Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Decentralized Federated Learning for Network Security

Bc. Pavel Janata

Supervisor: Garcia Sebastian, Assist. Prof., PhD
Supervisor–specialist: Ing. Rigaki Maria
January 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Janata Pavel**	Personal ID number:	**465810**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Data Science**		

II. Master’s thesis details

Master’s thesis title in English:

Decentralized Federated Learning for Network Security

Master’s thesis title in Czech:

Decentralizované federativní učení pro síťovou bezpečnost

Guidelines:

ederated Learning (FL) is a type of decentralized learning that allows for collaborative training of machine learning models without the explicit exposure of participants' private data. This is very important for network security, where data privacy is of great concern. The traditional collaborative model of FL uses a single aggregator node, but this is not desirable in certain scenarios and a decentralized approach may be required.
The goal of the thesis is to review the existing state-of-the-art of Federated Learning and to propose a decentralized FL method for network security. The approach should be able to detect threats in peers and benefit from the cooperative training of the model. The models should be evaluated under realistic conditions, including statistical heterogeneity of the data and handling an imbalance in both the participants' computational resources and data volume. The student should consider taking advantage of a trust model between participants to mitigate the security concerns. Both supervised and unsupervised methods should be evaluated. The developed method should be compared to a similar state-of-the-art method.

Bibliography / sources:

Kairouz, Peter, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, et al. 2019. “Advances and Open Problems in Federated Learning,” December. https://ieeexplore.ieee.org/document/9464278.
Rey, Valerian, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and Gérôme Bovet. 2022. “Federated Learning for Malware Detection in IoT Devices.” Computer Networks 204 (February). https://doi.org/10.1016/j.comnet.2021.108693.
Wang, Jianyu, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Aguera y Arcas, Maruan Al-Shedivat, et al. 2021. “A Field Guide to Federated Optimization.” ArXiv:2107.06917 [Cs], July. http://arxiv.org/abs/2107.06917.

Name and workplace of master’s thesis supervisor:

Ing. Sebastián García, Ph.D.   Artificial Intelligence Center FEE

Name and workplace of second master’s thesis supervisor or consultant:

Ing. Maria Rigaki   Department of Computer Science FEE

Date of master’s thesis assignment: **25.07.2022**   Deadline for master's thesis submission: **10.01.2023**

Assignment valid until: **19.02.2024**

Ing. Sebastián García, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisors *García Sebastián, Assist. Prof. PhD* and *Ing. Maria Rigaki* for all their patience and the help they provided me over the past year. I also have to thank everyone in the Stratosphere Laboratory for providing a warm and welcoming environment for writing this thesis.

Finally, I would like to thank my wife, Anna, and the rest of my family for being with me and providing me with their unending support during all my years of study.

Declaration

I hereby declare that the presented work has been composed solely by myself and that I have listed all sources of information used within it in accordance with the methodical instructions about ethical principles in the preparation of academic theses.

V Praze, 10. January 2023

Abstract

Network security is an increasingly important concern in today's connected world as the number and complexity of threats continue to grow. Federated learning (FL) is a machine learning method to distributedly train an model using clients' data while protecting their privacy. In this thesis, we present an FL solution for network security, specifically for detecting malware activity in HTTPS traffic. We developed both supervised and unsupervised methods for detecting malware in the clients' data. We evaluate our methods using the CTU-50-FEEL dataset, which contains realistic benign traffic of ten users spanning five days, as well as traffic of six distinct malware. Our experimental results show that our federated learning approach is able to detect a wider range of threats with higher accuracy than if the clients relied only on their own data to create their models. Overall, our work demonstrates the feasibility of using Federated Learning for detecting malware activity in clients with non-IID network traffic while preserving their privacy.

