

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

Roberto Doriguzzi-Corin

rdoriguzzi@fbk.eu

Fondazione Bruno Kessler

Silvio Cretti

Fondazione Bruno Kessler

Petr Sabel

University of Trento

Silvio Ranise

Fondazione Bruno Kessler

Research Article

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

Posted Date: October 23rd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11039682/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

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

Roberto Doriguzzi-Corin^{a,1}, Silvio Cretti¹, Petr Sabel², Silvio Ranise^{1,2}

¹Cybersecurity Centre, Fondazione Bruno Kessler, Trento - Italy

²University of Trento, 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 shown poor convergence in heterogeneous environments characterised by non-independent and identically distributed (i.i.d.) data distributions and unbalanced dataset sizes across clients, conditions frequently observed in cybersecurity contexts. To overcome these challenges, several alternative FL algorithms have been developed, yet their applicability to network intrusion detection remains insufficiently explored.

This study systematically evaluates a range of FL algorithms in the context of network intrusion detection for DDoS attacks. Using a dataset of recent network attacks, the evaluation considers detection performance, training time, and communication overhead under non-i.i.d. settings, providing practical insights into the selection of FL solutions for network intrusion detection.

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 remains under the control of the data owners (also called “clients” in the FL context) and is not shared with other parties. 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.

^aCorresponding author. E-mail: rdoriguzzi@fbk.eu

Despite its potential, FL 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 also arise in the cybersecurity domain, where clients may observe highly diverse traffic patterns and feature distributions for both benign and malicious activities. For example, some clients may mostly 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, starting from FEDAVG, the canonical FL algorithm [6], and subsequently extending it to address specific challenges. For instance, FedProx [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. Despite

these advancements, there's still no single consensus on the most effective approach, 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 network intrusion detection. While some of the algorithms were originally designed for tasks unrelated to cybersecurity, our primary objective is to answer the following research question:

How do different strategies for improving the FL process, including informed client selection, optimised aggregation, and enhanced local training, affect detection performance, training duration, and communication cost under heterogeneous and unbalanced client data distributions?

Our evaluation focuses on key metrics such as training time, communication overhead, 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 Network Intrusion Detection Systems (NIDSs) must rapidly incorporate updates related to newly observed attack profiles in order to remain effective against evolving threats. To ensure reproducibility, we configured a testbed based on the Flower framework [14, 15], which allows for the implementation of FL algorithms and the management of communication between clients and the server. As a data source, we utilise recent benchmark datasets that includes both benign traffic and 15 distinct types of DDoS attack patterns [16].

To evaluate the algorithms, we consider a federated setting characterised by severe client-level heterogeneity. Each client contains benign traffic and one specific attack type, while the amount of data varies by more than four orders of magnitude across clients. Consequently, although each local dataset is approximately balanced between benign and malicious samples, the federation is highly unbalanced with respect to both client dataset size and attack-type representation. 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 state-of-the-art FL methods:** we review a set of FL algorithms, including methods originally designed for other domains, and adapt them to the requirements of NIDSs for DDoS detection.
- **Multi-dimensional comparative evaluation:** we compare the considered FL algorithms in terms of detection performance, model generalisation, training time, and communication overhead, providing a comprehensive assessment of their practical suitability for cybersecurity applications.

- **Analysis under heterogeneous and unbalanced conditions:** we investigate how these algorithms cope with a challenging non-i.i.d. setting in which clients observe unique attack profiles and different data volumes, highlighting the effectiveness and limitations of different approaches to client selection, model aggregation, and local training.
- **Open-source implementation and reproducibility:** we provide an open-source implementation of the evaluated algorithms and the experimental setup (dataset included), enabling other researchers to reproduce our results and build upon our work [17], [18].

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 provides a comprehensive description of the experimental setup, including the evaluation environment, the dataset 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 [19, 20]. 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 [21] [22] [23] [24]. However, as discussed in [20], 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 [20] [25] [26]. 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 [25] [20] [26].

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

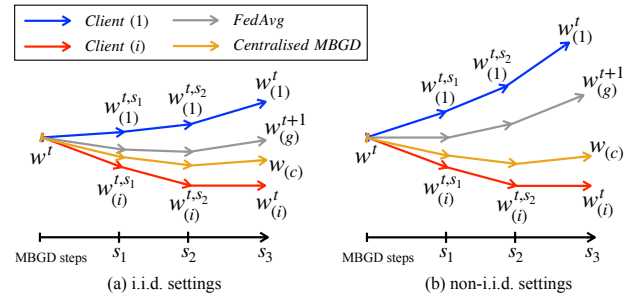


Fig. 1: Weight divergence of FEDAVG in non-i.i.d. settings.

contrasts with works such as [20] and [23], which rely on GPU-accelerated training for larger models.

On the server side, optimisation is particularly important in FL [20] [25], 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 [20].

Most prior works [27] [9] [19] [22] and benchmark frameworks [25] assess FL algorithm performance using datasets like CIFAR-10 or MNIST, which differ substantially from NIDS data in both structure and semantics. Other studies [28] distribute equal-sized datasets with differing labels among clients. Instead, we vary both label distribution and dataset size across clients.

While there is a growing body of research dedicated to assessing the privacy implications of FL methods [25, 29], 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 [24], we maintain a clear focus on decentralised learning. 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 settings [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, with the aim of highlighting their relative strengths and limitations in network intrusion detection under highly heterogeneous data distributions.

