arise in the Syn attack, which benefits from a larger dataset. Nevertheless, as shown in Figure 8c and discussed above, DAFL, FedALA, and FedProx still fail to maintain high accuracy after aggregation on this dataset, which, together with the WebDDoS attack, is the only TCPbased attack in our setup.

6.6. Test accuracy

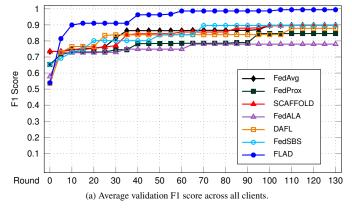
In the previous section, we evaluated the performance of the global models on the clients' validation sets to highlight the impact of different aggregation and client selection strategies on the learning process. We now assess the final global models on the clients' test sets, simulating the deployment scenario in which the global model is distributed to all clients and applied to unseen data. As in the training and validation phases, we assume that each client's test set contains only samples of a single attack type (along with benign traffic).

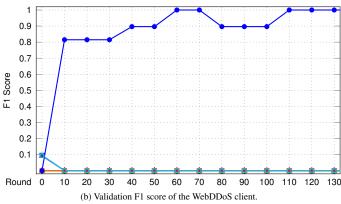
The results reported in Table 6 confirm the trend observed during the FL process and described in Section 6.5. Overall, we observe that most attacks are correctly identified, F1 score over 0.9, by all methods, with the exception of *WebDDoS*, *Syn*, and *MSSQL*, which present significant challenges. These outlier cases are particularly useful for evaluating the strengths and weaknesses of each approach.

Notably, SCAFFOLD emerges as the second-best method after FLAD, achieving performance comparable to FEDAvg in terms of average F1 score and standard deviation. However, it incurs higher computational costs due to the use of control variates. Although its correction mechanism proves effective, it still

Table 6: Test set accuracy (F1 score). The values in bold indicate the scores that are significantly lower than the others.

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	Mean	Std_dev
FedAvg	0.00	0.96	0.93	0.97	1.00	0.99	0.96	1.00	0.99	1.00	1.00	0.94	0.92	0.897	0.271
FedProx	0.00	0.96	0.98	0.93	0.95	0.95	0.92	0.93	0.28	0.91	0.93	0.93	0.93	0.815	0.306
SCAFFOLD	0.00	0.95	0.92	0.97	1.00	0.99	0.95	1.00	0.99	1.00	1.00	0.92	0.92	0.893	0.270
FedALA	0.00	0.98	0.97	0.97	1.00	0.98	0.97	0.99	0.00	0.92	0.99	0.96	0.85	0.814	0.363
DAFL	0.00	0.98	1.00	0.98	1.00	0.99	1.00	1.00	0.21	0.97	1.00	0.99	1.00	0.855	0.336
FedSBS	0.00	0.96	0.95	0.95	0.98	0.97	0.94	0.97	0.96	0.97	0.97	0.93	0.85	0.877	0.266
FLAD	0.93	0.98	0.99	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.984	0.018





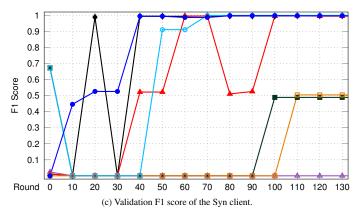


Figure 8: F1 score of the global model over increasing rounds, evaluated on the validation sets: (a) average score across all clients, (b) WebDDoS client, and (c) Syn client. Results are from the longest experiment (133 rounds).

struggles on the WebDDoS attack, indicating that a variate correction alone is insufficient in this case. Other approaches, such as FedProx, FedALA, DAFL and FedSBS, yield lower average

F1 scores and underperform on at least two attack types.

In summary, although all the methods are designed to address the client-drift issues of FedAvg, only the adaptive mechanism of FLAD succeeds in learning effectively from all the o.o.d. datasets. This suggests that requiring additional information from clients, such as validation accuracy scores, to optimise client selection and local training is crucial for managing heterogeneous data distributions in FL.

7. Discussion and conclusion

This study evaluates the performance of state-of-the-art FL algorithms for cybersecurity, with a focus on network intrusion detection where ML models are trained on network traffic. We considered six algorithms designed to address the limitations of FedAvg in handling non-i.i.d. and unbalanced datasets. FedProx, FedALA, and SCAFFOLD were originally proposed for general applications and primarily tested on image data, whereas DAFL, FedSBS, and FLAD were specifically developed for network security. The objective of this work is to assess how these algorithms perform in cybersecurity scenarios and to quantify the cost of their improvements over FeDAvg. To this end, we compared them with FedAvg in terms of network overhead, client selection strategies, process duration, and global model accuracy in classifying DDoS attacks before and after aggregation. Table 7 provides an overview of the results, summarising the average metrics for each method over 10 test

In terms of accuracy, only FLAD achieves top scores across all attack types. It leverages client validation scores to guide selection and allocate training, ensuring low-performing clients train more often and for longer. This adaptive strategy enables FLAD to produce a global model that consistently detects all attacks, including o.o.d. ones. The cost is a higher network overhead and longer training time compared to FedAvg and other methods.