Keywords: Federated Learning, Network Security, HTTPS, Malware, Machine Learning, Variational Autoencoder, Anomaly Detection, Classification

Supervisor: Garcia Sebastian, Assist. Prof., PhD

Abstrakt

Síťová bezpečnost je v dnešním propojeném světě stále důležitějším problémem, protože počet a složitost hrozeb neustále roste. Federativní učení (FL) je metoda strojového učení, která umožňuje distribuovaně trénovat model s využitím dat klientů a zároveň chránit jejich soukromí. V této práci představujeme FL řešení pro síťovou bezpečnost, konkrétně pro detekci aktivity malwaru v HTTPS provozu. Vyvinuli jsme metody s učitelem i bez učitele pro detekci malwaru v datech klientů. Naše metody vyhodnocujeme pomocí datové sady CTU-50-FEEL, která obsahuje realistický „neškodný“ provoz deseti uživatelů v rozpětí pěti dnů a také provoz šesti různých druhů malwaru. Naše experimentální výsledky ukazují, že náš přístup založený na federativním učení je schopen detekovat širší škálu hrozeb s vyšší přesností, než kdyby se klienti při vytváření svých modelů spoléhali pouze na svá vlastní data. Celkově naše práce prokazuje proveditelnost použití federativního učení pro detekci aktivity malwaru u klientů s non-IID síťovým provozem při zachování jejich soukromí.

Klíčová slova: Federativní Učení, Síťová Bezpečnost, HTTPS, Strojové Učení, Variační autoencoder, Detekce Anomálií, Klasifikace

Překlad názvu: Decentralizované federativní učení pro síťovou bezpečnost

Contents

Figures

Tables

Chapter 1

Introduction

Network security is an increasingly important concern in today's connected world as the number and complexity of threats continue to grow [1, 2]. Many solutions exist, both using conventional threat intelligence and machine learning (ML) methods to address these security threats. However, one challenge of using ML for this purpose is that algorithms need to be trained with data that is usually private, since the organizations owning them may be unwilling to share the data with others, making a traditional or centralized ML solution hard to implement. Moreover, centralized machine learning models may need vast computational and data storage resources. Therefore, Federated learning (FL) is proposed by the community as a ML method where each client trains its own model in its own private data and then shares the model updates with a central aggregator, which creates a global model for all the clients. FL is a way to collaboratively and distributedly train with clients' data while protecting their privacy. This thesis presents an FL solution for network security.

There has been a growing interest in using federated learning for various applications in network security. Previous research has demonstrated the feasibility of using federated learning for tasks such as intrusion detection, sharing cyber threat intelligence, and network traffic classification. These studies have shown that federated learning can achieve good performance while preserving the privacy of individual networks [3, 4].

While federated learning has shown promising results for various applications in network security, there are still many open research questions and challenges to be addressed. One important area of focus is the heterogeneity of data across different networks, which can pose a challenge for the models trained using FL, which often struggle to converge on data which is not independent and identically distributed (non-IID). This is especially relevant in the context of network security, where data may vary significantly due to the diversity of both benign traffic generated by users and malicious traffic originating from a wide range of malware. Additionally, most of the existing research in federated learning for network security has focused on specific domains, such as the Internet of Things (IoT), or targeted a narrow range of threats. There is a need for more research that addresses the generalizability and scalability of federated learning to a wider range of network security scenarios.

The usage of HTTPS has become widespread in the past years [5], and its privacy benefits to the users are well known. However, new versions of malware also started taking advantage of it, as encrypting their traffic makes them more difficult to discover. There is a growing need to develop new methods for detecting these types of malware. The traditional machine learning methods rely on gathering data from users to train models. In the network security setting, this translates to collecting potentially sensitive information about the users' network traffic and thus violating their privacy. This privacy issue is addressed by using Federated Learning. We have used FL to train both unsupervised and supervised machine learning models for detecting malware activity. Our methodology involves multiple clients participating in the training process, some of them only with benign traffic. This represents a particularly challenging scenario due to the extreme class imbalance across the clients, where the majority of traffic is benign and only a small fraction is malicious. Moreover, this thesis uses, as far a we know, for the first time the concept of network *malware vaccines*, which in our case, are a small dataset of malicious feature vectors that can be shared with the clients and incorporated into their local datasets for federated training. Vaccines for malware binaries and system infections have been known for decades [6], but no use of network traffic vaccines has been reported so far. Results of our experiments show that the use of vacciones significantly improves the convergence and performance of the global model, which is then able to detect even types of threats not originally present in the vaccine.