3 Limitations of Federated Averaging

Recent research [30, 31] has underscored the limitations of FEDAVG when applied to non-i.i.d. and unbalanced datasets distributed across participating clients. In partic-

ular, 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:

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

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 $h_{\mathbf{w}^t}(\mathbf{x})$ 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:

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

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.

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
$\mathbf{w}_{(i)}^t$	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 -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

Moreover, when datasets sizes 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 under-representing 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 show slow convergence and unstable behaviour, highlighting the challenges posed by non-i.i.d. and unbalanced data in FL.

These limitations represent a fundamental drawback of FEDAVG [30]. 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 traffic distributions and unbalanced amounts 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 $\mathbf{w}_{(i)}^t$.
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:

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

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 FedProx, 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:

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

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:

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

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 $\Delta c_{(i)} = c_{(i)}^{t+1} - c^t$, 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 [32], 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 $\mathbf{w}^t$ with the previous local model $\mathbf{w}_{(i)}^{t-1}$ as formulated as follows:

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

where $\odot$ is the Hadamard product, $\theta_{(i)}^p$ denotes the aggregation weights for the higher layers, whereas $1^{|\mathbf{w}_{(i)}^{t-1}|-p}$ represents the identity matrix applied to the remaining $|\mathbf{w}_{(i)}^{t-1}| - p$ 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 $\theta_{(i)}^p$ frozen:

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

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 and evaluated to address the convergence issues of FEDAVG in cybersecurity scenarios, particularly in the presence of heterogeneous distributions of network traffic features across clients. DAFL and FedSBS were evaluated on the CIC-IDS2018 [33] and CIC-IDS2017 [34] datasets of network traffic, which include Brute Force, DoS/DDoS, Infiltration, Botnet, and SQL Injection attacks. FLAD was evaluated on the CIC-DDoS2019 a dataset containing 13 types of DDoS attacks [35]. Each algorithm introduces modifications to the different stages 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 the 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:

$$\mathbf{w}^{t+1} = \sum_{i \in S'} \frac{\rho_{(i)} a_{(i)}}{\sum_{j \in S'} \rho_{(j)} a_{(j)}} \mathbf{w}_{(i)}^t \quad \text{where:}$$

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

where S' 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:

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

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

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

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 $1 - \varepsilon$, or in a random fashion with probability ε , with $0 < \varepsilon < 1$. 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:

$$\mathbf{w}^{t+1} = \sum_{i \in S'} \frac{|D_{(i)}|}{\sum_{i \in S'} |D_{(i)}|} (\mathbf{w}^t - \eta_g (\mathbf{w}_{(i)}^t - r(\mathbf{w}_{(i)}^t)))$$

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 approach. 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 clients' IG scores, which the server uses to determine the next round's participants.

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 heterogeneous DDoS attack data across the clients. The method is designed to

Table 2: Main features of the FL 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 averaging 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

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 validation 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. The average accuracy score across clients is used as a metric to evaluate the overall performance of the global model and to determine 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 8-core Intel Xeon Silver 4110 @ 2.1 GHz CPUs and 64 GB of RAM.

Each FL algorithm has been implemented as a Flower strategy. Flower [14, 15] is a framework that provides a high-level interface for implementing FL algorithms and managing the communication between the server and clients. The system has been deployed using the Flower Deployment Runtime on a single server machine, with the server and clients running as separate processes. Communication between the server and clients is handled through the gRPC [36] protocol integrated into the Flower framework, making it completely transparent to both the server and client applications. The source code of the environment is publicly available for evaluation and testing [17].

5.1 Algorithms and hyperparameters

The FL algorithms and the global ML model are implemented in Python, using the Tensorflow framework v2.21 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 7 out of 15 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 all the clients are selected for training, but their participation in the aggregated model is based on their local accuracy scores.

Table 3: Hyperparameters used for each algorithm.

Algorithm	Hyperparameters
FEDAVG	Selection ratio: 0.5
	Local epochs E : 1
	Batch size B : 50
	MBGD steps/epoch: $\frac{\text{training samples}}{B}$
FedProx	Regularization weight λ : 1
	Selection ratio: 0.5
	Local epochs E : 1
	Batch size B : 50
SCAFFOLD	Global learning rate η_g : 1
	Selection ratio: 0.5
	Local epochs E : 1
	Batch size B : 50
FedALA	Higher layers p : 1
	Selection ratio: 0.5
	Local epochs E : 1
	Sample ratio: 0.8
DAFL	Batch size B : 50
	MBGD steps/epoch: $\frac{\text{training samples}}{B}$
	Threshold accuracy β : 0.75
	Selection ratio: accuracy-based
FedSBS	Local epochs E : 1
	Batch size B : 50
	MBGD steps/epoch: $\frac{\text{training samples}}{B}$
	Regularization weight λ : 1
FLAD	ϵ min: 0.1
	Initial ϵ : 1.0
	Temperature: 5.0
	Beta momentum: 0.9
FLAD	Server learning rate: 1.0
	Selection ratio: 0.5
	Local epochs E : 1
	Batch size B : 50
FLAD	MBGD steps/epoch: $\frac{\text{training samples}}{B}$
	Patience value: 25 rounds
	Selection ratio: accuracy-based
	Min local epochs E : 1
FLAD	Max local epochs E : 5
	Min MBGD steps/epoch: 10
	Max MBGD steps/epoch: 1000

As summarised in Table 3, the number of local epochs is set to $E = 1$, 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 per epoch is determined by dividing the number of samples in the local dataset by the batch size ($B = 50$). Alternative configurations, such as $E = 20$ or $B = 10$, have also been adopted in previous work (e.g., McMahan et al. [6]). However, our preliminary experiments showed that the modest

improvement in accuracy obtained by increasing the number of local MBGD steps did not justify the substantial increase in local computation, particularly for clients with large datasets. Note that FLAD is the only method that dynamically adjusts both the number of local epochs and the batch size, resulting in different numbers of epochs and MBGD steps across clients and communication 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.