The network overhead is not a critical issue in our setting, as the model size is small and the communication infrastructure is efficient. However, in scenarios with larger models or limited bandwidth, it could become a bottleneck. In this regard, our results show that FedAvg is the most efficient, while DAFL and FedSBS incur in more than double the overhead of FedAvg. DAFL suffers from high overhead due to its client selection strategy, which involves all clients in every round (100% in the Table). FedSBS, on the other hand, requires additional communication to compute and exchange performance scores among clients for selection purposes.

Table 7: Overall summary of the results for each FL method. Client selection (rounds/clients in percentage), network overhead (MB), duration (minutes), and test accuracy (average F1 score across the clients). The best results for each metric are highlighted in bold.

Method	Client Selection	Network overhead	Overall Duration	Test accuracy
FedAvg	46	34.59	8.27	0.897
FedProx	46	34.59	8.88	0.815
SCAFFOLD	46	60.25	8.71	0.893
FedALA	46	34.59	12.74	0.814
DAFL	100	71.98	13.01	0.855
FedSBS	46.2	72.06	11.33	0.877
FLAD	39.4	52.18	15.26	0.984

The duration of the FL process is a key metric, as fast algorithms enable timely updates of the global model against new attack types, while slow ones leave clients exposed to emerging threats. FedProx and SCAFFOLD achieve durations similar to FedAvg thanks to their low client overhead, whereas DAFL, FedSBS, and FedALA have shown slower without accuracy gains. FLAD is the slowest due to its dynamic assignment of training epochs and MBGD steps, which increases client training time and makes most rounds longer than those of FedAvg and methods with fixed local training.

Overall, our results show that there is no perfect solution for training ML models for cybersecurity in federated settings, particularly with non-i.i.d. and unbalanced datasets. A combination of strategies may offer the best trade-off between accuracy and duration. FLAD stands out as the most effective approach to ensure that the global model learns all attack types, though its long training rounds could be mitigated by adjusting the ranges of local training epochs and MBGD steps. Techniques from FedProx (regularisation) and SCAFFOLD (control variates) may also help reduce client drift, easing the reliance on FLAD's adaptive focus. Finally, the weighted averaging of FedAvg penalises attacks present in smaller datasets; in pathological scenarios like ours, a simple arithmetic mean appears more effective.

Acknowledgment

This work was supported by Ministero delle Imprese e del Made in Italy (IPCEI Cloud DM 27 giugno 2022 - IPCEI-CL-0000007) and European Union (Next Generation EU).

References

[1] U. Ahmad, M. Han, A. Jolfaei, S. Jabbar, M. Ibrar, A. Erbad, H. H. Song, and Y. Alkhrijah, "A comprehensive survey and tutorial on smart vehicles: Emerging technologies, security issues, and solutions using machine learning," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

- [2] R. Rajesh, S. Hemalatha *et al.*, "Threat detection and mitigation for tactile internet driven consumer iot-healthcare system," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 4249–4257, 2024.
- [3] Y. Xue, J. Pan, Y. Geng, Z. Yang, M. Liu, and R. Deng, "Real-Time Intrusion Detection Based on Decision Fusion in Industrial Control Systems," *IEEE Transactions on Industrial Cyber-Physical Systems*, vol. 2, pp. 143–153, 2024.
- [4] M. Abdelaty, R. Doriguzzi-Corin, and D. Siracusa, "AADS: A noise-robust anomaly detection framework for industrial control systems," in *Proc. of International Conference on Information and Communications Secu*rity, 2019, pp. 53–70.
- [5] M. Elsisi, J.-T. Yu, C.-C. Lai, and C.-L. Su, "A drone-assisted deep learning-based iot system for monitoring ship emissions in ports considering adversarial attacks," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–11, 2024.
- [6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelli*gence and statistics, 2017.
- [7] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.
- [8] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. of International* conference on machine learning. PMLR, 2020.
- [9] J. Zhang, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and H. Guan, "FedALA: Adaptive Local Aggregation for Personalized Federated Learning," *Proc. of the AAAI Conference on Artificial Intelligence*, 2023.
- [10] H. N. Cunha Neto, J. Hribar, I. Dusparic, N. C. Fernandes, and D. M. Mattos, "FedSBS: Federated-Learning participant-selection method for Intrusion Detection Systems," *Computer Networks*, vol. 244, p. 110351, 2024.
- [11] J. Li, X. Tong, J. Liu, and L. Cheng, "An efficient federated learning system for network intrusion detection," *IEEE Systems Journal*, vol. 17, no. 2, pp. 2455–2464, 2023.
- [12] R. Doriguzzi-Corin and D. Siracusa, "FLAD: Adaptive Federated Learning for DDoS attack detection," *Computers & Security*, vol. 137, p. 103597, 2024.
- [13] R. Doriguzzi-Corin, S. Cretti, and D. Siracusa, "Resource-efficient federated learning for network intrusion detection," in 2024 IEEE 10th International Conference on Network Softwarization (NetSoft), 2024.

