GöwFed
A novel Federated Intrusion Detection System for IoT devices

Aitor Belenguer Rodriguez

Zuzendaritza
Dirección

Jose A. Pascual Saiz
Konputagailuen Arkitektura eta Teknologia Saila
Departamento de Arquitectura y Tecnología de Computadores

Javier Navaridas Palma
Konputagailuen Arkitektura eta Teknologia Saila
Departamento de Arquitectura y Tecnología de Computadores

eman ta zabal zazu

Universidad del País Vasco — Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Master's Thesis

Computational Engineering and Intelligent Systems Master's Degree

GöwFed
A novel Federated Intrusion Detection System for IoT devices

Aitor Belenguer Rodriguez

Advisors
Jose A. Pascual Saiz
Javier Navaridas Palma

September 2022

Abstract

Intrusion detection systems are evolving into intelligent systems that perform data analysis while searching for anomalies in their environment. The development of deep learning techinques paved the way to build more complex and effective threat detection models. However, training those models may be computationally infeasible in most Internet of Things devices. Current approaches rely on powerful centralized servers that receive data from all their parties – violating basic privacy constraints and substantially affecting response times and operational costs due to the huge communication overheads. To mitigate these issues, Federated Learning emerged as a promising approach, where different agents collaboratively train a shared model, without exposing training data to others or requiring a compute-intensive centralized infrastructure. This work presents **GöwFed**, a novel network threat detection system that combines the usage of **Gower Dissimilarity matrices** and **Federated averaging**. Three different approaches of GöwFed have been developed based on state-of the-art knowledge: (1) a vanilla version; (2) an autoencoder version; and (3) a version counting with an attention mechanism. Furthermore, each variant has been tested using simulation oriented tools provided by TensorFlow Federated framework. In the same way, a centralized analogous development of all the Federated systems is carried out to explore their differences in terms of scalability and performance – across a set of designed experiments/scenarios. Overall, GöwFed pretends to be the first stone towards the combined usage of Federated Learning and Gower Dissimilarity matrices to detect network threats in Internet of Things devices.

Contents

List of Figures

List of Tables

1 Introduction

In the era of digitization, the amount of generated and stored data has increased exponentially. The current trend of storing and analyzing any digital transaction, combined with the cheapening of storage devices and infrastructures, has caused an outburst of database sizes. In parallel, the number of (Industrial) Internet of Things ((I)IoT) devices is increasing due to the establishment of domestic intelligent gadgets, the spread of smart cities and the rapid advancement of Industry 4.0. Information generated by those devices is highly appreciated by big data conglomerates, which rely on data analysis for Business Intelligence and understanding market trends with the ultimate goal of improving products and services. As a consequence, data has become a highly valuable asset that needs to be protected.

Cybersecurity has become an essential element in order to avoid data leakages, malicious intrusions, service availability denials and so on. However, the area involving information security is uncertain and needs to be constantly readjusted in line with the emergence of new attack patterns. When cybersecurity firstly appeared, the number of computers was insignificant and they were reserved for professional usage. In those days, fully sensorized smartphones generating massive network traffic and containing tons of sensitive information did not exist. In this context of security preservation, **Intrusion Detection Systems (IDS)** play an important role by monitoring system activity to proactively detect potential attacks. The evolution of threat detection systems has evolved in tandem with the development of new Machine Learning (ML) techniques. The first generation of IDS was rather rudimentary and simply relied on collating system events against manually updated tuples of a signature database. However, these methods were quickly found to have severe limitations, most critically, in terms of flexibility. Primarily, they lacked proactivity in the sense that they were unable to detect new threats that were not in the signature database. Secondly, the period from when an attack was first discovered until new signatures were produced and updated in the IDS was potentially lengthy, leaving the systems vulnerable for long periods of time.

As a mitigation, second generation IDS started to gradually incorporate some form of intelligence to detect new threats. This way, they were capable of automatically learning attack patterns using basic ML models, e.g., Support Vector Machines (SVM), Random Forests (RF) and so on. The evolution continued with the incorporation of Deep Learning (DL) techniques, which contributed to the advent of more accurate and sophisticated models, e.g., Multilayer Perceptrons (MLP), Recurrent Neural Networks (RNN) and others.

As these systems kept improving in terms of accuracy and new threat detection capabilities, the next natural step is to allow them to share information about newly detected threats so that new attack vectors are promptly recognized by all involved parties and, in turn, global impact is reduced. One possible way of achieving this is the incorporation of centralized learning, in which different parties contribute to the training of a complex model by sending their local data to a centralized computing infrastructure. The whole training process is typically performed in a data center (cloud) which will then distribute the new model parameters to all involved parties.

Nonetheless, performing centralized learning could be infeasible due to traditional information sharing approaches that deal with data in a raw way. That could cause network traffic flow struggle; especially in cases where low resource IoT devices are the main communication agents. Moreover, sharing raw data to third parties is generally discouraged and, indeed, could violate regulations involving data management policies [2]. Therefore, using collaborative learning algorithms with strict data protection policies is vital to achieve good reliability and scalability, as well as a privacy-friendly infrastructure.

In this context, **Federated Learning (FL)** has emerged as a promising tool to deal with the information exchange of different parties and sensitive data exploitation challenges. FL is an avant-garde ML technique that has gained special interest in IoT computing for its reduced communication cost and privacy preserving features [3]. First, raw data located in the end-devices never leaves these devices – following an on-device policy. Instead, it is used to learn internal models and share local model parameters. Then, local parameters from agents are aggregated into a global model following some predefined rules – e.g., by averaging them as in FedAVG [3]. Finally, the consolidated global parameters are sent back to each edge party and the process is iterated until convergence is achieved. Thus, knowledge acquired by collaborating devices is pooled to improve the overall metrics of each local model and obtain improved training scores.

After analyzing the evolution of IDS, we are convinced that FL will conform the backbone of new generation IDS. While FL is a relatively recent technique and its application to IDS technologies is very limited, the designed system intends to propose a novel FL-IDS to detect network threats in IoT devices. The innovation comes from the usage of Gower Distance matrices [4] as the main input for the designed models – in combination with state-of-the-art FL techniques; FedAVG, Attention Mechanism (AM) [5] and so on.

2 Research questions

The main objective of this project is to explore the possibility of creating a FL-IDS, having Gower Distance matrices as inputs. In order to do so, a series of custom FL-IDS are going to be designed and implemented. Those systems will perform supervised binary classification of the incoming network traffic – labeling it as normal or malicious. Moreover, the viability of deploying the designed FL-IDS in low resource, tiny IoT agents needs to be studied. Following B. Li et al. [6] proposal, those IoT agents will monitor network traffic generated by several IoT/IIoT devices, learn a local model based on their behavior and average parameters with other agents of different networks.

However, will the designed systems perform well in comparison to their analogous CNL approaches? As the next natural step after creating FL-IDS prototypes, experimentation on a simulated IoT environment will be carried out to measure how similar FLS and CNL developments are. Nevertheless, beyond statistical metrics involving model evaluation, the number of rounds to achieve convergence and scalability will be taken into account as well. Linked to the previous point, a series of best-practice metrics will be proposed to correctly evaluate the experiments in a standardized way.

Once the feasibility of the designed system is tested; a repository containing all the implementation details and the experimentation results will be made available. Creating a well structured and documented system is a top priority to ensure future contributions to GöwFed's research branch. Additionally, a modular implementation will be developed to ease the creation of new system architectures and facilitate debugging.

3 State of the art

It is essential to carry out a review of the state of the art to summarize existing knowledge and facilitate future research by highlighting some limitations of the literature. In order to have a deeper understating of how current FL-IDS technologies work, we recommend the lecture of our survey [7]. Nonetheless, the main concepts are summarized in the following lines.

3.1 Intrusion detection systems

IDS can be classified into Host-based (HIDS) and Network-based (NIDS). HIDS are typically computing systems that analyze local system data, application registers, log accesses, system calls and so on in order to detect malicious applications [8]. Meanwhile, NIDS focus on network traffic with the aim of finding malicious patterns that target the devices inside a monitored infrastructure [9]. Recent research [10] has extensively shown that leveraging ML techniques for intrusion detection is a highly successful methodology. In the way of learning complex relationships in the data and, in turn, build strong IDS – both in the context of NIDS and HIDS.

3.1.1 Datasets for evaluating IDS

Another important aspect in the life-cycle of advanced IDS is the evaluation of their detection capabilities. There are many popular datasets available for IDS evaluation and, indeed, the main datasets discussed in Section IV are gathered together in Table 3.1. The table uses the following conventions. Raw captures correspond to the availability of the whole captured datagram; usually stored in pcap files. Payload features are extracted from the application data in the dataframe and processed using natural language processing, regular expressions or similar techniques. Single Flow Derived Features (SFD) correspond to a collection of packets sharing any property on the IP and transport layers. SFD features are extracted from the aggregation of a packets flow delimited by a given event (e.g., end of a TCP connection, a timeout and so on). Multiple Flow Derived Features (MFD) correspond to the aggregation of information belonging to multiple flow records, containing higher level statistics (e.g., time window delimited flows, last n flows and so on). Finally, dataset labeling could be done manually (M) by a skillful professional; automatically (A) using a rule repository and a script; or on a scheduled (S) way, launching specific attacks in pre-established time windows. It is also possible to merge some of the mentioned methods (MS, AS).

Table 3.1: A summary of public datasets available for the evaluation of IDS.

Dataset	Raw captures	Payload features	Single flow features	Multi flow features	Labeling method [1]	Domain	Year of capture
AWID3 [11]	✓				-	wireless IoT	2020
IOT-23 [12]	✓	✓	✓	✓	A	IoT	2020
TON_IOT [13]	✓		✓	✓	A	IoT/IIoT	2019
CICIDS-2017 [14]	✓		✓		S	application traffic	2017
ISCXTor2016 [15]	✓		✓		S	application traffic (raw/Tor)	2016
ISCXVPN2016 [16]	✓		✓		S	application traffic (raw/VPN)	2016
WSN-DS [17]			✓		S	wireless IoT	2016
AWID2 [18]	✓		✓		M	wireless IoT	2015
SEA [19]		✓			A	UNIX commands	2001
NSL-KDD [20]		✓	✓	✓	MS	networking	1998

[1] M: manually, A: automatically, S: scheduled.

3.2 Federated Learning

With the aim of achieving a greater understanding about state-of-the-art FL-IDS, this section provides useful background information on FL. Although FL is the main focus of this project, we also address three additional learning paradigms which are typically used as baselines for benchmark comparisons within IoT/Edge infrastructures.

1. **Self learning (SL)** Neither data nor parameters leave the device; training is performed individually by edge devices. SL can be used as a baseline to measure individual learning ability when no information is shared.

2. **Centralized learning (CNL)** Data is sent from different parties to a centralized computing infrastructure, which is in charge of performing the training with all the received data. CNL is used as a yardstick of the learning ability when the models are built using all available data.

3. **Collaborative learning (CL)** Wraps up custom variants of distributed learning (including FL) where involved agents benefit from training a model jointly. Paul Vanhaesebrouck et al. [21] presented a fully decentralized collaborative learning system, where the locally learned parameters are spread and averaged without being under the orchestration of a centralized authority – in a P2P network.

According to Chaoyang He et al. [22], the main limitations of FL when compared with other CNL are concerning statistical heterogeneity, system constraints and trustworthiness. Those challenges have been addressed using different approaches in the literature. For instance, statistical heterogeneity has been tackled by distributed optimization methods such as Adaptive Federated Optimizer [23], FedNova [24], FedProx [25] and FedMA [26]. System constraints, such as communication overheads or high training computation costs [27–33] are mitigated using gradient sparsification [34] and quantization techniques. Finally, to tackle trustworthiness issues, Differential Privacy (DP) and secure multiparty computation (SMPC) privacy mechanisms have been proposed [35–43]. Similarly, new defense techniques to make FL robust against adversarial attacks have been proposed as well [44–53].

3.2.1 Federated Learning Systems

By definition, FL enables multiple parties to jointly train a ML model without exchanging local data. It involves distributed systems, ML and privacy research areas [1, 54], and, since the pioneer FedAVG [3] approach, many new Federated Learning Systems (FLS) have emerged. A general taxonomy describing the difference of those FLS is presented in [1] and replicated in Figure 3.1. This classification is multidimensional and includes the most important aspects of FL architectures including data partitioning, learning model, privacy, communication characteristics and so on.