5.2 The Dataset

The evaluation is performed with recent datasets of DDoS attacks, IDS2017 [34, 37], IDS2018 [33, 37] and DDoS2019 [35, 38], provided by the Canadian Institute of Cybersecurity of the University of New Brunswick. These datasets contain several days of network activity, provided in the form of packet capture files (pcap). In this work, the pcap files are processed to extract network flows using the LUCID network traffic parser eXtended (LUCX) [39], which aggregates packet-level features into flow-level representations. Each flow is represented as a two-dimensional array of shape $(10, 32)$, where 10 is the number of packets extracted from the traffic traces for each flow and 32 is the number of packet-level features. These features include packet length, IP and TCP flags, and other relevant properties that characterise a traffic flow.

The aforementioned datasets include both benign traffic and several types of network attacks. The benign traffic in the dataset was generated using the B-profile [40], 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 15 types of DDoS attacks, 13 of which are included in the DDoS2019 dataset, while the remaining two are found in IDS2017 and IDS2018. The attacks are described as follows:

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.
5. **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.
14. **LOIC**: A DDoS attack tool that generates a high volume of TCP, UDP, or HTTP packets with the goal of disrupting service.
15. **HOIC**: A DDoS attack tool that works via an application layer HTTP Flood DDoS attack, flooding a victim's server with HTTP GET and POST requests with the goal of overloading the server's request capacity.

As described in previous work [12], these attacks present disjoint feature distributions and different traffic volumes, enabling the configuration of non-i.i.d. and unbalanced FL scenarios. Compared to previous work, this study includes two additional TCP-based attack types (LOIC and HOIC) to better represent realistic FL scenarios with a more balanced distribution between TCP and UDP-based attacks.

5.3 Clients

The evaluation environment consists of 15 clients and a single central server that communicates with all clients. 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 Table 4. This scenario simulates an FL environment where clients possess heterogeneous data both in terms of distribution and quantity.

Table 4: Overview of the experimental environment. Each client is assigned a dataset containing benign traffic and a single DDoS attack type.

Dataset	Client/Attack	Train	Validation	Test
DDoS2019	1-WebDDoS (TCP)	323	37	40
	2-LDAP (UDP)	646	72	82
	3-Portmap (UDP)	1,296	145	161
	4-DNS (UDP)	2,592	288	320
	5-UDPLag (UDP)	5,182	578	640
	6-NTP (UDP)	10,368	1,152	1,280
	7-SNMP (UDP)	20,732	2,304	2,566
	8-SSDP (UDP)	41,472	4,608	5,120
	9-Syn Flood (TCP)	82,934	9,222	10,251
	10-TFTP (UDP)	165,888	18,433	20,481
	11-UDP Flood (UDP)	331,776	36,864	40,960
	12-NetBIOS (UDP)	663,552	73,728	81,920
	13-MSSQL (UDP)	1,327,108	147,460	163,841
IDS2017	14-LOIC (TCP)	161,060	17,896	19,885
IDS2018	15-HOIC (TCP)	1,480,599	164,512	182,791
Total		4,105,697	456,299	507,338

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.

5.3.1 Global model

The global model is a 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 [41].

The input to the model is an encoded representation of a network flow that aggregates packet-level features. As described in Section 5.2, a single sample is a two dimensional array of shape (10,32), where 10 is the number packets extracted from the traffic traces for each flow and 32 is the number of packet-level features. 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.

Table 5: Average network bandwidth (MB) consumed by each client across all 10 experiments.

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	LOIC	HOIC	Total
FEDAVG	6.00	5.89	5.89	5.89	5.80	6.01	6.05	6.02	6.10	6.13	6.05	6.25	6.00	5.73	6.13	90
FedProx	6.00	5.89	5.89	5.89	5.81	6.02	6.05	6.03	6.10	6.13	6.05	6.25	6.00	5.73	6.13	90
SCAFFOLD	11.96	11.74	11.76	11.74	11.58	11.99	12.07	12.01	12.16	12.23	12.07	12.47	11.98	11.43	12.23	179
FedALA	6.00	5.89	5.90	5.89	5.81	6.02	6.05	6.03	6.10	6.14	6.05	6.25	6.01	5.73	6.14	90
DAFL	12.13	12.83	12.84	12.84	12.84	12.84	12.84	12.84	12.84	12.84	12.84	12.84	12.84	12.82	12.83	192
FedSBS	5.97	6.16	5.94	5.80	5.97	5.91	6.23	6.01	6.03	6.06	6.13	5.79	6.14	5.93	5.94	90
FLAD	16.90	17.54	16.19	7.86	17.20	16.63	7.32	6.54	8.00	6.56	7.04	13.65	7.23	17.91	11.68	178

6 Evaluation results

As anticipated in the previous sections, the experiments are designed to evaluate the performance of FL algorithms in *pathological* settings with non-i.i.d. and unbalanced data distributions across clients. 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. With these experiments, we aim to provide a comprehensive understanding of the strengths and limitations of each algorithm in the context of cybersecurity, where the global model must generalise across diverse attack types, be updated quickly to minimise the time clients remain exposed to new attacks, and incur minimal network overhead to avoid saturating communication channels in scenarios with limited bandwidth and large model sizes.