To train and evaluate our proposed FL methods, we use the *CTU-50-FEEL* dataset [7] developed in the Stratosphere Laboratory [8]. It contains everyday traffic of ten different users using a diverse mix of operating systems and applications. The malicious traffic is from malware deployed in the Stratosphere Laboratory. The traffic captures span five consecutive days allowing for the simulation of longer-term deployment with realistic differences in distributions and activity between the days.

The evaluation the proposed FL methods is done in a series of experiments using both supervised and unsupervised algorithms. We designed different scenarios to evaluate the FL methods. These scenarios are designed to find out how is the performance of the FL model affected by fewer clients participating in the training. The scenarios are also designed to evaluate if FL models from previous days can be reused and retrained on the following days. To test the methods under harder conditions, we created a variant of the dataset called *CTU-50-FEEL-less-malware* which has more challenging properties for detection. This variant of the dataset contains fewer infected clients each day, making the dataset more imbalanced. With fewer infections, there are spans of no activity for some malware, enabling us to evaluate the ability of the models to remember past threats.

Every FL experiment consist of three sub-experiments: (i) the FL experiment where clients send updates to a central aggregator and receive a global model; (ii) a local experiment where each client trains alone and tests alone; (iii) a central experiment where we get all the data together as if there were no clients, and no privacy concerns. The central experiment provides an upper bound on the performance that can be achieved with FL. In contrast, the (ii) local learning setting illustrates a case where data privacy is a concern, and FL nor any other communication is used. As such, each client trains a local model using only its own data, without any collaboration.

Results shows that Federated Learning is a viable method for detecting malware that use TLS encrypted traffic. The FL models consistently outperform the models trained only on the local data of the clients. We also show that in the multi-day deployment scenarios, models from previous days can be retrained on new data using fewer training rounds while preserving comparable performance to the models trained from scratch. The use of the vaccine enabled the training of supervised detection models with both infected and non-infected clients. The FL methods achieved overall high accuracy with a minimal performance penalty compared to the experiments of type (iii) where the dataset was completely centralized.

All our experiments showed that there is a clear performance benefit to the federated training (i) over the local setting (ii). In the unsupervised experiment, the collaboratively trained anomaly detection model achieved a 95.26 % accuracy, while the models trained only on the local data reached average accuracy of 94.80 %. The best unsupervised results were achieved in the central experiments (iii) with an average accuracy of 96.55 %, but these scenarios require violating the privacy of clients' data.

The benefits of the clients' collaboration are even more pronounced in the supervised experiments. The type (iii) experiments on the completely centralized dataset had an average accuracy of 99.80 %, while in the federated experiments (i), the models achieved only a slightly worse accuracy of 99.46 %, while providing stronger privacy guarantees to the clients. Both these scenarios are an improvement over the type (iii) experiments with no cooperation, in which the models had an average accuracy of 98.27 %.

These results support the idea that cooperation is beneficial to developing new malware detection methods in network security. They also show that Federated Learning is a viable method to achieve collaboration without compromising the privacy of the data. Although there is a performance penalty to the models, it may often be outweighed by privacy concerns.

The novel contributions of this thesis are:

- A new dataset called CTU-50-FEEL with HTTPS features for the CTU-50 dataset.

- A multi-head neural network architecture that includes a autoencoder and fully connected neural network.
- Highly unbalanced dataset variations and experiments for realistic extreme malware classes.
- A multi-day methodology to accelerated training with past models.
- Use of network data vaccines sent to the clients to improve detection.

Chapter 2

Theoretical Background

This chapter aims to give a theoretical overview of areas and methods related to or directly used in this work. We introduce some commonly used metrics used for evaluating and comparing machine learning models, describe neural networks and how they are trained, and then focus on specific types of neural networks.

Then we focus on the area of Federated Learning, describe the motivation behind it, how it achieves its goals, and classification of the types of FL. We introduce the field of Federated Optimization, which devises algorithms and methods for training machine learning models in the federated setting. Finally, we focus on the challenges commonly encountered when developing FL methods.