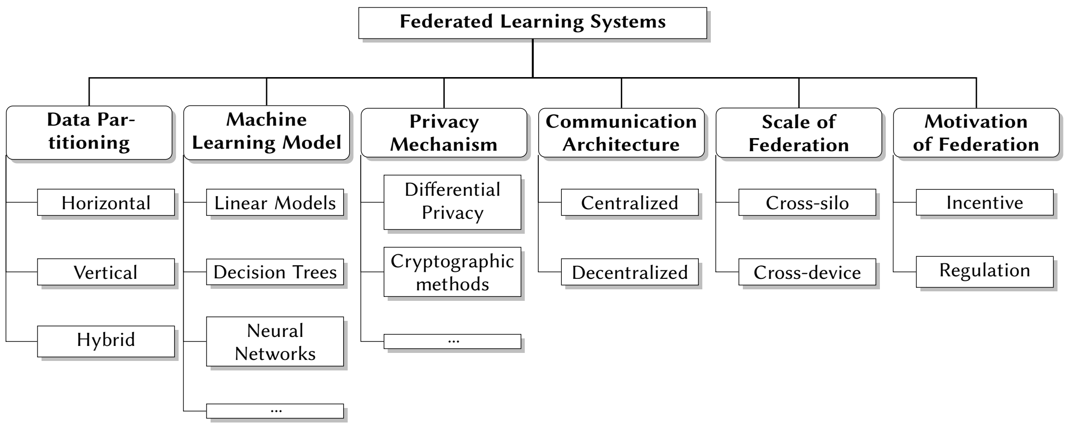


Figure 3.1: A big picture classification of existing Federated Learning Systems presented by Q.Li et Al. [1].

3.2.2 Federated Learning Frameworks

There exist many available frameworks and libraries which can be used to develop FL applications, as thoroughly discussed in [22]. Frameworks can be categorized based on their main objectives: *Simulation-oriented* libraries provide multiple development and benchmark tools, placing the emphasis on extensibility (adding new functionalities) and evaluation purposes (e.g., using simulation and virtual devices). In contrast, *Production-oriented* libraries offer enterprise level solutions, giving support to various FL scenarios, by focusing on usability and productivity (i.e., facilitating system deployment).

Examples of simulation oriented libraries are TensorFlow-Federated (TFF) [55], PySyft [38], LEAF [56] and FedML [22]. Conversely, instances of production oriented libraries are FATE [57] and PaddleFL [58]. Among the APIs mentioned, FedML and PySyft pave the way for the creation of adaptable systems, providing FLS topology and message exchange customization. Nonetheless, regarding the disposal of the parties, all the aforementioned libraries support vanilla FL centralized algorithms (e.g., FedAVG, FedProx). In the same way, FedML, FATE and PaddleFL exclusively incorporate vertical data partitioning. Bearing all this in mind, FedML seems to be the most complete research-oriented library, in terms of supporting multiple FL setups. Additionally, it simplifies codification with a modular worker/client-oriented architecture.

3.3 Employing FL in ID

The combination of both previously explained technologies (IDS in Section 3.1 and FL in Section 3.2) has become a hot topic of research. Considering that the overwhelming majority of IDS rely on DL models, we introduce a taxonomy based on the DL variants employed by the FL-IDS literature illustrated in Figure 3.2.

The proposed classification is performed taking into account the DL model architecture used on each edge device. Since vanilla FL is the *de facto* implementation choice and that many disjointed custom variants of it exist, it is not possible to perform a tree structure taxonomy by type of FL algorithm used. Hence, existing FL-IDS are split into two major groups depending on the NN architecture; Recurrent Neural Networks (RNN) [59] and Multilayer Perceptrons (MLP) [60]. Each group is respectively divided into two subgroups. The RNN models are divided based on the neurons architecture into Long Short-Term Memory (LSTM) [61] and Gated Recurrent Units (GRU) [62]. In contrast, MLP models are divided by the model architecture. In particular, Autoencoders (AE) [63] is considered an important subclass because it is commonly employed in the literature.

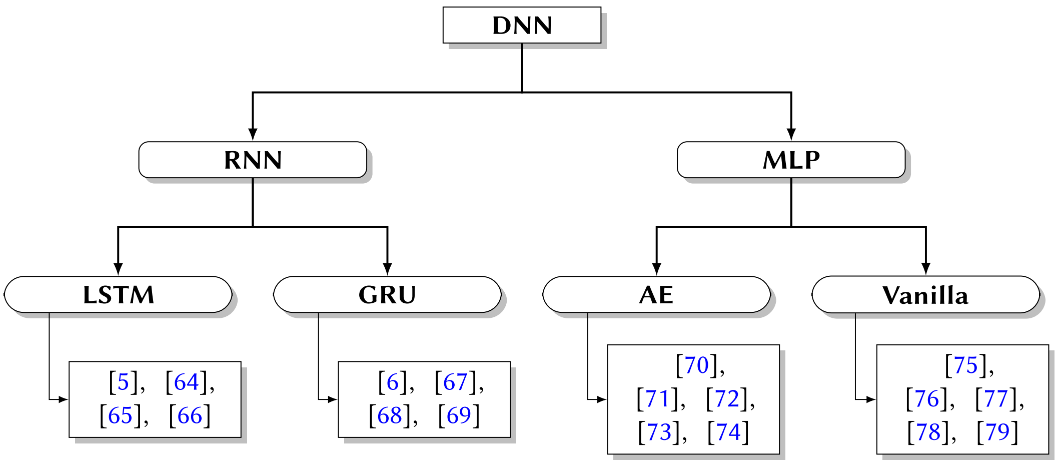


Figure 3.2: Existing Deep Learning Federated Intrusion Detection Systems by model architecture.

3.3.1 Relevant approaches

AEs are the most common FL-IDS architecture [70–74] to perform ID via anomaly detection due to their input reconstruction abilities. Once the usual network traffic patterns are learned, anomalies are translated into high reconstruction loss instances. (1) Qin et al. [70] face the challenge of using high dimensional time series with resource limited IoT devices. A greedy feature selection algorithm is employed to deal with data dimensionality issues as well as a sequential implementation of batch learning is applied to an autoencoder. (2) Tian et al. [74] propose a Delay Compensated Adam (DC-Adam) approach [80] to overcome gradient delay – inconsistency issues in the learning process. Combined with a pre-shared data training strategy to avoid model divergence in non-IID data scenarios.

The utilization of LSTM NNs is interesting due to their ability to process data sequences [5, 64–66]. If network traffic flow is considered as a time series, it becomes a suitable input for a NN using LSTM neurons. Similarly GRU architectures are a good candidate for processing time series [6, 67–69]. In contrast to LSTMs, they do not contain an internal memory. However, their simpler architecture makes the learning process lighter which, in turn, renders it suitable for low resource IoT scenarios. (1) DeepFed [6] is an FL-IDS that introduces the concept of Industrial Agents as network monitoring devices and the usage of advanced privacy mechanisms based on Paillier cryptosystem [81] during the FedAVG learning rounds. (2) Dïot [67] also presents relevant advances by identifying device features connected to a local monitoring agent and maintaining a type specific global anomaly repository via FL.

Moreover, FL-IDS using custom MLP NN architectures are presented in [75–79]. Beyond the mentioned architectures, those approaches focus on data preprocessing and custom learning variants. (1) Al-Marri et al. [77] merge the advantages of FL and mimic learning [82] by training a teacher (private) and a student (public) model per device to then apply FedAVG and create an IDS. (2) Weinger et al. [75] show how SMOTE [83] and ADASYN [84] data augmentation techniques could accelerate model convergence – reducing communication rounds among agents.

4 Development

Among the discussed FL frameworks in Section 3.2.2, TensorFlow Federated is selected to carry on GöwFed's developments. Although it is not as complete as FedML in terms of dedicated IoT functionalities, it is extensively documented and has powerful simulation oriented tools as well. Furthermore, the selected dataset to work in simulated environments is TON_IOT [13], due to its versatility; wrapping up vanilla network traffic, modbus devices traffic, raw data captures, well documented datasets as well as its wide usage by state-of-the-art systems. However, designed FL-IDS should behave in a generic way and equally work with similar datasets of Table 3.1. That is why, the main objective of Section 4.1 will not be to excessively focus on the principal components extraction or descriptive analysis of the selected dataset. Instead, a more general view of it will be given by performing generic outlier detection and shap values exploration.

Finally, the simulation of a distributed system adds an extra layer of complexity to the main duty of intrusion detection. Therefore, before creating a FL system, its analogous CNL implementation is carried out. Nevertheless, the selected TON_IOT dataset requires a minimum exploration due to its relevance in both implementations.

4.1 Dataset

In this specific task of monitoring network traffic, the Network dataset subset of TON_IOT (UNSW-IoT20) is chosen – A. Alsaedi et al. [13] give a detailed description of it. The TON_IOT contains both categorical and numerical data, making the process of learning more challenging. In spite of not being used, it contains pcap raw network captures, as shown in Table 3.1, to fully customize input data. A fully detailed description of the features is provided in the *description_stats_Network_dataset*[1] – available in Appendix 8. Specifically, all the features are taken into account except the timestamp (ts) which is discarded, alongside class type (string) – label 0 or 1 (normal or attack record) is the only used class variable. The main reason to discard the timestamp is because no time trace will be used in the following development; model architecture is not recurrent nor works with time series. Equally, a binary classification problem to detect the nature of the datagrams is enough to this initial system – making the problem multiclass is left as future work.

[1]https://tiny.cc/ton_iot

4.1.1 Analysis of the dataset

Some data labeled as numerical should be treated as categorical, as a consequence of not having any magnitude relationship: src_port, dst_port, dns_qclass, dns_qtype, dns_rcode, http_trans_depth, http_status_code, http_user_agent – containing network information. On the other hand, one hot encoding is not a viable option owing to the high amount of different possible combinations that could make the training infeasible. In order to have a good understanding of how the true numerical features can influence the binary outcome, a shap values [85] analysis has been performed, as shown in Figure 4.1.

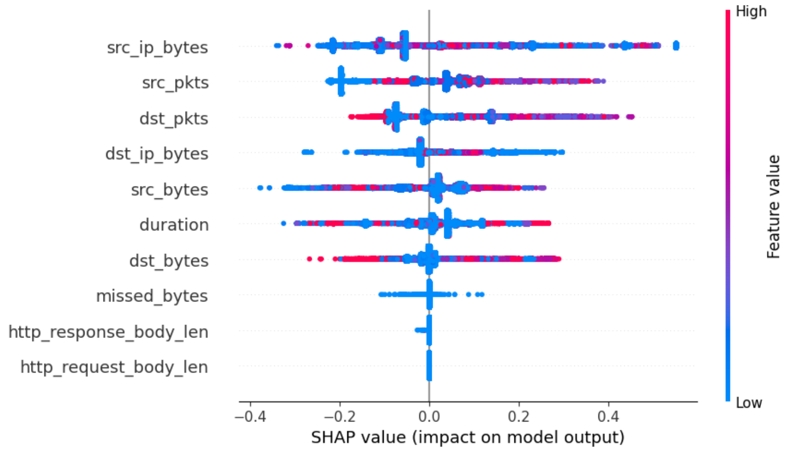


Figure 4.1: Shap values analysis to explore numerical features influence in the class.

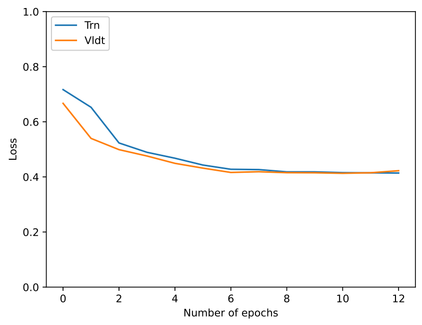
However, the results do not show a particular relationship between the numerical values and the class. As not enough information is extracted, neither weight addition is performed nor numerical data is removed – in the possibility of existence of hidden relationships with other variables.

Outlier detection could be interesting, a priori, in scenarios where reducing the amount of anomalies is recommended (e.g., to train an AE). That can be applied in both CNL and FL cases, with the whole dataset or federated subsets of it, respectively. Nevertheless, reducing the amount of outliers does not necessarily imply a decrease in the anomalous class. In other words, when the number of normal instances is high among the outliers, ignoring them could cause a higher false positive rate.