For each metric, we perform 10 runs with different random seeds to improve the reliability of the results.

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 the clients and the server during training, including global and local model parameters, as well as any additional information required to manage the FL process. The network traffic overhead introduced by the Flower platform is excluded, as it is independent of the specific FL algorithm and common to all evaluated approaches. 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 5.3.1, which consists of more than 11K trainable parameters and has a size of around 45 KBytes.

Table 5 presents the experimental results about network traffic consumption, with FEDAVG as the baseline. The network consumption data for FedProx, FedSBS and FedALA is similar to that of FEDAVG. FedProx and FedALA use the same client selection strategy and do not require any additional data exchange. FedSBS, on the other hand, requires

clients to send their IG scores to the server for client selection. However, the size of these loss values is negligible compared to the size of the model parameters, resulting in a similar network overhead to FEDAVG. On the other hand, the network overhead introduced by SCAFFOLD, DAFL and FLAD differs due to their specific design choices:

- **SCAFFOLD** requires exchanging arrays of control variates, which have the same number of parameters as the model itself.
- **DAFL** always requires all clients to train, although only those that meet a predefined accuracy threshold send their local model to the server.
- **FLAD** evaluates the performance of the global model on local datasets and requires clients to send their local accuracy values to the server. Consequently, the server sends the global model to all clients in every round for evaluation.

With DAFL, the server always sends a copy of the global model to all 15 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 15. In practice, with our setup, all the clients consistently achieve a local accuracy above the threshold ($\beta = 0.75$). The only exception is the WebDDoS client, which fails to meet the accuracy requirement in around 15 rounds on average across all the experiments. This results in approximately 30 model exchanges per round, more than double of the 14 exchanges of FEDAVG.

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 seven selected clients and receives seven local models and seven control variate arrays in return. In total, therefore, 28 data structures are exchanged between the server and clients at each round, close to the 30 exchanged by DAFL. However, 14 of them are control variates, whose size is smaller than a model.

Table 6: 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) and FedSBS choose 7 out of 15 clients in each round, resulting in an average participation rate of about 47%.

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	LOIC	HOIC	Average
FEDAVG-based	46.7	45.8	45.9	45.8	45.2	46.8	47.1	46.9	47.4	47.7	47.1	48.6	46.7	44.6	47.7	46.7
DAFL	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
FedSBS	46.4	47.9	46.2	45.1	46.4	45.9	48.4	46.7	46.9	47.2	47.7	45.0	47.7	46.1	46.2	46.7
FLAD	81.4	86.4	75.9	11.0	83.7	79.3	6.8	0.7	12.1	0.9	4.6	56.1	6.1	89.3	40.8	42.3

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), which compute the IG score just after training, FLAD adaptively adjust the number of clients selected in each round as training progresses (see Figure 2), 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. As shown in Table 5, FLAD significantly reduces network consumption for clients whose attack traffic is easier to learn, such as DNS, SNMP, SSDP, Syn, TFTP, UDP and MSSQL. However, it is important to note that FLAD always requires all clients to receive the global model in each round for evaluation, which contributes to its overall network overhead.

As a final consideration, the Deep Learning (DL) models used in these experiments are relatively small (45 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 [42]. These models contain hundreds of millions of trainable parameters and can exceed 1 GByte in size. In such cases, FedProx, FedALA or FedSBS 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 participate 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 selec-

tion 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 6. 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 approach 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 6, clients such as WebDDoS, LDAP, Portmap, UDPLag, NTP and LOIC are selected more frequently than others. A higher number

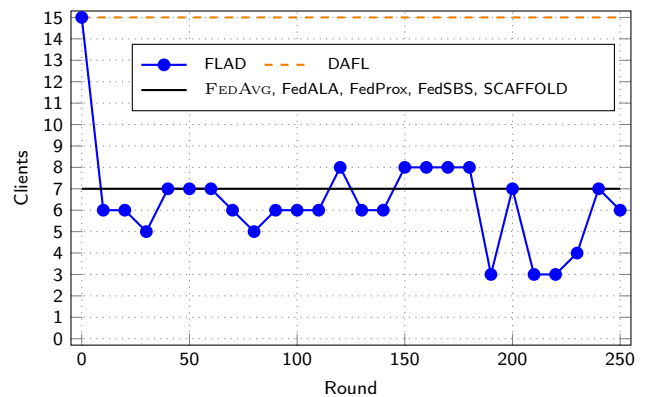


Fig. 2: Clients/round selected during the longest experiment run (250 rounds). The two horizontal lines represent the static number of clients/round selected by DAFL (15 clients) and the other methods (7 out of 15 clients).

of training rounds indicates that more computation is needed to build an effective model for that client.

Figure 2 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 6.42 participants per round. All the other methods select 7 out of 15 clients, which is the result of a selection ratio of 0.5.

6.3 Federated training time

The duration of FL is critical, as timely model updates are essential in cybersecurity to maintain system performance. In 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 3a, which presents the distribution of execution times for each method, reporting the minimum and maximum observed values, the interquartile range, and the average. The reported duration includes the overhead introduced by the Flower framework (including client orchestration, gRPC communication, and message serialization/deserialization), client-side operations (e.g., model training and evaluation), data transmission (model weights, control variates, accuracy scores, etc.), and global model aggregation.

Figure 3b shows the contribution of the clients' local operations to the overall training time. Specifically, it reports the cumulative execution time of the slowest client at each

federated learning round, as the server must wait for all selected clients before proceeding. The results indicate that local operations account for the largest share of the total training time (approximately 60%) for all methods except FLAD. In FLAD, local training represents only about 9% of the total execution time. Consequently, the remaining time (between 4500 and 5500 seconds) is largely dominated by the Flower framework overhead described above, including client orchestration, communication, and synchronization.