2.1 Supervised and Unsupervised Learning

An often-used categorization of machine learning is supervised and unsupervised learning [9, Chapter 5]. This splits most of the algorithms based on what kind of data they are able to observe when learning.

In supervised learning, the algorithm is given information about the label or target for each data point. For example, in the case of classification, the label provides information to which category an input sample belongs.

In unsupervised learning, the algorithm is not given such information and can only observe the structure and properties of the dataset. Tasks on such datasets could be, for example, to somehow learn the underlying probability distribution or to cluster together similar samples.

However, the boundary between supervised and unsupervised learning can become blurry, as in some cases, there does not have to be a label for every data point in the dataset (we refer to this as semi-supervised learning), or the label information can be weak or unreliable [10]. Active learning systems are designed to use as little labeled information as possible to achieve their learning goals and usually work in a framework where they query the user to provide additional labeled information during the learning process.

2.2 Performance Measures

We use performance measures to quantitatively evaluate a machine learning algorithm on a given task [9, Chapter 5].

To properly evaluate the algorithm, we usually use a test dataset - a collection of data points separate from the training data. The performance measures vary based on the types of tasks and used data.

We will describe specific performance measures for binary classification as they are particularly relevant to this work. Figure 2.1 shows the four possible outcomes of classifying a data point. This structured table is referred to as a confusion matrix, and it contains the absolute counts of the possible outcomes. We will use the notation in the table to express the measures [11].

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 2.1: Confusion matrix

Accuracy is one of the intuitive measures, and it expresses the ratio of correctly classified samples over the whole dataset.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

True Positive Rate (TPR), sometimes referred to as *recall* or *sensitivity*, is the proportion of positive samples which are correctly identified as positive.

$$TPR = \frac{TP}{TP + FN} \tag{2.2}$$

Similarly, True Negative Rate (TNR or $specificity$), False Positive Rate (FPR or fall-out), and False Negative Rate (FNR) can be expressed as

$$TNR = \frac{TN}{TN + FP} \qquad FPR = \frac{FP}{FP + TN} \qquad FNR = \frac{FN}{FN + TP} \tag{2.3}$$

Precision expresses the proportion of real positives in the positively predicted values

$$precision = \frac{TP}{TP + FP} \tag{2.4}$$

Finally, the *F-score* combines the *precision* and *recall* measure using their harmonic mean

$$F\text{-}score = 2\frac{precision \cdot recall}{precision + recall} \tag{2.5}$$

It is also referred to it as a balanced *F-score* as it weights both measures equally or F_1 coming from a generalized variant F_β which uses a β parameter to put β-times more importance on the *precision* metric.

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.6}$$

2.3 Feed-Forward Neural Networks (FF-NN)

Neural Networks (NN) are a group of machine learning models that are composed of a number of units that were originally meant to resemble biological neurons[12]. In Feed-Forward NN, the neurons are organized into units, and the information flows in one direction from one layer to another.

Goodfellow et al. [9, Chapter 6] describe Neural Networks as a composition of multiple functions. A case of a simple network with three layers could be described using functions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$. Those layers are connected in a chain forming $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. $f^{(1)}$ is called first layer or input layer as it takes the input vector $\mathbf{x}$; $f^{(2)}$ is referred to as a second layer, and the final layer is usually called output layer. The layers between the input and output layers are collectively referred to as hidden layers, and the total number of layers in a chain gives the depth of the model. The goal of FF-NN is to create such function $f(\mathbf{x})$, which approximates some function f^*.

Each layer j has a weight matrix $\mathbf{W}_j$ associated with it, a bias vector $\mathbf{b_j}$, and an activation function φ_j. The i-th row of the $\mathbf{W}_j$ represent the weights vector $\mathbf{w}_{ji}$ of i-th neuron in layer and b_{ji} its bias term. The output of the layer j is then

$$f_i^{(j)}(\mathbf{x}_{j-1}) = \varphi(\mathbf{x}_{j-1}^\intercal \mathbf{W}_j + \mathbf{b}_j)$$

Where $\mathbf{x}_{j-1}$ is the output of the previous layer. The activation layer is usually some non-linear function that is applied element-wise. The non-linearity of ϕ gives NN models the ability to approximate complex functions. A wide variety of different activation functions are used in modern neural networks, we provide an example of some of the common ones in the list below and their plots in Figure 2.2.