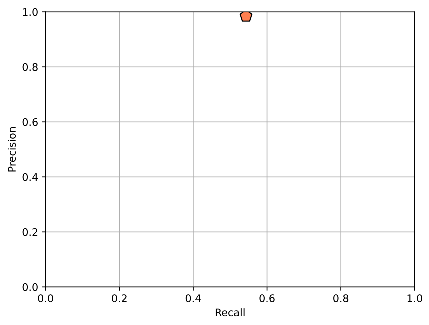
Outlier detection has been performed over the numerical features of the entire dataset, to evaluate possible correlations with the anomalous class. Before beginning the detection, the number of anomalous captures was measured at 34.93% of the total. On the one hand, Isolation Forest (IF) algorithm [86] is applied – 7.38% of the total instances are marked as outliers, where a 42.58% of them are true anomalous instances. On the other hand, One Class Classification SVM (OCC) algorithm [87] is applied – 72.09% of the total instances are marked as outliers, where 28.84% of them are true anomalous instances. The results show a poor correlation in the outlierness of the malicious class. Moreover, the disparity of the algorithms in the percentage of the instances detected as outlier is very notorious – the results of IF are better and more coherent but insufficient as well. Consequently, the outlier detection approach over the raw numerical dataset features is discarded.

4.2 A distance based approach

Working with mixed datasets implies complex data relationships. That is reaffirmed after a few warm up rounds of training a CNL toy classifier[2], where no significant learning is observed – no loss reduction and overall poor performance scores are achieved. Figure 4.2 shows how a toy classifier working with all the dataset instances[3] does not learn patterns in data – no significant loss reduction is observed and recall around 0.5. Therefore, with the aim of addressing the problem from a different perspective, the idea of calculating the Gower Distance among TON_IOT instances emerged.



(a) Train and validation loss.



(b) Position in the PR space.

Figure 4.2: Results of the **toy** classifier working with exclusively numerical features of TON_IOT dataset.

As mentioned in Section 4.1, TON_IOT does not only count with numerical features. Transforming categorical data into numerical is not feasible via techniques such as one-hot encoding, due to the number of features increasing excessively. Therefore, working with numerical and categorical data paves the way to the usage of Gower Distance. Using it, we believe that future models will learn data relationships easier. Thus, Gower Dissimilarity (GD) is computed in the following way – being GD the Gower Dissimilarity between two observations i and j.

$$GD_{ij} = \frac{1}{n}\sum_{f=1}^{n} pd_{ij}^{(f)} \tag{4.1}$$

Having each observation n different features, either numerical, categorical or mixed. For categorical features, the partial dissimilarity (pd) will be 0 if there is a match between the explored couple of features; 1 if there is not. For numerical features, pd is computed by the following partial expressions – the absolute of the subtraction between the specific numerical features divided by the total range of the feature.

$$pd_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{R_f} \tag{4.2}$$

$$R_f = maxf - minf \tag{4.3}$$

[2]Uses the same NN hyperparameters of CNL vanilla version.

[3]Only numerical features due to infeasibility reasons mentioned in Section 4.1

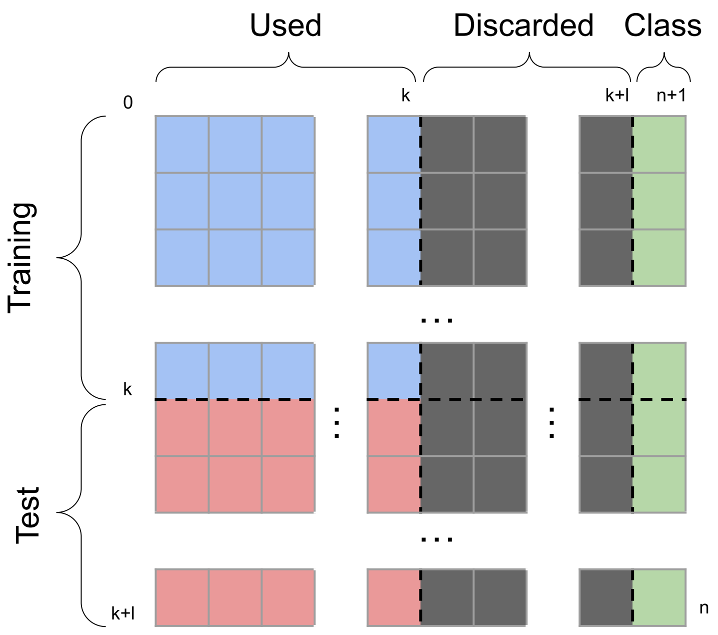


Figure 4.3: Segmentation of generated Gower matrix/matrices to be used in posterior training stages.

After random shuffling the dataset, Gower Distance among instances is computed and sliced as Figure 4.3 shows. In both **Gower Centralized (GC)** and **Gower Federated**[4] **(GF)** systems (Section 4.3), the first k rows and columns (blue) will be used to train the classifier – last l columns (gray) are discarded for containing test information – $k + l = n$. Similarly, the last l instances (red) will be part of the test subset. If an extra validation subset wants to be added, the test partition may be splitted in an additional subset. In the same thread, the computational cost of calculating the initial matrix is $O(n^2)$. However, in reality, once the training matrix is achieved, it must not be recalculated. It is contemplated that new instances will gradually arrive (data streaming), while network monitoring is performed – adding them to the matrix will have a computation cost of $O(n)$.

Nonetheless, to simulate federated devices, the original dataset is divided into n_i sized disjoint subgroups of instances – an independent Gower Matrix will be computed in each device. Then, $k_i * K$ training rows and $l_i * K$ test instances will be used in the learning process – being delimited by the agent which has the minimum training subset of size K among all.

4.3 Designed systems

For this point on, every designed variant will use the previously mentioned Gower Distance matrix (or matrices) as input. The designed systems are: (1) a vanilla GC version; (2) an AE GC version; (3) a vanilla GF version; (4) an AE GF version; (5) a GF version with an AM.

[4]GöwFed

Centralized A baseline implementation is performed using TensorFlow Keras to measure how fast the model is learned. Model architecture[5] is composed by 6 hidden layers of 128, 64, 64, 32, 32, 2 neurons, respectively, with a dropout rate of 0.15 between hidden layers 4 and 5. The input training data consists of a single Gower Distance matrix as described in Section 4.2. System configuration is loaded externally by initialization files, following the next structure.

- **Run name:** Name of the current experiment.
- **Training dataset size:** Number of training instances.
- **Test dataset size:** Number of test instances.
- **Balance dataset:** In the GC matrix creation module; balance data to have 50% of normal and 50% of anomalous instances. The new total number of instances will be the double of the class with less appearances.
- **Epochs:** Number of training epochs.
- **Learning rate:** Hyperparameter to specify model learning speed.
- **Batch size:** Hyperparameter.
- **Seed:** Added for replicability.

An AE version is coded as well, where the main device uses only normal instances of the training subset to train a NN[6] composed by 9 hidden layers of 128, 64, 32, 16, 8, 16, 32, 64, 128 neurons, respectively, with a dropout rate of 0.1 between hidden layers 4 and 5. Then, the average reconstruction Mean Squared Error (MSE) of the normal instances plus their standard deviation is used to compute a discriminating threshold. As the classifier is only trained with normal data, when an attack arrives, its reconstruction error is expected to be higher than the computed normal threshold; and thus, labeled as anomalous.

Federated Learning Previous NN is reimplemented using TFF [55], preserving model architecture and hyperparameters of the GC version. Equally, system configuration is loaded externally, following the next structure.

- **Run name:** Name of the current experiment.
- **Node number:** Total number of agents in the network.
- **Training dataset size:** Total number of training instances – summation of all agents' training datasets.
- **Test dataset size:** Total number of test instances – summation of all agents test datasets.

[5]Fully connected MLPs with a k sized input layer and a 1 sized output layer (*sigmoid*). Using *Adam Optimizer*, *Binary Accuracy* and *Binary Cross-entropy* as training hyperparameters

[6]AE NN architecture with a k sized input layer and a k sized output layer (*sigmoid*). Using *Adam Optimizer* and *Mean Squared Error* as training hyperparameters

- **Balance dataset:** In the GF matrices creation module; balance data to have 50% of normal and 50% of anomalous instances. The new total number of instances will be the double of the class with less appearances.
- **Total rounds:** Total number of communication rounds – averaging rounds.
- **Nodes per round:** Number of agents taking part in each averaging round.
- **Local epochs per round:** Number of training epochs in each device between averaging rounds.
- **Server learning rate:** Hyperparameter to specify global model learning speed.
- **Client learning rate:** Hyperparameter to specify local models learning speeds.
- **Training batch size:** Same hyperparameter for all local models.
- **Test batch size:** Same hyperparameter for all local models.
- **Seed:** Added for replicability.

An AE version is coded as well, where each device only uses the normal instances of the training subset to train a NN, that preserves NN architecture and hyperparameters of the GC AE version.

Finally, an Attention Mechanism (AM) [5] approach is implemented, using the same NN architecture of the vanilla version. In this system, only a pre-established percentage of the agents will contribute to the global model. P agents with the greatest ROC-AUC values will be selected in each round. Other criteria and their combinations could be used as well, such as picking the nodes with greater F1 score, accuracy and so on. Although this mechanism could show generalization issues in scenarios where a small percentage of agents is selected, it can also be interesting to reduce noise and achieve model convergence faster. In other words, selecting a suit percentage of agents can make the learning process easier and improve overall performance. Additionally, this approach is more robust against model poisoning attacks and divergence caused by highly non-IID split information among nodes.

Nonetheless, different variants of AM are explored in the state of the art as well. Weigner et al. [75] proposed the usage of data augmentation to reach a pre-established number of local instances and boost convergence rates. Similarly, agents not reaching a minimum threshold of instances will not be allowed to contribute to the averaged model and will only receive the updated parameters. FedAGRU [69] presents a similar AM to sort and limit client contributions by their importance, in bandwidth scarcity scenarios. This mechanism paved the way to advanced implementations of IDS, where agents contributing negatively to the model could be banned from future apportions [5].

5 Implementation details

As mentioned in Chapter 2, creating a well structured code repository[1] is one of the main priorities of this project. Relevant aspects of the implementation are wrapped-up in the following lines.

5.1 Matrix elaboration

Two modules have been implemented to feed the inputs of GC and GF versions; respectively *create_matrix_cnl* and *create_matrices_fl*. Additionally, those modules are in charge of balancing the datasets (if required) and distributing the instances among the different FL agents. Finally, separated files containing training and test Gower Matrix partitions will be saved – in the case of the GF module, training and test matrices files will be created per agent.

Gower library is used adding custom methods to elaborate the train/test partitions of Figure 4.3; *gower_matrix_limit_cols* and *sliced_gower_matrix_limit_cols*. In the GC version, existing *gower_matrix* function is called, passing as argument just the training instances, to get the $k \times k$ training matrix – cost $O(k^2)$. Then, *sliced_gower_matrix_limit_cols* is called passing the whole dataset, the number of training instances (to be skipped) and the column number limitation to get the test partition – cost $O((k + l) * k)$.

On the other hand, in the GF version, the whole dataset is uniformly distributed in small partitions corresponding to each client – the length of the partition with less number of training instances is stored for future use. As a consequence, an IID distribution of the dataset is simulated among the clients – they are expected to have similar local dataset sizes. Regarding gower library, *gower_matrix_limit_cols* method is called to generate the training matrices – limited by the previously stored minimum number of training instances. Therefore, if no data augmentation is performed, the agent with less training instances will limit future models' input sizes and define k. Finally, *sliced_gower_matrix_limit_cols* method is called to generate the independent test partitions.

5.2 Federated Learning

Firstly, the datasets (Gower Matrices) have to be adapted to a *tff.simulation. datasets.ClientData* federated object type in order to be suitable for **TensorFlow Federated** simulation. The

[1] https://github.com/AitorB16/GowFed

federated dataset is represented as a list of client ids, and a function to look up the local dataset for each client id. Although TFF contains its precompiled testing datasets (EMNIST, CIFAR…), the *tff.simulation.datasets.TestClientData* class allows the easy creation of custom toy datasets for simulation proposes. However a series of constraints have to be met before instantiating the class: (1) load the training and test datasets of each client and transform them into independent dictionaries, where the keys correspond to the features and the class of each dataframe; (2) create two global dictionaries, (one for training and the other for test) where the keys correspond to clients IDs and the values wrap up the dictionaries of step 1. Once the criteria is met, a couple of *TestClientData* instances (training and test) are created by passing the global dictionaries as arguments – in independent calls.