- [14] University of New Brunswick, "DDoS Evaluation Dataset," 2019. [Online]. Available: https://www.unb.ca/cic/datasets/ddos-2019.html
- [15] P. Sabel, S. Cretti, and R. Doriguzzi-Corin, "Comparison of Federated Learning Algorithms for Cybersecurity under Non-IID and Unbalanced Settings," 2025. [Online]. Available: https://gitlab.fbk.eu/fl/fl-comparison
- [16] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," in *Proc. of IEEE 38th international conference on data engineering (ICDE)*, 2022.
- [17] G. A. Baumgart, J. Shin, A. Payani, M. Lee, and R. R. Kompella, "Not all federated learning algorithms are created equal: A performance evaluation study," 2024. [Online]. Available: https://arxiv.org/abs/2403.17287
- [18] E. M. Campos, P. F. Saura, A. González-Vidal, J. L. Hernández-Ramos, J. B. Bernabé, G. Baldini, and A. Skarmeta, "Evaluating Federated Learning for intrusion detection in Internet of Things: Review and challenges," *Computer Networks*, vol. 203, p. 108661, 2022.
- [19] J. Dai, "Comparative analysis of federated learning algorithms under non-IID data," *Applied and Computational Engineering*, vol. 86, pp. 91–100, 07 2024.
- [20] Q. H. Nguyen, S. Hore, A. Shah, T. Le, and N. D. Bastian, "FedNIDS: A Federated Learning Framework for Packet-Based Network Intrusion Detection System," *Digital Threats*, vol. 6, no. 1, Feb. 2025.
- [21] Y. Liu, Z. Wang, S. Pang, and L. Ju, "Distributed Malicious Traffic Detection," *Electronics*, vol. 13, no. 23, 2024.
- [22] D. Chai, L. Wang, L. Yang, J. Zhang, K. Chen, and Q. Yang, "Fedeval: A holistic evaluation framework for federated learning," 2022. [Online]. Available: https://arxiv.org/abs/2011.09655
- [23] A. Deshmukh, P. E. de la Rosa, R. V. Rodriguez, and S. Dasari, "Enhancing Privacy in IoT-Enabled Digital Infrastructure: Evaluating Federated Learning for Intrusion and Fraud Detection," *Sensors*, vol. 25, no. 10, 2025.
- [24] Z. Sun, X. Niu, and E. Wei, "Understanding generalization of federated learning via stability: Heterogeneity matters," in *Proc. of International conference on artificial in*telligence and statistics. PMLR, 2024.
- [25] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proc. of the Second Workshop on Distributed Infrastructures for Deep Learning*, 2018.
- [26] E. Sorbera, F. Zanetti, G. Brandi, A. Tomasi, R. Doriguzzi-Corin, and S. Ranise, "Adaptive federated learning with functional encryption: A comparison

- of classical and quantum-safe options," arXiv preprint arXiv:2504.00563, 2025.
- [27] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *Proc. of IEEE INFOCOM*, 2020.
- [28] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, "Federated learning with non-iid data: A survey," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19188–19209, 2024.
- [29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [30] Eclipse Foundation. (2025) Eclipse Mosquitto: An open source MQTT broker. [Online]. Available: https://mosquitto.org/
- [31] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
- [32] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincon, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for ddos attack detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876–889, 2020.
- [33] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in 2019 International Carnahan Conference on Security Technology (ICCST). IEEE, 2019, pp. 1–8.
- [34] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2018, no. 1, pp. 177–200, 2018.
- [35] X. Li and H. Fu, "SecureBERT and Llama 2 Empowered Control Area Network Intrusion Detection and Classification," *IEEE Transactions on Intelligent Transportation* Systems, 2025.