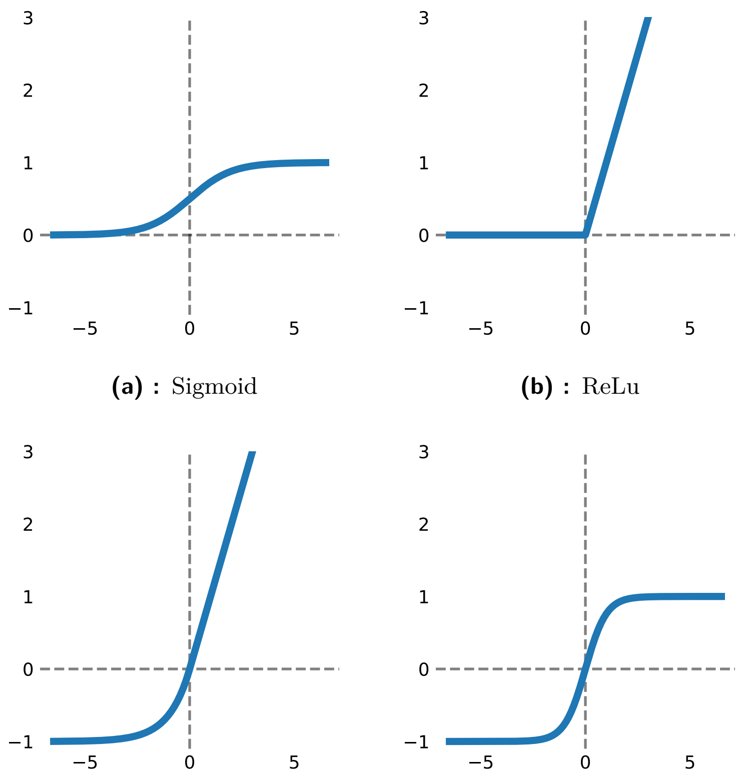
- Sigmoid: $s(x) = \frac{1}{1+e^{-x}}$
- Rectified Linear Unit (ReLU): $r(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
- Exponential Linear Unit (ELU): $e(x) = \begin{cases} e^x - 1 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
- Hyperbolic Tangent function (Tanh): $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

2.4 Training of Neural Network Models

The process of training refers to an effort to find parameters θ such that the network $f(\mathbf{x}, \theta)$ best approximates function $f*(\mathbf{x})$. In the case of NN, the θ represents the weights and biases of the individual layers [9, Chapter 8]. As $f*$ represents the desired function, we usually use a set of samples $\mathbf{x}$ and target values $\mathbf{y}$. A loss function $\mathcal{L}(\mathbf{y}, f\mathbf{x})$ is used to compare the outputs of the model to the desired ones.

We can formalize the training as an optimization process of finding parameters that will minimize the sum of loss over each sample. In order to improve the loss of the model, we need a way of computing its gradient to inform us how to adjust the weights. The backpropagation algorithm [12] is used for this purpose. It progresses in the opposite direction from the inference - from the last layer to the first. Since the $f(\mathbf{x})$ representing FF-NN consists of multiple layers compounding to produce the output, the partial derivative can be obtained using the chain rule. As its output, the algorithm produces a gradient of the loss function with respect to the parameters of the model as a function of the inputs $\mathbf{x}$ and outputs $\mathbf{y}$.

In practice, the whole set of training samples is not used to compute the gradients, and instead, an algorithm called Stochastic Gradient Descent (SGD) is employed, which uses an average gradient over a subset of the data to compute updates to the weights. The algorithm shows the process of



(a) : Sigmoid **(b) :** ReLu

(c) : ELU **(d) :** Tanh

Figure 2.2: Examples of commonly used activation functions.

computing new weights using a part of the dataset (batch), and a learning rate parameter η controls the step size of the update. Traditionally, the batches are obtained by splitting the training data into equally sized partitions, and one iteration over all batches is referred to as an epoch.