The backbone implementation of all developed FL algorithms is based on the **SimpleFederatedAveraging** guideline provided by the TFF team – all coded custom variants follow the same scheme. A TFF federated algorithm is typically represented as a *tff.templates.IterativeProcess*. This is a class that contains initialize and next functions. Initialize is used to instantiate the server, and next will perform one communication round of the federated algorithm [2] – in an iterative way. The four main components composing the federated algorithms are described in Figure 5.1:

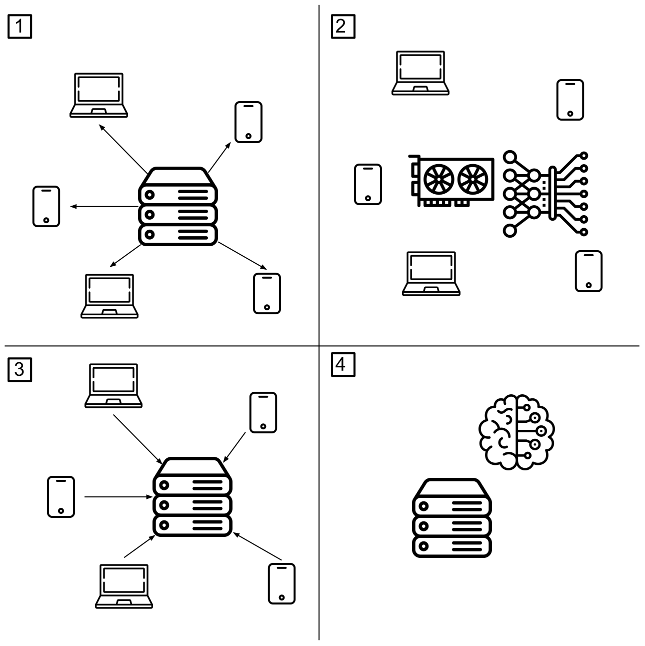


Figure 5.1: Diagram of the four main components of federated algorithms.

1. **A server-to-client broadcast step:** The server weights are broadcasted to the clients taking part in the communication.
2. **A local client update step:** The local gradient is computed on batches of data and then aggregated within the received server weights.
3. **A client-to-server upload step:** The computed local weights are uploaded to the aggregator server.

[2] https://tensorflow.google.cn/federated/tutorials/building_your_own_federated_learning_algorithm

4. **A server update step:** The server model weights are replaced by the average of clients' model weights (FedAVG); where the importance of each client during the averaging process is proportional to its number of local instances – intrinsic characteristic of developments based on *SimpleFederatedAveraging*.

In order to manage and customize the orchestration logic of what the server broadcasts to the client and what the client updates to the server, the Federated Core (FC) API is used. This API has three relevant elements to be mentioned: (1) Federated data type; a data structure hosted across the clients, where the federated type and the placement are defined (e.g., *float32@CLIENTS* meaning that each client has a *float32* type object). (2) *tff.federated_computation*; a specification in an internal platform-independent glue language [88] – functions with well-defined type signatures that can only contain federated operators. (3) *tff.tf_computation*; blocks containing TF code without specifying that can be mapped into federated computations via *tff.federated_map* method.

Previously mentioned statement about maintaining the same FL skeleton is partially true, but there are small variations in the AM version. In that case, the next step of the iterative process is splitted in two rounds. The first one computes the local models of the selected subset of clients and sends the partial results as well as the computed local weights to the server. Then, the server picks the k best performing nodes according to the best ROC-AUC areas and discards the rest $n - k$ agents – the k number of nodes varies according to the selection percentage specified in the initialization file. Finally, in the second round, the selected weights are uploaded to the server and **FedAVG** is performed; modulating the influence on the global model by the number of instances (as done in the vanilla approach).

6 Experimental setup

A series of experiments have been carried out to test GöwFed's (GF) performance and scalability. In order to do so, outputs from a series of analogous CNL versions are explored as well (GC) – contrasting their performance against the FL systems. The experimentation has been performed in a machine with the following specs: (1) CPU - Intel i9-7920X 4.3Ghz. (2) RAM - 64GB DDR4 2400MT/s. (3) GPUs - 2x Nvidia RTX 2080 Ti 11GB.

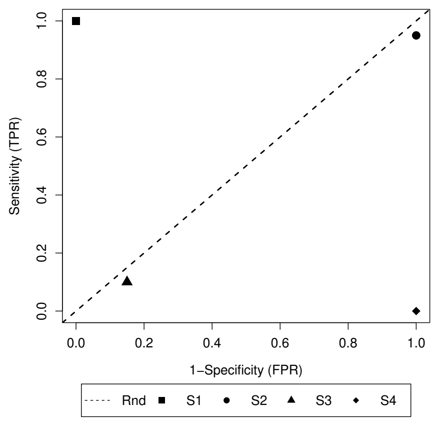
6.1 Metrics of interest

State-of-the-art systems use particular scoring metrics, lacking from unification. Accuracy is the most common metric, followed by detection rate and F1-score. At any rate, this variety of metrics makes comparing solutions a complex and non-intuitive process. For this reason, we strongly believe that standardizing the evaluation process under a reliable metric is imperative. Assuming the intrusions as the positive class, the use of detection rate is not a fair practice due to a possible high amount of false positives. Thus, the best-practice is for it to be accompanied by the false positive rate (FPR).

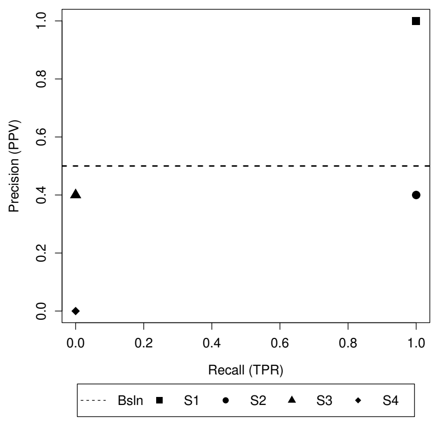
Among the metrics in the literature, F1-score is the most complete one due to its ability to wrap up precision and recall. However, as true negatives are not taken into account, F1- score could be problematic in asymmetric scenarios where the negative class is the minority (i.e., in a critical scenario where attacks are more common than normal traffic).

Although its utilization is not so popular, kappa statistic [89] is another interesting metric that quantifies the behavior of a predictive model in contrast to a random chance detector [90]. It works similarly to a correlation coefficient, rating model performance between $[-1, 1]$. A value of 1 represents a complete agreement, 0 means no agreement or independence and, finally, a negative value implies that the predictive model is worse than random [91]. The adoption of the kappa statistic over the commonly used metrics is highly recommended due to its reliability and interpretation simplicity.

At any rate, we advocate the use of 2D graphical metrics contrasting standard statistical metrics such as positive predictive value (PPV), true positive rate (TPR) or false positive rate (FPR). Combining the ROC space (*TPR vs FPR*) [92] and the PR space (*PPV vs TPR*) [93] with their corresponding curves and areas under the curves (AUCs) is a solid option to obtain a richer evaluation of most FL-ID models when compared with the metrics above. The graphical representation of those 2D metrics delivers complementary information. The ROC curve gives equal importance to positive and negative classes, whereas the PR curve is more informative in skewed scenarios, focusing on the positive class by penalizing false



(a) ROC space (TPR vs FPR).



(b) PR space (PPV vs TPR).

Figure 6.1: Example of the 2D graphical representation of the predictive performance metrics with the four illustrative scenarios.

negatives considerably. Albeit to a lesser extent, the ROC curve penalizes a naïve model behavior in unbalanced problems, when the majority of positive samples are predicted as negative. Therefore, it is equally capable of penalizing poor performance of predictive models in the minority (positive) class.

Figure 6.1 shows how proposed graphical metrics are expected to behave in some illustrative scenarios, as follows:

- **S1:** Absence of false negatives and absence of false positives (ideal case).
- **S2:** Absence of false negatives and abundance of false positives.
- **S3:** Abundance of false negatives and absence of false positives.
- **S4:** Only false negatives and only false positives (worst case).
- **Dashed line (TPR vs FPR):** Random guess.
- **Dashed line (PPV vs TPR):** Variable baseline.

The penalization difference exposed in previous paragraphs is illustrated in Scenario S3, as the PR curve shows a greater distance from the default baseline than the ROC curve from the random guess. That is especially noticeable when the positive is the extremely minority class. However, relying exclusively on the PR curve is not recommended due to its asymmetry – not considering true negatives (Scenario S2). Therefore, we believe that combining both graphical representations should be the best-practice to make richer interpretations of the results, instead of other more commonly used metrics.

Nonetheless, working with FLS entails the consideration of additional performance indicators. Measuring the variability of required communication rounds and elapsed time to achieve model convergence (i.e., until the learning model reaches a predefined quality threshold under a reliable metric), subject to a changeable number of parties

Table 6.1: Configuration parameters used in CNL experiments.

Run name [123]	Training ds size	Test ds size	Dataset balanced	Epochs	Learning rate	Batch size	Seed
GC_SB	10000	2000	✓	100	0.0001	64	26
GC_SU	10000	2000	×	100	0.0001	64	26
GC_MB	20000	4000	✓	100	0.0001	64	27
GC_MU	20000	4000	×	100	0.0001	64	27
GC_LB	40000	8000	✓	100	0.0001	64	28
GC_LU	40000	8000	×	100	0.0001	64	28

[1] : GC: Gower Centralized Learning. [2] S: Small; M: Medium; L: Large.
[3] B: Balanced; U: Unbalanced.

Table 6.2: Configuration parameters used in FL experiments.

Run name [123]	Node numbr	Training ds size	Test ds size	Dataset blnced	Total rounds	Nodes /round	Local epochs	Srvr lrng rate	Clnt lrng rate	Training batch	Test batch	Seed
GF_SB	10	30000	5000	✓	100	4	10	0.0001	0.00001	128	32	26
GF_SU	10	30000	5000	×	100	4	10	0.0001	0.00001	128	32	26
GF_MB	20	60000	10000	✓	100	8	10	0.0001	0.00001	128	32	27
GF_MU	20	60000	10000	×	100	8	10	0.0001	0.00001	128	32	27
GF_LB	40	120000	20000	✓	100	16	10	0.0001	0.00001	128	32	28
GF_LU	40	120000	20000	×	100	16	10	0.0001	0.00001	128	32	28

[1] GF: Gower Federated Learning – GöwFed. [2] S: Small; M: Medium; L: Large. [3] B: Balanced; U: Unbalanced.

(scalability), is important. In the same way, measuring the time required to perform a complete communication round in different bandwidth scarcity scenarios is interesting to evaluate the robustness and resilience of the system.

6.2 Gower Centralized

As mentioned previously, this is a baseline to compare ongoing GF approaches. Table 6.1 summarizes the different run configurations. In the same way, after each experiment, training and validation losses are stored as well as a copy of the running configuration, the learned h5 model and a series of overall metrics: accuracy, precision, recall, F1 score and ROC-AUC. Moreover, following the best-practice metrics proposed in Section 6.1, a combination of PR space and accuracy plots are used to display the results.

6.3 Gower Federated

The same procedure is followed to test GF performance and scalability. Working with Gower Dissimilarity matrices requires a high amount of system memory. That is why, a pseudo-fixed amount of samples is used at each client – obtained as a consequence of uniformly distributing the dataset as mentioned in Section 4.3. The local dataset sizes per experiment will be the same of Figure 6.2 in every implemented version. Moreover, the total number of samples and the number of agents taking part in each averaging round, will be proportional to the total number of agents. In other words, as the data subset size in each agent will be similar, when the number of agents increases, the summation of all their instances will be higher. Specifically, six configurations, with different seeds, have been tested to measure the scalability of the system. Table 6.2 summarizes the mentioned configurations.