The main advantage of FLAD is its adaptive client selection mechanism, which prioritises clients whose data is harder to classify. Client selection is based on the accuracy of the global model on each client's local validation set, so more challenging clients are selected more frequently and assigned more training epochs and steps, while easier clients are selected less often. Since clients train in parallel, the duration of each round is determined by the slowest selected client, typically the one with the largest local dataset. As shown in Table 6, some large clients such as *MSSQL*, *UDP*, and *TFTP* are selected much less frequently than in other methods, reducing the overall duration of the FL process.

Other relevant observations from Figure 3 are:

- The duration of the FL process with **FedProx**, **SCAFFOLD**, **FedALA** and **FedSBS** aligns with that of **FEDAVG**, as they require minimal extra computation to train the clients. FedProx introduces a small overhead due to the additional regularisation term, SCAFFOLD requires the computation of control variates, which does not significantly affect the training time. FedALA requires an additional training phase to estimate the aggregation weights, which adds some overhead, but this is not significant compared to the overall training time. The execution time of FedSBS slightly exceeds that of FEDAVG due to the additional evaluation phase, during which the global model is systematically assessed on each client's local dataset.

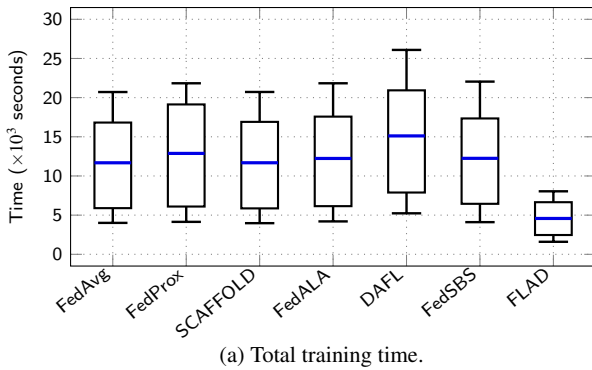
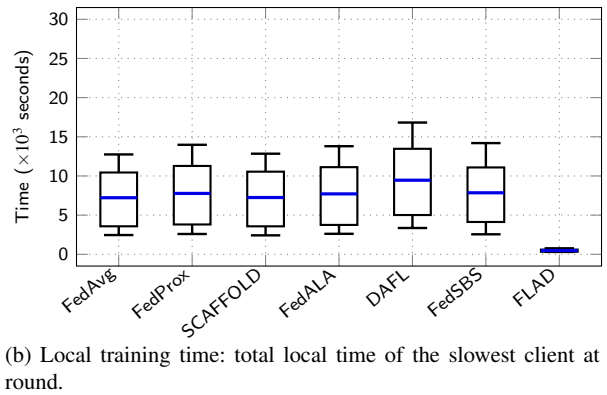


Fig. 3: Distribution of execution durations for each method, reporting the minimum and maximum observed values. The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the average value.

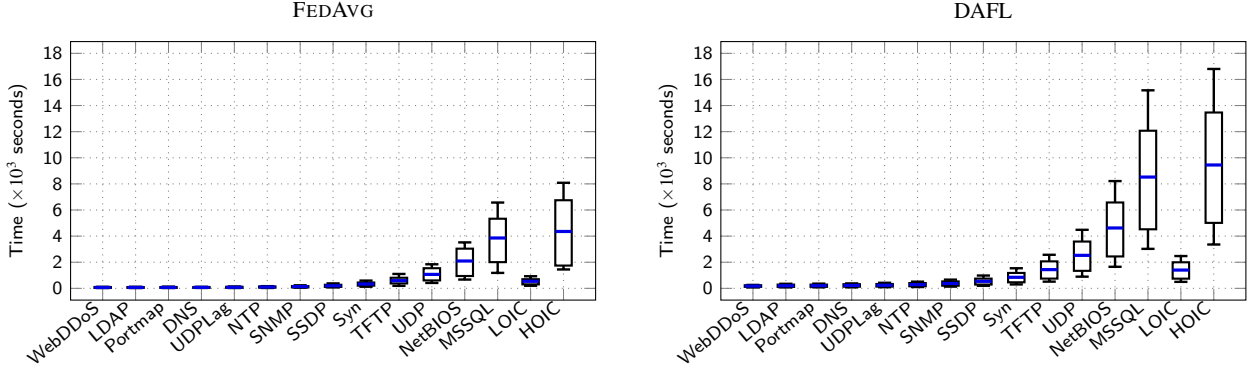


Fig. 4: Distribution of local training times for each client with FEDAVG and DAFL (10^3 scale). The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the average value.

- **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 *HOIC*) determines the overall duration of each round, and with DAFL this client is always selected.

6.4 Local execution time

As mentioned above, the local execution time of each client 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 4 shows the distribution of local training times of each client for FEDAVG, and DAFL. FedProx, SCAFFOLD, FedALA and FedSBS are not included in this analysis, as they show similar results to FEDAVG (see Figure 3b). 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 *HOIC*, require more time to train their local models compared to others. However, while with FEDAVG (as well as FedProx, SCAFFOLD, 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.