The size of the batch controls the amount of stochasticity in the process. With larger batch sizes, the update steps approach the true gradient of the whole dataset, but they can also more easily get stuck in a local minimum of the non-convex function. Decreasing the size of the batches can somewhat mitigate it and also uses less memory, but the algorithm will take longer to compute the larger number of updates.

There exists a large number of algorithms driving from SGD and improving its convergence speed. The often-used method is momentum which introduces a velocity variable $\mathbf{v}$. It accumulates the gradients from previous steps and enables the algorithm to take larger steps if multiple successive updates point in the same direction.

One particular method which uses momentum is Adaptive Moment Estimation (referred to as Adam [13]). It uses the first and second momentums of the gradient, and it also individually adapts the learning rates to progress faster in less sloped areas of the parameter space. Although this algorithm is more computationally demanding, it accelerates the convergence and seems to be more robust in the choice of hyperparameters.

2.5 Autoencoders

An autoencoder (AE) is a type of neural network architecture which is trained to produce copies of its inputs as its outputs [9, Chapter 14]. The particular type of autoencoders we deal with are Undercomplete Autoencoders shown in Figure 2.3. They are split into two parts. An encoder projects the input into some lower dimension (latent space), and a decoder attempts to reconstruct the input from it. The model is trained using a reconstruction loss function which measures the differences between the input and output of the model. Mean square error (MSE) is often used as such a loss.

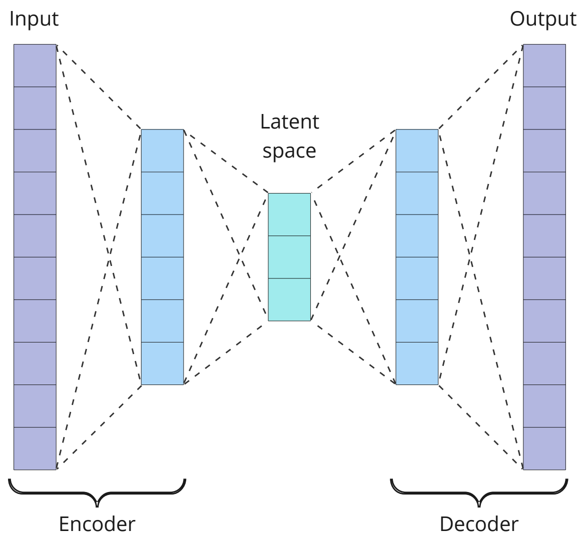


Figure 2.3: A structure of an *Undercomplete Autoencoder*. It excepts 10-dimensional vectors as its input, and the encoder projects them down to 3 dimensions from which the decoder attempts to reconstruct the original vector.

Since the model only compares its input to its output, it does not need any label information and can be used for unsupervised learning. The encoder can be used as a method for non-linear dimensionality reduction or to extract useful features from the data. Autoencoders can also be used in generative tasks, as the decoder can produce samples for any point in the latent space. They can also be used for anomaly detection by training the model on normal data and relying on the model to create an output with a greater loss for outlier samples.

A special variant of autoencoder is Variational Autoencoders (VAE) [14]. The VAE does not produce the embedding directly but instead uses a probabilistic distribution which it learns - an unbiased multivariate normal distribution in our case. The encoder is trained to produce mean (μ) and log-variance vectors ($log(\sigma^2)$), which are then used when sampling the normal distribution. The log variance is used to prevent the model from producing negative values.

To train the model using backpropagation, a representation trick is used for sampling the normal distribution. A random vector ε is sampled from the standard normal distribution, which is scaled by the variance vector and added to the mean vector. By using a specific distribution, VAEs tend to generalize better and are less prone to overfitting. They are also better suited for generative tasks as the encoder can generate multiple samples coming from the same distribution.