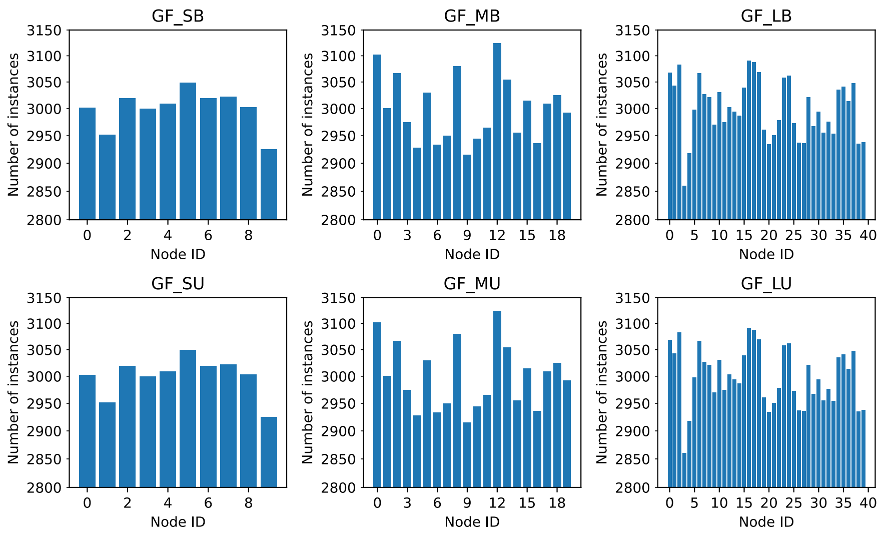


Figure 6.2: Training dataset partition sizes per agent ID in every GF version.

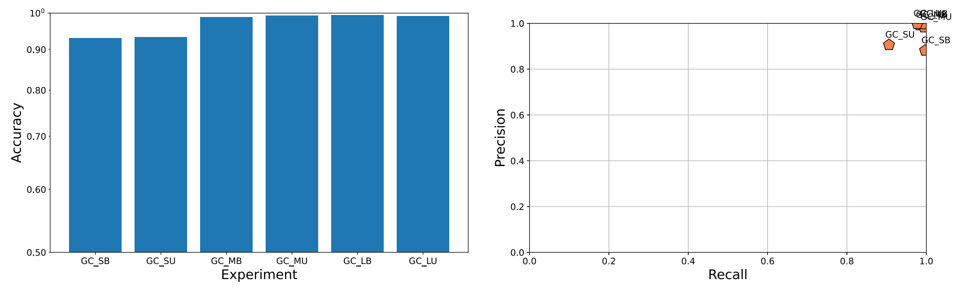
Similarly, per experiment, a series of attributes and metrics are stored containing each nodes' information: agent id, agent training dataset size, accuracy, precision, recall, F1 score and ROC-AUC area. Those metrics are obtained by evaluating the averaged global model against the test subset matrix of each agent – each agent is expected to converge to the same global model after the *SimpleFederatedAveraging* emulation. The experimentation is performed over the three designed systems mentioned in Section 4.3 – AM version is run twice with different attention percentages. Overall configuration parameters are the same for each system – specified in Table 6.2.

7 Results

After running the simulations, a deep analysis of the results have to be made. On the one hand, the vanilla and AE versions of the centralized system are explored. On the other hand, the results obtained in the vanilla, AE and two AM versions of the federated system are discussed as well as contrasted to their analogous centralized approach.

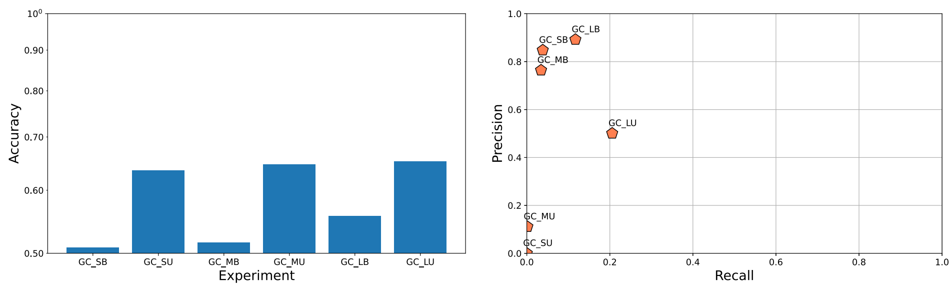
7.1 Gower Centralized

As Figures 7.1a and 7.1b show, the results obtained by the GC vanilla version are nearly perfect. PR space shows models trained with more than 40000 instances near the point $[1, 1]$ as well as an accuracy of around 0.99 for those same experiments. Nevertheless, the AE version seems to have a really poor performance in experiments SB, MB and LB with low accuracies – corresponding to the ones with balanced datasets. That could happen because the AE is being trained with less (normal) instances; learning quality is lost because the dataset size remains unaltered but the anomalous and normal instances percentage is the same – described in Section 4.1. Similarly, recall values are nearly 0, whereas precision is close to 0.9 in experiments GC_SB, GC_MB and GC_LB – close to 0.1 in experiments GC_SU and GC_MU. Therefore, a very small percentage of the anomalous instances is being detected (recall near 0), even so among the detected ones, the probability of being a false positive is low (precision around 0.9) – for the non balanced experiments.



(a) Accuracy. **(b)** Position in the PR space.

Figure 7.1: Results of learned **vanilla GC** models in the test partition of each experiment.



(a) Accuracy. **(b)** Position in the PR space.

Figure 7.2: Results of learned **GC AE** models in the test partition of each experiment.

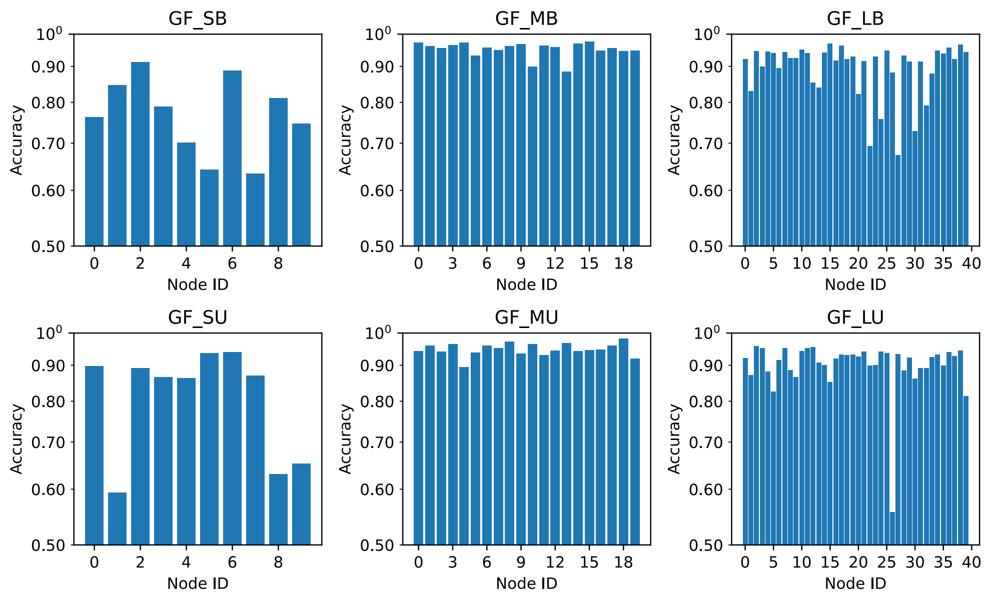
7.2 Gower Federated

Figure 7.3 shows learned global model performance per agent and experiment in the GF vanilla version – each bar or point corresponds to a specific agent ID. PR space shows an overall good performance of the global model in each nodes' test partition. Those results are seconded by the accuracies, being around 0.9 and consistent in the experiments with more than 20 agents. Nevertheless, the AE version seems to have performance issues as its GC analogous – achieving worse results in experiments with balanced datasets; Figure 7.4. However, in this case, precision is around 0.5 and recall is close to 0.9 in all the experiments. Therefore, all the anomalous instances are being detected (recall near 0.9), whereas the false positive rate is very high (precision around 0.5).

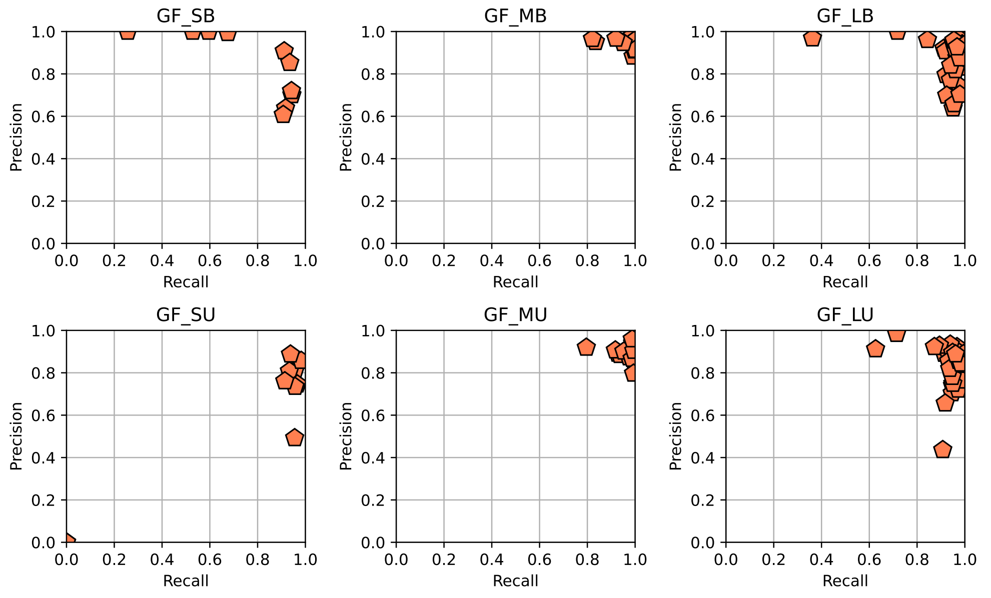
Moreover, the AM development with 0.2 of best performing agents, have very dissimilar results. As it is expected, some agents never contribute to the averaged model and 0.2 of them are not enough to learn all the threat patterns – the learned model does not generalize well and bad results are achieved. In other words, the gap between the well and bad performing agents is accentuated alongside the total number of agents increases – Figure 7.5 shows the mentioned disparity. However, in the 0.8 best performing nodes case, the results are overall comparable to the vanilla GF version – with an acceptable performance of the global model in the majority of the nodes; Figure 7.6. Despite not being the current scenario, this mechanism could be interesting to ensure convergence in cases where negatively contributing agents are present. Nevertheless, AM causes decrement of performance in scenarios where data is IID distributed among nodes – as the current one.

7.3 Gower Centralized vs Gower Federated

As it could be expected, GC slightly outperforms GF in the vanilla version experiments. However, the comparison is not completely fair due to the number of training rounds that FL versions could require to achieve the same convergence levels than CNL versions. As it is described in Tables 6.1 and 6.2, the GC models are trained with 100 epochs and the GF models with 100 communication rounds – 10 local epochs per round are performed in each round. Furthermore, an early stopping criteria is used in the GC systems, with 2 rounds of patience, that causes each experiment to have a variable number of training rounds. GF systems are forced to stop at 100 communication rounds, as a timeout, because they still continue learning – keep in mind that epochs in FL systems are performed independently by each agent. Figure 7.7a shows that training and validation losses in each



(a) Accuracy per node ID.

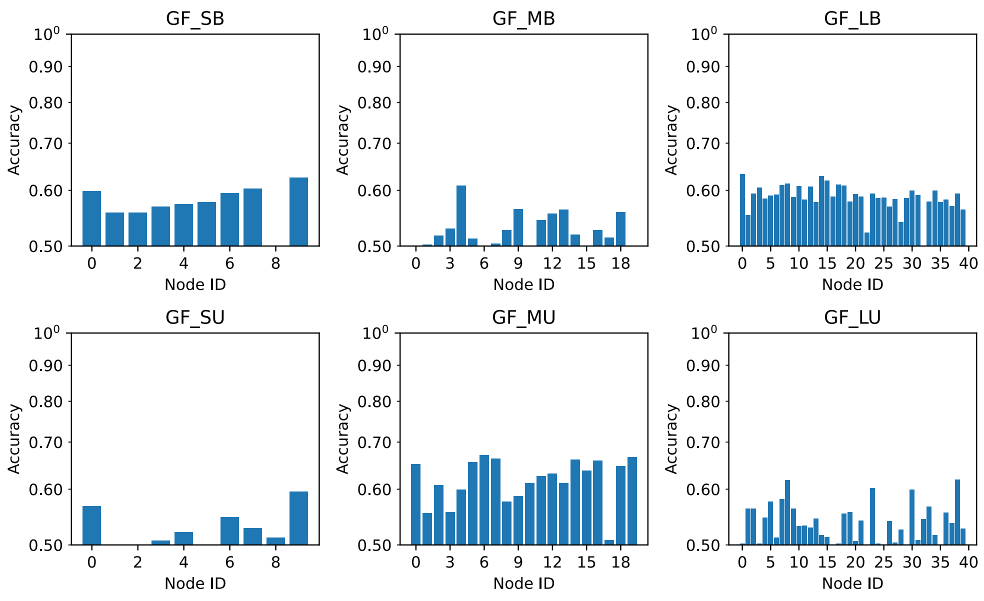


(b) Position in the PR space.