A different behaviour is observed with FLAD, as shown in Figure 5 (please note that, in this figure, the y-axis is scaled by a factor of 10^2 to improve readability). The local training time of each client is influenced not only by the size of its local dataset, but also by the number of training rounds, epochs, and steps assigned to it, which are adap-

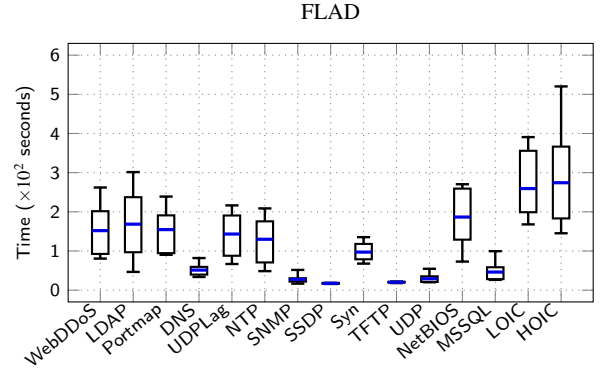


Fig. 5: Distribution of local training times for each client with FLAD (10^2 scale). The boxplots indicate the interquartile range (i.e., the range between the 25th and 75th percentiles), with the horizontal blue line denoting the average.

tively determined based on the performance of the global model on the client’s local validation set. As a result, clients with smaller datasets and harder-to-learn attacks, such as *WebDDoS*, *LDAP*, and *Portmap*, may require a total training time comparable to, or even longer than, that of clients with larger datasets, such as *UDP*, *NetBIOS*, and *MSSQL*. The adaptive approach of FLAD can also be observed by comparing the considerably longer local training time of *LOIC* with that of *TFTP*, despite their local datasets being of similar size. This difference can be attributed to the more challenging attack represented by *LOIC*.

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

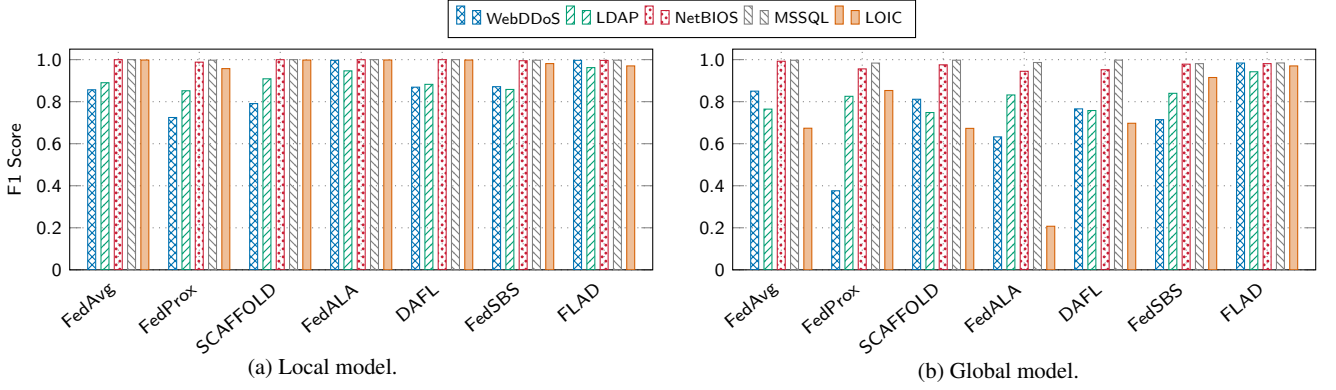


Fig. 6: F1 scores on the validation datasets of the local models (left) and the global model (right) after aggregation. Only the clients that show significant changes in accuracy after aggregation are included.

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.

The plots in Figure 6 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 (P) and recall (R), which are formally defined as follows:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \quad P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN}$$

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

FEDAVG, FedProx, SCAFFOLD, DAFL and FedSBS show comparable performance on the WebDDoS, LDAP and LOIC attacks. While they achieve satisfactory accuracy before aggregation on all the three attacks, after aggregation they struggle to maintain a similar level of accuracy. These findings are consistent with previous research [12], which highlighted the limitations of relying on random client selection and weighted averaging as aggregation strategies in such scenarios (though FedSBS does not rely on purely random client selection). DAFL implements a client selection strategy that prioritises clients with higher validation accuracy. However, this approach does not produce satisfying results of the global model on these three attacks.

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 compared to FEDAVG (see Figure 6a), once the server aggregates these adapted models, the resulting global model loses its ability to generalise to two of the four TCP-based attacks, namely *WebDDoS* and *LOIC* (see Figure 6b).

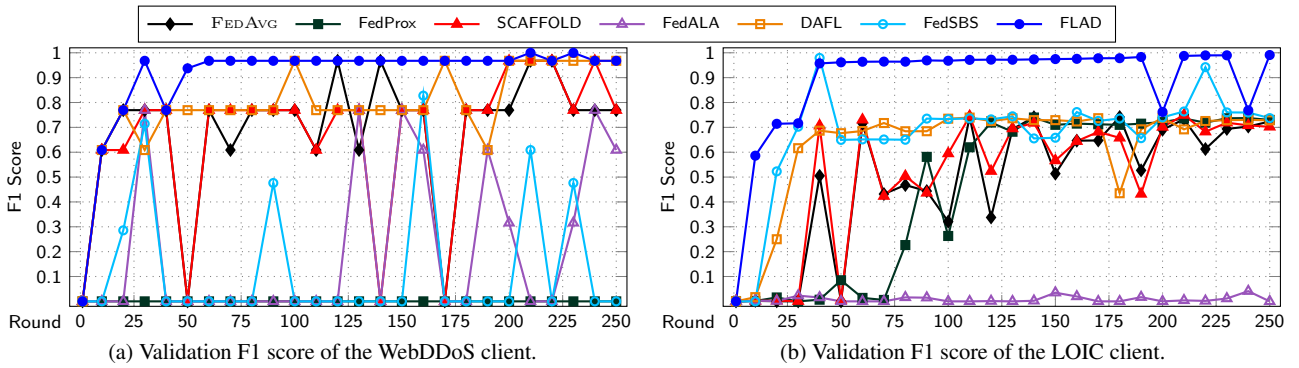


Fig. 7: F1 score of the global model over increasing rounds, evaluated on the validation sets of the WebDDoS and LOIC clients. Results are from the longest experiment (250 rounds).