To train VAE, a reconstruction loss is used as in a regular autoencoder. To it, a Kullback-Leibler divergence is added as a regularization term which discourages the learned distribution from diverging. This metric is used to compare two probabilistic distributions, and in this case, it describes the distance of the model's diagonal multivariate normal distribution to a standard normal distribution as can be expressed as

$$D_{KL}\left(\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2),\mathcal{N}(\mathbf{0},\mathbf{1})\right) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - log(\sigma_i^2) - 1)$$

2.6 Federated Learning

Since the introduction of Federated Learning (FL), many variants have been developed targeting specific use cases. The basic version of FL assumes a central aggregator node, which communicates with a number of clients and orchestrates the model training with them [15]. In this case, the clients hold the data, which usually have different distributions, but the features observed in every client are the same, and the different clients usually hold data describing different entities. In this chapter, we introduce some variants of Federated Learning which differ in those basic assumptions.

2.6.1 Cross-device and Cross-silo Federated Learning

The first applications of FL focused mainly on training on edge devices like mobile phones or IoT devices. We refer to those approaches as *cross-device* as the learning and data are present on the devices themselves. This is to oppose a *cross-silo* approach, where the data is usually first siloed across an organization onto a more powerful server, which then serves as a node participating in the Federate Learning. This, for example, allows multiple organizations to collectively train a model without sharing their data directly.

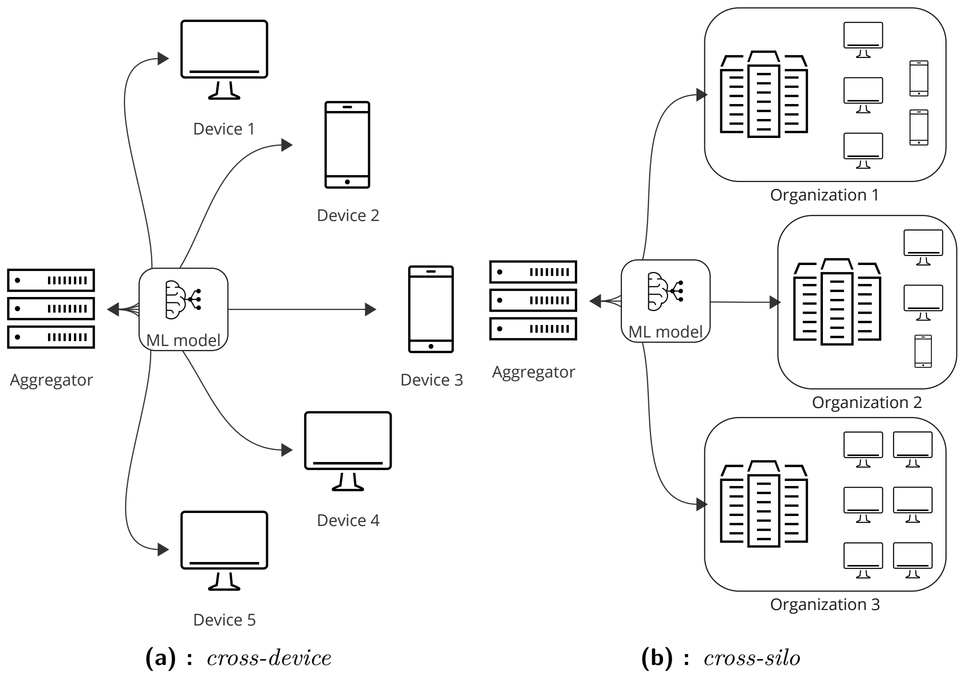


Figure 2.4: Classification of Federated Learning based on the type of participants. In *cross-device* FL, the models are trained in the devices from which the data originates. As opposed to *cross-silo*, where the participants are usually larger organizations that have dedicated servers in which the data of the organization are siloed. These servers usually provide better reliability and more computational resources available for the training. The number of participants is generally lower in the cross-silo setting.

Cross-device methods are often designed to handle thousands of participants and are thus more difficult to coordinate and can have higher demands on transmission efficiency. Furthermore, the edge devices participating in cross-device learning often offer limited computational power or availability. For instance, when training on mobile phones, conditions such as the charging state, network connectivity, and the user's activity must be considered [16, 17, 18]. This is usually done by selection strategies, which would only accept participating devices that fulfill defined conditions. It is also assumed that some devices might become unavailable during the training process.