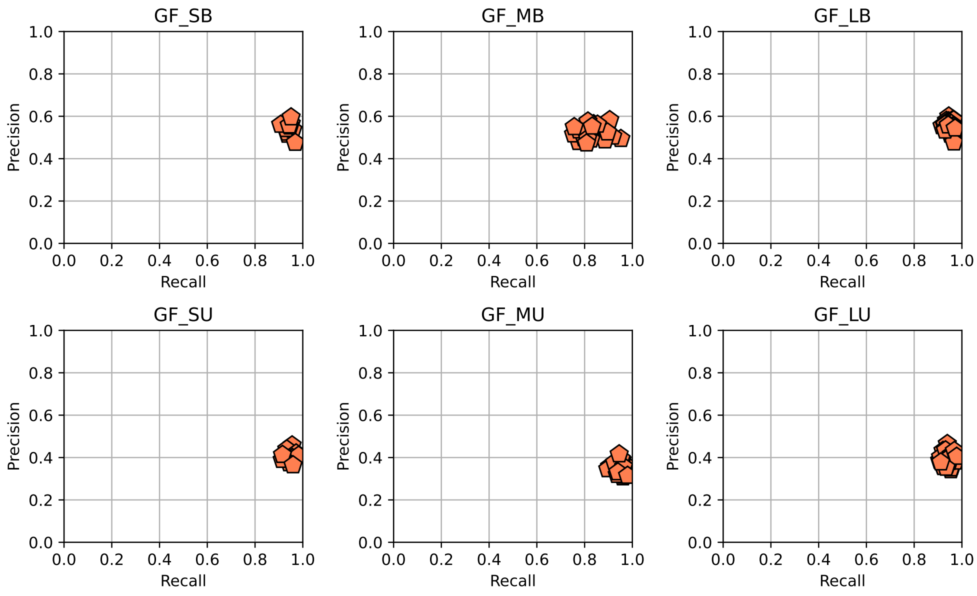
Figure 7.3: Results of learned **vanilla GF** models in the test partition of each agent per experiment.

vanilla GC experiment end up converging into the same values. Nonetheless, training and validation losses of vanilla GF experiments do not converge equally in the explored number of rounds – Figure 7.7b. Hence, more communication rounds might be needed to reach similar convergence rates than those of GC models.

Moreover, the comparison is not strictly fair due to scalability reasons. In the used machine (Chapter 6), working with CNL Gower Matrices of more than 40000 instances is computationally infeasible. However, the summation of all agents' Gower training instances



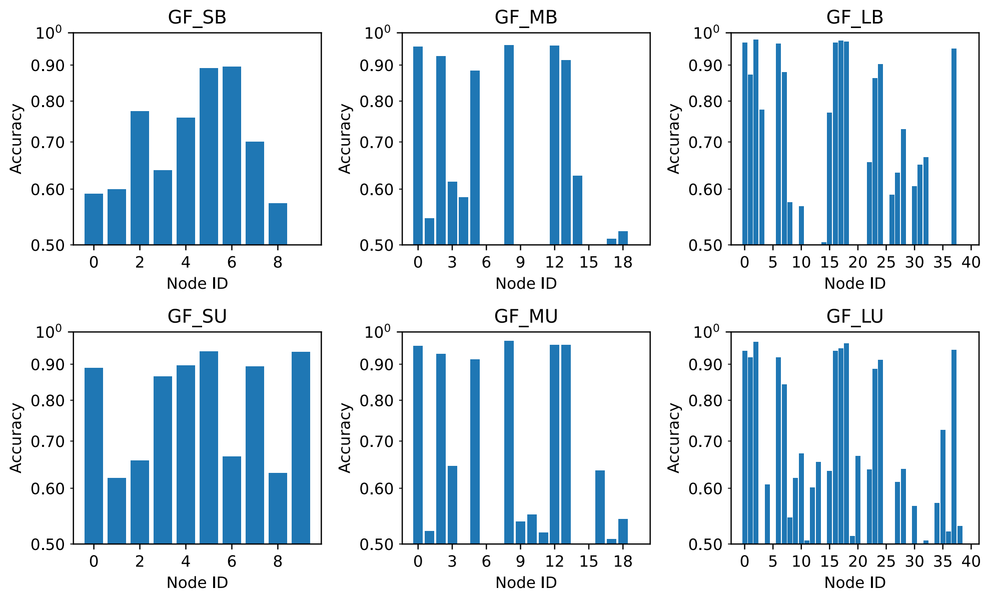
(a) Accuracy per node ID.



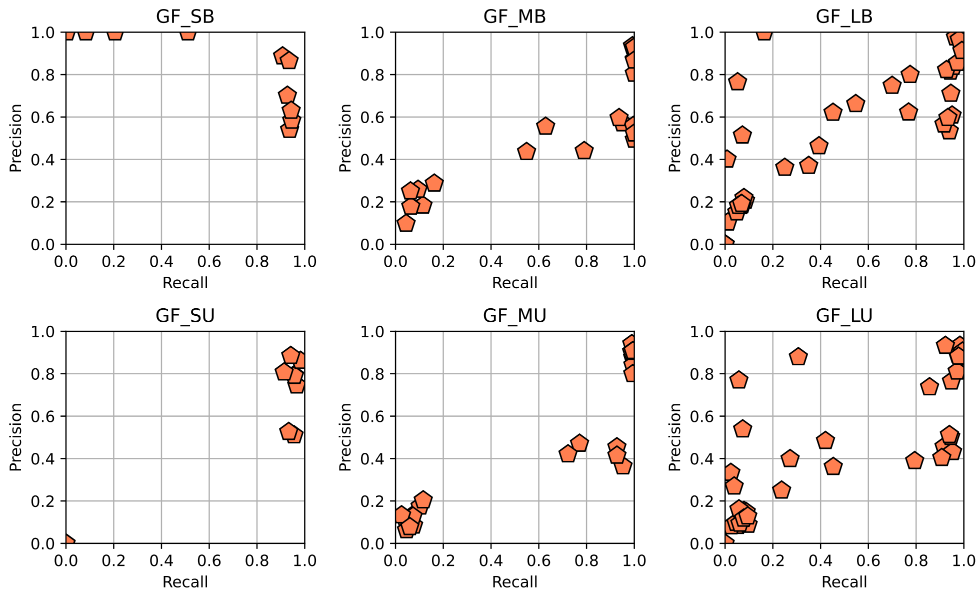
(b) Position in the PR space.

Figure 7.4: Results of learned **GF AE** models in the test partition of each agent per experiment.

in e.g., experiment GF_LU, is 120000; meaning that using Gower Distance approach in distributed systems is more scalable – total instance limit has not been reached in the performed experiments. The previous happens because the CNL training matrix will be squared, whereas the federated ones will have a variable size – Section 4.2. Thus, performed experiments do not count with exactly equivalent training/test instances for hardware limitation reasons.

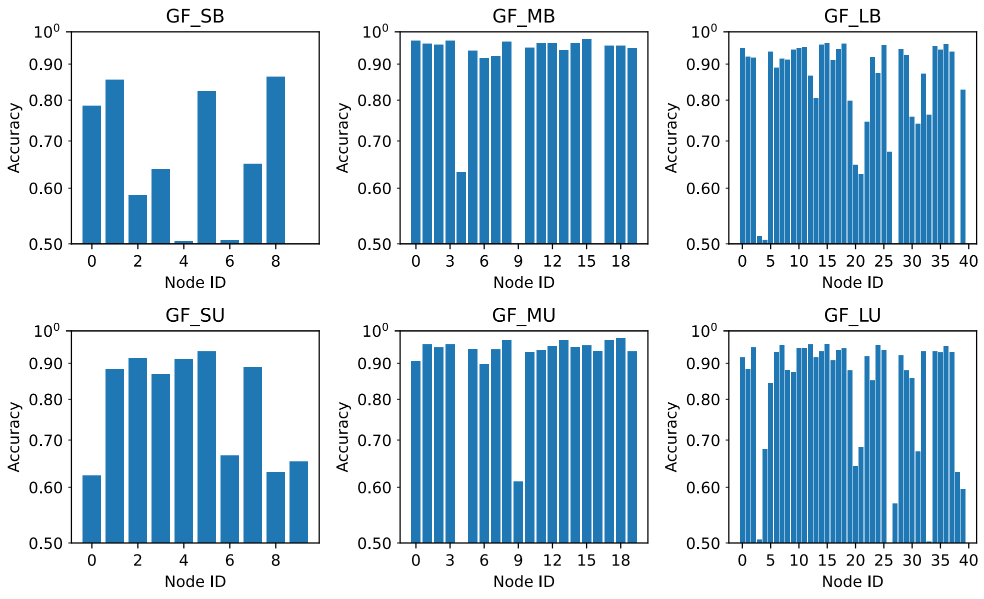


(a) Accuracy per node ID.

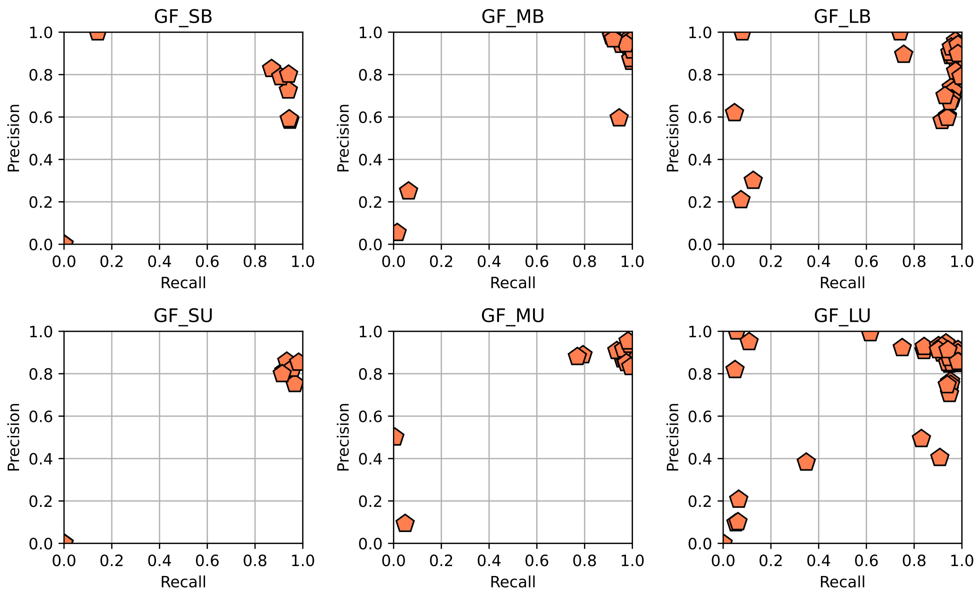


(b) Position in the PR space.

Figure 7.5: Results of learned **GF AM 0.2** models in the test partition of each agent per experiment.