FLAD shows high robustness to aggregation, producing a global model capable of effectively classifying all 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.

The limitations FEDAVG and the other methods in handling some of the attacks are evident in Figure 7, which report the F1 score of the aggregated global model on the WebDDoS and LOIC attacks 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.

A key observation in this context is the number of rounds required to reach a given F1 score threshold, as fewer rounds allow clients to receive the final updated global model for inference sooner. Figure 8 shows that FLAD and FEDAVG are the fastest methods to reach robust F1 scores of 0.8 and 0.9, while FedProx and FedSBS are the slowest, requiring more than 70 rounds to reach an F1 score of 0.9. FedALA reaches an F1 score of 0.8 in around 45 rounds, but fails to reach 0.9 altogether.

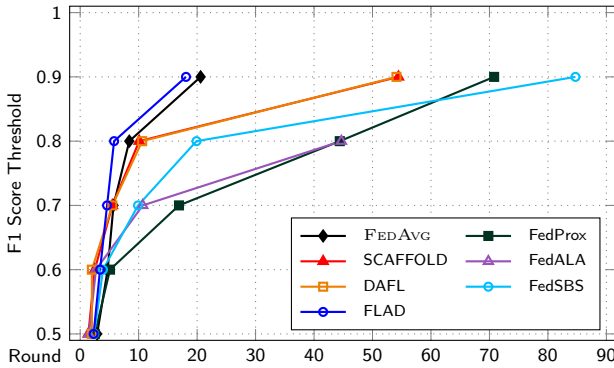


Fig. 8: Rounds taken to reach a certain F1 score threshold.

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 7 are obtained by evaluating each of the 10 global models trained in the previous experiments on each client's test set. For each client, the resulting F1 scores are first averaged across the 10 experiments. The last two columns then report the mean and standard deviation of these per-client average F1 scores across all 15 clients, providing an overall measure of the global model's performance. These results 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*, *LDAP*, and *LOIC*, which present significant challenges for all methods. In particular, the *WebDDoS* and *LDAP* attacks are the most difficult to detect, with F1 scores below 0.8 for all methods except FLAD. This is likely due to the fact that these attacks are underrepresented in the training data, making it difficult for the global model to learn their characteristics due to weighted aggregation and client selection mechanisms that favour clients with more training samples. *LOIC* is also challenging for most methods, with F1 scores below 0.8, except for FLAD. Although the training set size is relatively large for this attack, it is a TCP-based attack, whose feature distribution is significantly different from the other attacks, which are mostly UDP-based.

Notably, DAFL 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 fact that all clients must train in each round. Other approaches, such as FedProx, SCAFFOLD, FedALA and FedSBS, yield lower average F1 scores and underperform on at least two attack types.

Table 7: Test set accuracy (F1 score). The values in bold indicate the scores that are significantly lower than 0.80.

Method	WebDDoS	LDAP	Portmap	DNS	UDPLag	NTP	SNMP	SSDP	Syn	TFTP	UDP	NetBIOS	MSSQL	LOIC	HOIC	Mean	Std_dev
FEDAVG	0.78	0.61	0.88	0.92	0.90	0.93	0.95	0.98	0.91	0.99	1.00	0.96	0.99	0.56	1.00	0.889	0.137
FedProx	0.02	0.77	0.89	0.90	0.91	0.89	0.99	0.99	0.85	0.99	1.00	0.97	1.00	0.68	0.99	0.856	0.248
SCAFFOLD	0.62	0.54	0.89	0.86	0.90	0.91	0.95	0.99	0.91	0.99	1.00	0.96	0.99	0.57	1.00	0.871	0.160
FedALA	0.36	0.46	0.48	0.42	0.60	0.89	0.79	0.92	0.11	0.85	0.95	0.49	0.69	0.01	0.40	0.562	0.287
DAFL	0.77	0.66	0.88	0.95	0.93	0.93	0.95	0.99	0.91	0.99	1.00	0.94	0.99	0.67	1.00	0.904	0.113
FedSBS	0.07	0.65	0.92	0.94	0.97	0.90	0.99	0.99	1.00	1.00	1.00	0.98	1.00	0.73	1.00	0.877	0.245
FLAD	0.97	0.96	0.95	0.98	0.98	0.98	0.98	1.00	0.99	0.99	0.98	0.98	0.98	0.97	0.98	0.978	0.012

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 consider a representative set of FL algorithms addressing the challenges posed by non-i.i.d. and unbalanced datasets. These algorithms differ in how they address client heterogeneity, including modifications to local training, model aggregation and client selection.

The objective of this work is to understand how different FL training algorithms behave in cybersecurity scenarios characterised by challenging client-level heterogeneity and to quantify the resulting trade-offs between detection performance, communication overhead, and training time.

Overall, the results suggest that how clients are selected and how training workloads are distributed across them can be particularly important in strongly non-i.i.d. scenarios. Algorithms that explicitly account for differences between clients and adapt their participation or training workload accordingly achieve higher detection performance than approaches that primarily modify the local optimisation or global aggregation process while maintaining a fixed client participation scheme. In our setting, the most effective algorithms (DAFL and FLAD) dynamically prioritise clients according to their contribution to the global learning process.

This observation is particularly relevant for network intrusion detection. Under highly heterogeneous data distributions, different clients may observe substantially different attack types, with some attacks being available only at a small subset of clients. Treating all clients equally, either by selecting them with a fixed probability or by assigning them the same training workload, does not necessarily ensure that all attack types contribute sufficiently to the global model. Conversely, an informed selection mechanism can focus the training process on clients that provide information currently underrepresented in the global model.

The overall duration of the FL process is also influenced by how client participation and training workloads are managed. Fixed participation schemes may repeatedly involve computationally expensive clients, even when their contribution is not critical to the current training stage. In contrast, adaptive algorithms can reduce training time by excluding clients that provide limited additional benefit. This is particularly relevant in cybersecurity scenarios, where timely updates of the global model are essential to enable a prompt response to new or emerging attack types.

From this perspective, the main finding of our experiments is that informed client selection and adaptive workload distribution provide a particularly effective mechanism for addressing severe non-i.i.d. and unbalanced data distributions. By dynamically determining which clients should

participate and how much training effort should be allocated to them, the federation can make more effective use of the heterogeneous information available across clients. This comes at the cost of additional communication and coordination, highlighting a trade-off between adaptivity and communication efficiency. Nevertheless, the results indicate that, for cybersecurity scenarios characterised by highly heterogeneous attack distributions, investing in informed participation and workload allocation can provide greater benefits than modifying local training or aggregation strategies alone.

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, 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 Security*, pp. 53–70, 2019.
5. M. Elsis, 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 intelligence 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*, 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 *Proc. of 10th IEEE NetSoft Conference*, 2024.
 14. D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, H. L. Kwing, T. Parcollet, P. P. d. Gusmão, and N. D. Lane, “Flower: A friendly federated learning research framework,” *arXiv preprint arXiv:2007.14390*, 2020.
 15. Flower Community, “Flower AI.” <https://flower.ai/>, 2026. Accessed: 2026-09-01.
 16. Canadian Institute for Cybersecurity (CIC), “IDS Datasets.” <https://www.unb.ca/cic/datasets/>, 2026. Accessed: 2026-09-01.
 17. S. Cretti and R. Doriguzzi-Corin, “Comparison of Federated Learning Algorithms for Cybersecurity under Non-IID and Unbalanced Settings.” <https://github.com/daisyfbk/fl-comparison>, 2026. Accessed: 2026-08-31.
 18. S. Cretti and R. Doriguzzi-Corin, “DoS201X Highly Unbalanced (Federated, Pre-Partitioned).” https://huggingface.co/datasets/silviocretti/DOS201X_highly_unbalanced, 2026. Accessed: 2026-09-10.
 19. 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.
 20. 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.
 21. 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, 2022.
 22. J. Dai, “Comparative analysis of federated learning algorithms under non-IID data,” *Applied and Computational Engineering*, vol. 86, pp. 91–100, 07 2024.
 23. 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, Feb. 2025.
 24. Y. Liu, Z. Wang, S. Pang, and L. Ju, “Distributed Malicious Traffic Detection,” *Electronics*, vol. 13, no. 23, 2024.
 25. D. Chai, L. Wang, L. Yang, J. Zhang, K. Chen, and Q. Yang, “Fedeval: A holistic evaluation framework for federated learning,” 2022.
 26. 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.
 27. Z. Sun, X. Niu, and E. Wei, “Understanding generalization of federated learning via stability: Heterogeneity matters,” in *Proc. of International conference on artificial intelligence and statistics*, PMLR, 2024.
 28. 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.
 29. 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,” in *2025 3rd International Conference on Foundation and Large Language Models (FLLM)*, IEEE, 2025.
 30. 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.
 31. Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, “Federated learning with non-iid data: A survey,” *IEEE Internet of Things Journal*, pp. 19188–19209, 2024.
 32. Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
 33. Canadian Institute for Cybersecurity (CIC), “CSE-CIC-IDS2018 on AWS.” <https://www.unb.ca/cic/datasets/ids-2018.html>, 2018. Accessed: 2026-09-01.
 34. Canadian Institute for Cybersecurity (CIC), “Intrusion detection evaluation dataset (CIC-IDS2017).” <https://www.unb.ca/cic/datasets/ids-2017.html>, 2017. Accessed: 2026-09-01.

-
35. Canadian Institute for Cybersecurity (CIC), “DDoS evaluation dataset (CIC-DDoS2019).” <https://www.unb.ca/cic/datasets/ddos-2019.html>, 2019. Accessed: 2026-09-01.
 36. “gRPC: A high performance, open source universal RPC framework.” <https://grpc.io/>, 2026.
 37. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in *International Conference on Information Systems Security and Privacy*, 2018.
 38. 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)*, pp. 1–8, IEEE, 2019.
 39. Roberto Doriguzzi-Corin, “LUCX: LUCID network traffic parser eXtended.” https://github.com/doriguzzi/LUCX_network_traffic_parser, 2025. Accessed: 2026-09-01.
 40. 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.
 41. 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)*, pp. 807–814, 2010.
 42. X. Li and H. Fu, “SecureBERT and Llama 2 Empowered Control Area Network Intrusion Detection and Classification,” *IEEE Transactions on Intelligent Transportation Systems*, 2025.