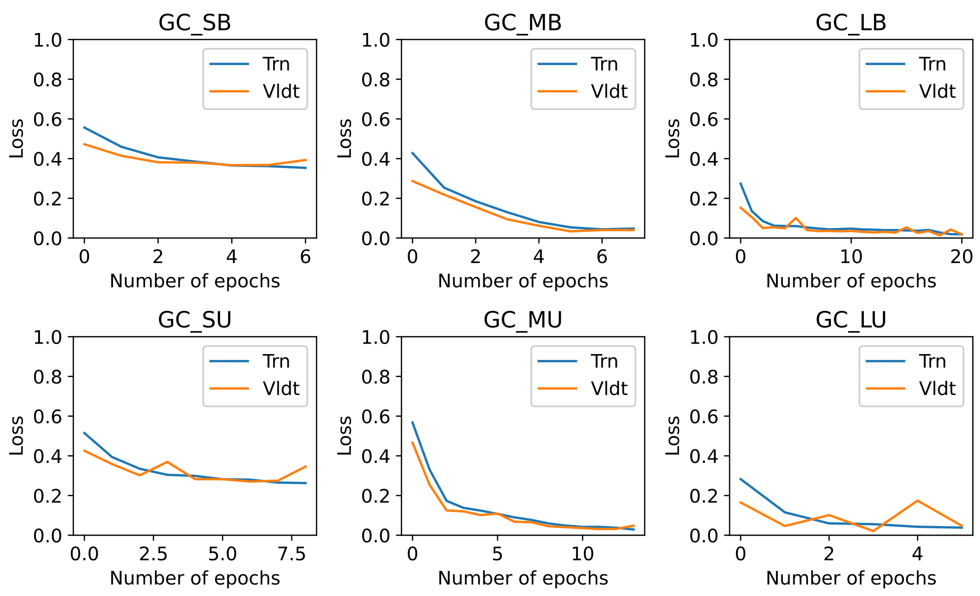


(a) Accuracy per node ID.

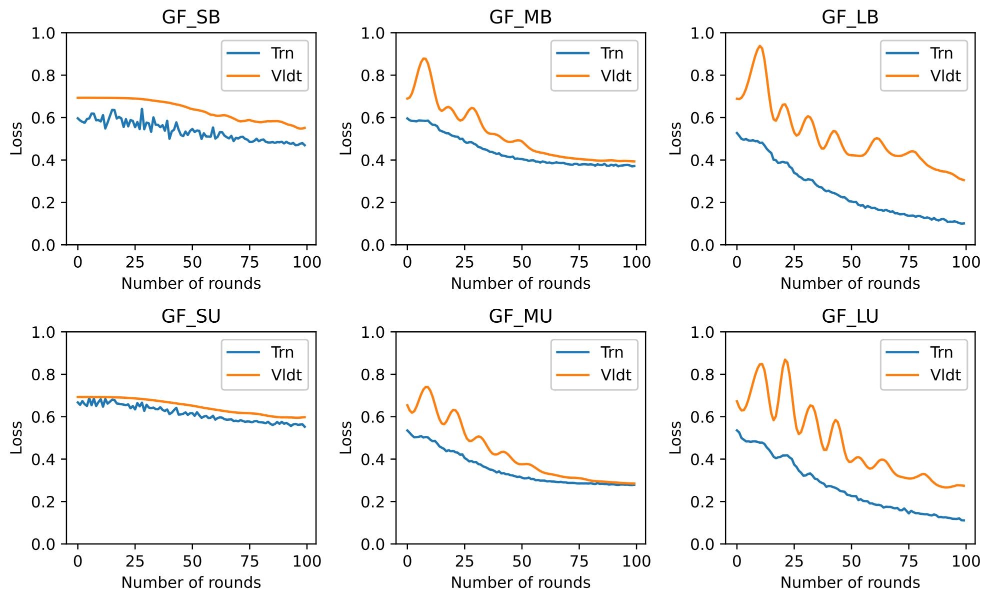


(b) Position in the PR space.

Figure 7.6: Results of learned **GF AM 0.8** models in the test partition of each agent per experiment.



(a) 100 Epochs of training each GC model – with early stopping.



(b) 100 Averaging rounds of each GF global model.

Figure 7.7: Training and validation losses of **vanilla GC and GF** systems respectively in each experiment.

8 Conclusions and future work

The designed system pretends to be an intermediate step before being deployed in real IoT devices. On the one hand, the modularity of the implementation makes the experimentation with different configurations easier. Similarly, development can be conducted incrementally and debugging performed trivially. On the other hand, little adaptations might be necessary to work in real streaming data scenarios, where a single Gower row will be computed instead of the whole matrix – as mentioned in Section 4.2. However, the development is partially adapted with pre-implemented functions 5.1 *gower_matrix_limit_cols* and *sliced_gower_matrix_limit_cols*. At the end of the day, GöwFed is created pursuing the continuity of new experiments using Gower Distance matrix and advanced Deep Learning architectures into a Federated Learning framework.

Regarding the achieved results, GC systems perform slightly better than GF systems in all the explored experiments – except in the AE development. As it was expected, GF systems add an extra layer of complexity that minimally burdens overall performance. However, the comparison is not completely fair due to the mentioned scalability issues that GC versions have. As it is mentioned in Section 7.3, the number of instances used to create GC and GF system matrices are different due to hardware limitations – e.g., counting with a total number of 120000 instances for federated and 40000 for centralized versions. In addition, epochs (CNL) and communication rounds (FL) do not work in a similar way, and therefore, model convergence rates can not be compared in a raw manner. At this point, it can be concluded that the usage of independent IoT devices under the GöwFed approach, makes the system more scalable than its analogous GC development.

The AE implementations of both GC and GF systems are not as promising as the other variants due to the less favorable results achieved – in comparison to other versions. In the GC system, high precisions and low recalls are obtained, whereas in the GF system, low precisions and high recalls are obtained. Consequently, the GC system does not learn how to correctly detect anomalies, whereas the ones detected are certainly classified. On the other hand, the GF system is very sensible detecting anomalies, but it misclassifies a considerable amount of normal traffic – high false positive rate. Furthermore, balancing the datasets performed in a remarkable bad way in all the AE experiments of both systems. That could be explained by a reduction in the amount of normal instances as a consequence of balancing the datasets in favor of the anomalous class – regarding the prevalence of normal instances, over the anomalous, mentioned in Section 4.1. The solution could possibly come from the opposite side, by adding more normal instances via capturing new ones or using data augmentation techniques.

The GF AM version using 0.8 of best performing agents, achieves good results and makes the system more robust against poisoning attacks. Nonetheless, its potential does not shine in the elaborated experiments due to the IID data splitting performed among agents – the results show a general worsening compared to the non-AM version. Additionally, as mentioned in Section 5.2, non-AM versions count with a weight mechanism that gives (during the averaging process) more importance to nodes with a higher number of instances. As a consequence, results of non-AM version are boosted from the beginning. On the other hand, a small percentage of best performing nodes (0.2) is not enough to learn a model that generalizes well – achieving bad results in the majority of the agents. Nevertheless, AM is incorporated to GowFed as part of working with independent model results and parameters; that can not be easily adapted to GC systems – making them more vulnerable. In short, AM is a promising approach that needs to be studied in a deeper way – using other evaluation criteria than ROC-AUC, testing over heterogeneous devices with non-IID data splitting to make it shine and so on.

After summarizing the results, it can be concluded that the usage of Gower Distance matrices to create a FL-IDS is feasible and doable. That is seconded by the comparison between the losses of GöwFed experiments 7.7b and the ones obtained by the toy (non Gower) classifier 4.2; where GöwFed's loss decreases drastically – especially in experiments FL_LB and FL_LU. In the same way, GF versions perform in a similar way to their analogous GC ones, in terms of performance and capabilities. Therefore, research questions posed in Chapter 2 are successfully satisfied – obtained results are quite optimistic.

In the future, GöwFed will have to count with an incremental learning version tested over data streaming scenarios. Because, as mentioned in Chapter 4, current batch learning version simulates an artificial environment working over the TON_IOT dataset. However, modifying the implementation to work with real IoT devices should require just some small adaptations; i.e., making the nodes able to process captured datagrams, update local matrices and so on.

Similarly, the combined usage of the designed system and input data of different nature such as time series has not been covered yet – many state-of-the-art approaches work with temporal relationships and RNN; Section 3.2. In the same way, a fully distributed FL-IDS has not been developed yet, dispensing with the need of a central orchestration server like the one in FedAVG – the path opened by P. Vanhaesebrouck et al. [21] could be followed to accomplish that task. Nevertheless, novel approaches can trigger new convergence challenges to the nodes. In order to face those incoming challenges, the incorporation of information pre-sharing mechanisms could prevent divergence of the matrices (and models) without compromising the privacy of the agents. Tian et al. [74] propose a mechanism to share a small percentage of local instances among nodes, committing the mentioned privacy constraint. Linked to convergence, complementary methods that use data augmentation [83] could be used to accelerate the convergence rates of the models [75].

Appendix

Description of Network Features

Service profile: Connection activity			
ID	**Feature**	**Type**	**Description**
1	ts	Time	Timestamp of connection between flow identifiers
2	src_ip	String	Source IP addresses which originate endpoints' IP addresses
3	src_port	Number	Source ports which Originate endpoint's TCP/UDP ports
4	dst_ip	String	Destination IP addresses which respond to endpoint's IP addresses
5	dst_port	Number	Destination ports which respond to endpoint's TCP/UDP ports
6	proto	String	Transport layer protocols of flow connections
7	service	String	Dynamically detected protocols, such as DNS, HTTP and SSL
8	duration	Number	The time of the packet connections, which is estimated by subtracting 'time of last packet seen' and 'time of first packet seen'
9	src_bytes	Number	Source bytes which are originated from payload bytes of TCP sequence numbers
10	dst_bytes	Number	Destination bytes which are responded payload bytes from TCP sequence numbers
11	conn_state	String	Various connection states, such as S0 (connection without replay), S1 (connection established), and REJ (connection attempt rejected)
12	missed_bytes	Number	Number of missing bytes in content gaps

Service profile: Statistical activity			
ID	**Feature**	**Type**	**Description**
13	src_pkts	Number	Number of original packets which is estimated from source systems
14	src_ip_bytes	Number	Number of original IP bytes which is the total length of IP header field of source systems
15	dst_pkts	Number	Number of destination packets which is estimated from destination systems
16	dst_ip_bytes	Number	Number of destination IP bytes which is the total length of IP header field of destination systems

Service profile: DNS activity			
ID	**Feature**	**Type**	**Description**
17	dns_query	string	Domain name subjects of the DNS queries
18	dns_qclass	Number	Values which specifies the DNS query classes
19	dns_qtype	Number	Value which specifies the DNS query types
20	dns_rcode	Number	Response code values in the DNS responses
21	dns_AA	Bool	Authoritative answers of DNS, where T denotes server is authoritative for query
22	dns_RD	Bool	Recursion desired of DNS, where T denotes request recursive lookup of query
23	dns_RA	Bool	Recursion available of DNS, where T denotes server supports recursive queries
24	dns_rejected	Bool	DNS rejection, where the DNS queries are rejected by the server

Service profile: SSL activity			
ID	**Feature**	**Type**	**Description**
25	ssl_version	String	SSL version which is offered by the server
26	ssl_cipher	String	SSL cipher suite which the server chose
27	ssl_resumed	Bool	SSL flag indicates the session that can be used to initiate new connections, where T refers to the SSL connection is initiated
28	ssl_established	Bool	SSL flag indicates establishing connections between two parties, where T refers to establishing the connection
29	ssl_subject	String	Subject of the X.509 cert offered by the server
30	ssl_issuer	String	Trusted owner/originator of SLL and digital certificate (certificate authority)

Service profile: HTTP activity			
ID	**Feature**	**Type**	**Description**
31	http_trans_depth	Number	Pipelined depth into the HTTP connection
32	http_method	String	HTTP request methods such as GET, POST and HEAD
33	http_uri	String	URIs used in the HTTP request
35	http_version	String	The HTTP versions utilised such as V1.1
36	http_request_body_len	Number	Actual uncompressed content sizes of the data transferred from the HTTP client
37	http_response_body_len	Number	Actual uncompressed content sizes of the data transferred from the HTTP server
38	http_status_code	Number	Status codes returned by the HTTP server
39	http_user_agent	Number	Values of the User-Agent header in the HTTP protocol
40	http_orig_mime_types	String	Ordered vectors of mime types from source system in the HTTP protocol
41	http_resp_mime_types	String	Ordered vectors of mime types from destination system in the HTTP protocol

Service profile: Violation activity			
ID	**Feature**	**Type**	**Description**
42	weird_name	String	Names of anomalies/violations related to protocols that happened
43	weird_addl	String	Additional information is associated to protocol anomalies/violations
44	weird_notice	bool	It indicates if the violation/anomaly was turned into a notice

Service profile: Data labelling			
ID	**Feature**	**Type**	**Description**
45	label	Number	Tag normal and attack records, where 0 indicates normal and 1 indicates attacks
46	type	String	Tag attack categories, such as normal, DoS, DDoS and backdoor attacks, and normal records

Bibliography

[1] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. pages 1–44, 2019. See pages v, 7.

[2] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer Publishing Company, Incorporated, 1st edition, 2017. See page 2.

[3] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, 54, 2017. See pages 2, 7.

[4] J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971. See page 2.

[5] Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M. Shamim Hossain. Deep Anomaly Detection for Time-Series Data in Industrial IoT: A Communication-Efficient On-Device Federated Learning Approach. *IEEE Internet of Things Journal*, 8(8):6348–6358, 2021. See pages 2, 8, and 16.

[6] Beibei Li, Yuhao Wu, Jiarui Song, Rongxing Lu, Tao Li, and Liang Zhao. DeepFed: Federated Deep Learning for Intrusion Detection in Industrial Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*, 17(8):5615–5624, 2021. See pages 3, 8, and 9.

[7] Aitor Belenguer, Javier Navaridas, and Jose A. Pascual. A review of federated learning in intrusion detection systems for iot, 2022. See page 5.

[8] Heikki Topi Carol V. Brown. Is management handbook. *CRC Press*, 1999. See page 5.

[9] Robert E. Heady, George F. Luger, Arthur B. Maccabe, and Mark Servilla. The architecture of a network level intrusion detection system. 1990. See page 5.

[10] Borja Molina-Coronado, Usue Mori, Alexander Mendiburu, and Jose Miguel-Alonso. Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process. *IEEE Transactions on Network and Service Management*, 17(4):2451–2479, 2020. See page 5.

[11] E. Chatzoglou, G. Kambourakis, and C. Kolias. Empirical evaluation of attacks against ieee 802.11 enterprise networks: The awid3 dataset. *IEEE Access*, 9:34188–34205, 2021. See page 6.

[12] & Maria Jose Erquiaga Sebastian Garcia, Agustin Parmisano. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0) [data set]. Zenodo, 2020. See page 6.

[13] Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adna N Anwar. TON-IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020. See pages 6, 11.

[14] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7:479–482, 01 2018. See page 6.

[15] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of tor traffic using time based features. pages 253–262, 01 2017. See page 6.

[16] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of encrypted and vpn traffic using time-related features. 02 2016. See page 6.

[17] Iman Almomani, Bassam Kasasbeh, and Mousa AL-Akhras. Wsn-ds: A dataset for intrusion detection systems in wireless sensor networks. *Journal of Sensors*, 2016:1–16, 01 2016. See page 6.

[18] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Stefanos Gritzalis. Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Communications Surveys Tutorials*, 18(1):184–208, 2016. See page 6.

[19] William DuMouchel, Wen-Hua Ju, Alan F. Karr, Matthias Schonlau, Martin Theusan, and Yehuda Vardi. Computer Intrusion: Detecting Masquerades. *Statistical Science*, 16(1):58 – 74, 2001. See page 6.

[20] Sathyanarayanan Revathi and A. Malathi. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International journal of engineering research and technology*, 2, 2013. See page 6.

[21] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized collaborative learning of personalized models over networks. 10 2016. See pages 6, 34.

[22] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning, 2020. See pages 6, 7.

[23] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. *CoRR*, abs/2003.00295, 2020. See page 6.

[24] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in Neural Information Processing Systems*, 2020-December, 2020. See page 6.

[25] Anit Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks, 12 2018. See page 6.

[26] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *CoRR*, abs/2002.06440, 2020. See page 6.

[27] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *CoRR*, abs/1712.01887, 2017. See page 6.

[28] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. See page 6.

[29] Hanlin Tang, Xiangru Lian, Shuang Qiu, Lei Yuan, Ce Zhang, Tong Zhang, and Ji Liu. Deepsqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *CoRR*, abs/1907.07346, 2019. See page 6.

[30] Constantin Philippenko and Aymeric Dieuleveut. Artemis: tight convergence guarantees for bidirectional compression in federated learning. *CoRR*, abs/2006.14591, 2020. See page 6.

[31] Mohammad Mohammadi Amiri, Deniz Gündüz, Sanjeev R. Kulkarni, and H. Vincent Poor. Federated learning with quantized global model updates. *CoRR*, abs/2006.10672, 2020. See page 6.

[32] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. *CoRR*, abs/2007.01154, 2020. See page 6.

[33] Zhenheng Tang, Shaohuai Shi, and Xiaowen Chu. Communication-efficient decentralized learning with sparsification and adaptive peer selection. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*, pages 1207–1208. IEEE, 2020. See page 6.

[34] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization, 2017. See page 6.

[35] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *CoRR*, abs/1611.04482, 2016. See page 6.

[36] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017. See page 6.

[37] Tribhuvanesh Orekondy, Seong Joon Oh, Bernt Schiele, and Mario Fritz. Understanding and controlling user linkability in decentralized learning. *CoRR*, abs/1805.05838, 2018. See page 6.

[38] Théo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *CoRR*, abs/1811.04017, 2018. See pages 6, 7.

[39] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 691–706. IEEE, 2019. See page 6.

[40] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning - (extended abstract). *Inform. Spektrum*, 42(5):356–357, 2019. See page 6.

[41] Aleksei Triastcyn and Boi Faltings. Federated learning with bayesian differential privacy. In *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 2587–2596. IEEE, 2019. See page 6.

[42] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019*, pages 13–23. ACM, 2019. See page 6.

[43] Aleksei Triastcyn and Boi Faltings. Federated generative privacy. *IEEE Intell. Syst.*, 35(4):50–57, 2020. See page 6.

[44] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 603–618, New York, NY, USA, 2017. Association for Computing Machinery. See page 6.

[45] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 10–15 Jul 2018. See page 6.

[46] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. See page 6.

[47] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 739–753. IEEE, 2019. See page 6.

[48] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pages 2512–2520. IEEE, 2019. See page 6.

[49] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin B. Calo. Analyzing federated learning through an adversarial lens. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 634–643. PMLR, 2019. See page 6.

[50] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *CoRR*, abs/1808.04866, 2018. See page 6.

[51] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2938–2948. PMLR, 2020. See page 6.

[52] Wenqi Wei, Ling Liu, Margaret Loper, Ka Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *CoRR*, abs/2004.10397, 2020. See page 6.

[53] Chien-Lun Chen, Leana Golubchik, and Marco Paolieri. Backdoor attacks on federated meta-learning. *CoRR*, abs/2006.07026, 2020. See page 6.

[54] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021. See page 7.

[55] Alex Ingerman Krzys Ostrowski. Tensorflow federated. Google, 2019. See pages 7, 15.

[56] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018. See page 7.

[57] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. *Federated Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. See page 7.

[58] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105, 2019. See page 7.

[59] Sajid A. Marhon, Christopher J. F. Cameron, and Stefan C. Kremer. *Recurrent Neural Networks*, pages 29–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. See page 8.

[60] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5):183–197, 1991. See page 8.

[61] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. See page 8.

[62] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. See page 8.

[63] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021. See page 8.

[64] Truong Thu Huong, Ta Phuong Bac, Dao Minh Long, Tran Duc Luong, Nguyen Minh Dan, Le Anh Quang, Le Thanh Cong, Bui Doan Thang, and Kim Phuc Tran. Detecting cyberattacks using anomaly detection in industrial control systems: A Federated Learning approach. *Computers in Industry*, 132:103509, 2021. See page 8.

[65] Ruijie Zhao, Yue Yin, Yong Shi, and Zhi Xue. Intelligent intrusion detection based on federated learning aided long short-term memory. *Physical Communication*, 42:101157, 2020. See page 8.

[66] Kuang Yao Lin and Wei Ren Huang. Using Federated Learning on Malware Classification. *International Conference on Advanced Communication Technology, ICACT*, 2020:585–589, 2020. See page 8.

[67] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad Reza Sadeghi. DÏoT: A federated self-learning anomaly detection system for IoT. *Proceedings - International Conference on Distributed Computing Systems*, 2019-July:756–767, 2019. See pages 8, 9.

[68] Viraaji Mothukuri, Prachi Khare, Reza M. Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava. Federated Learning-based Anomaly Detection for IoT Security Attacks. *IEEE Internet of Things Journal*, 4662(c):1–10, 2021. See page 8.

[69] Zhuo Chen, Na Lv, Pengfei Liu, Yu Fang, Kun Chen, and Wu Pan. Intrusion Detection for Wireless Edge Networks Based on Federated Learning. *IEEE Access*, 8:217463–217472, 2020. See pages 8, 16.

[70] Y. Qin and M. Kondo. Federated Learning-Based Network Intrusion Detection with a Feature Selection Approach. In *3rd International Conference on Electrical, Communication and Computer Engineering, ICECCE 2021*, 2021. See page 8.

[71] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences (Switzerland)*, 8(12):1–21, 2018. See page 8.

[72] Ana Cholakoska, Bjarne Pfitzner, Hristijan Gjoreski, Valentin Rakovic, Bert Arnrich, and Marija Kalendar. Differentially Private Federated Learningfor Anomaly Detection in eHealth Networks. (Ml):514–518, 2021. See page 8.

[73] Burak Cetin, Alina Lazar, Jinoh Kim, Alex Sim, and Kesheng Wu. Federated Wireless Network Intrusion Detection. *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, pages 6004–6006, 2019. See page 8.

[74] Pu Tian, Zheyi Chen, Wei Yu, and Weixian Liao. Towards asynchronous federated learning based threat detection: A DC-Adam approach. *Computers and Security*, 108:102344, 2021. See pages 8, 34.

[75] Brett Weinger, Jinoh Kim, Alex Sim, Makiya Nakashima, Nour Moustafa, and K. John Wu. Enhancing IoT anomaly detection performance for federated learning. *Proceedings - 2020 16th International Conference on Mobility, Sensing and Networking, MSN 2020*, pages 206–213, 2020. See pages 8, 9, 16, and 34.

[76] Ekaterina Khramtsova, Christian Hammerschmidt, Sofian Lagraa, and Radu State. Federated learning for cyber security: SOC collaboration for malicious URL detection. *Proceedings - International Conference on Distributed Computing Systems*, 2020-Novem:1316–1321, 2020. See pages 8, 9.

[77] Noor Ali Al-Athba Al-Marri, Bekir S. Ciftler, and Mohamed M. Abdallah. Federated Mimic Learning for Privacy Preserving Intrusion Detection. *2020 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2020*, 2020. See pages 8, 9.

[78] Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. Internet of Things intrusion Detection: Centralized, On-Device, or Federated Learning? *IEEE Network*, 34(6):310–317, 2020. See pages 8, 9.

[79] Ying Zhao, Junjun Chen, Di Wu, Jian Teng, and Shui Yu. Multi-task network anomaly detection using federated learning. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, pages 273–279, 2019. See pages 8, 9.

[80] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. See page 8.

[81] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. volume 5, pages 223–238, 05 1999. See page 9.

[82] Ahmed Shafee, Mohamed Baza, Douglas A. Talbert, Mostafa M. Fouda, Mahmoud Nabil, and Mohamed Mahmoud. Mimic learning to generate a shareable network intrusion detection model. In *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, 2020. See page 9.

[83] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, jun 2002. See pages 9, 34.

[84] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008. See page 9.

[85] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017. See page 12.

[86] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. See page 12.

[87] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *J. Mach. Learn. Res.*, 2:139–154, mar 2002. See page 12.

[88] Tensorflow Federated Google. Building Your Own Federated Learning Algorithm. https://tensorflow.google.cn/federated/tutorials/building_your_own_federated_learning_algorithm, 2022. [Online; accessed 04-September-2022]. See page 19.

[89] Wilhelm Kirch, editor. *Kappa CoefficientKappa coefficient*, pages 821–822. Springer Netherlands, Dordrecht, 2008. See page 21.

[90] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. See page 21.

[91] Yinglin Xia. Chapter eleven - correlation and association analyses in microbiome study integrating multiomics in health and disease. In Jun Sun, editor, *The Microbiome in Health and Disease*, volume 171 of *Progress in Molecular Biology and Translational Science*, pages 309–491. Academic Press, 2020. See page 21.

[92] Francisco Melo. *Area under the ROC Curve*, pages 38–39. Springer New York, New York, NY, 2013. See page 21.

[93] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA, 2006. Association for Computing Machinery. See page 21.