In the cross-silo scenario, the data from an organization are usually concentrated on a single machine within an organization. This and a number of participants as low as two, allow for stronger assumptions on availability, computational resources, and network connectivity.

Sometimes, the cross-silo approach is employed when organizations share the same entities in their datasets but observe different features describing them, but for privacy reasons, they can't share data directly. This can be the case, for example, when hospitals and laboratories conduct medical tests and have data about the same individuals. Due to the high privacy concerns related to medical records, privacy-preserving FL can be the only option to train a model using both datasets. This approach is called Vertical Federated Learning and is further discussed in the next section.

2.6.2 Horizontal and Vertical Federated Learning

One of the key classifications of Federated Learning is how the samples and features are partitioned among the clients participating in FL. When the features in the datasets are the same, but the samples come from different (or partially overlapping) sets of entities, it is called Horizontal Federated Learning (HFL). The opposite case, when clients share different features from the same set of entities, it is called Vertical Federated Learning (VFL).

HFL is applicable in cases where similar behaviors can be observed in a number of clients. For example, it can be used to improve wake word detection models used by AI assistants. In this case, each device collects the same audio features, but each device contains only samples generated by a single user. In this setup, a CNN is able to learn to detect key phrases with high accuracy without the data leaving the edge devices.

The concept of the vertical split of the data is more applicable in the cross-silo FL. It assumes that all participants are in possession of data from the same ID space but different feature spaces. This has direct effects on the architecture of the models. In HFL, all clients generally have the same models, which are being locally updated by them and then synced through a central coordinator. While in VFL, the clients usually only maintain part of the model, and both training and inference have to be done in a more coordinated manner.

In the case of neural networks, each client maintains a part of the whole model. During training, it inputs its set of features through its model and sends the output to a participant in possession of the labels. The responsibility of the label owner is to produce an output of the whole model and send the backward messages to all participants so that they can update their models. One of the pre-requisites for this process of training is the establishment of a common set of IDs of samples which further adds to the coordination complexity.

2.6.3 Federated Optimization

In this subsection, we will introduce the main methods used for federated optimization, which refers to the optimization algorithms used in Federated Learning. The focus of this subsection is on introducing the main methods used for training models in FL, which are mostly extensions of Stochastic Gradient Descent (SGD) and can be used to train a variety of models, including

Neural Networks [15], Linear Regression [19], Support Vector Machine [20], Gradient Boosting Trees [21], and Random Forests [22].

Federated Learning SGD can be implemented with the Federated Averaging algorithm (FedAvg) (described in Section 2.6.3). Since its introduction, new methods were proposed to extend and improve upon it in order to achieve better stability, and performance. This includes both the performance of the model and the computational efficiency of the training process.

Federated Averaging

FedAvg was proposed by McMahan et al. [15], and it can be viewed as an extension of Stochastic Gradient Descent and is described in Algorithm 1. The training of the models is done in rounds, coordinated by a central aggregator server. The server initializes a model $\mathbf{w}_0$, which then broadcasts to all clients who take part in the training. The clients then apply a local SGD using their data $\mathcal{D}_k$ for a number of epochs - E. After the clients finish their rounds, they send the adjusted weights $\mathbf{w}_t^k$ of their models back to the central server, which aggregates the weights by computing their average. This results in a new global model $\mathbf{w}_t$, which is broadcasted to the clients in the next round of training.

The main contribution of the FedAvg compared to previous algorithms used distributed learning [23] is the introduction of the multiple local epochs. This allows for more communication-efficient learning as the model parameters are transmitted less often. However, higher values of E also lead to more divergence between the clients' models in the case of non-IID data.

Adaptive Optimization

Extensions of FedAvg were developed to allow for better control of the training process. One of those is FedOpt proposed by Reddi et al. [24], which generalizes the "vanilla" FedAvg and enables the use of momentum-based and adaptive optimization techniques. Algorithm 2 shows the federated variant of the ADAM optimizer, which utilizes these techniques together with separate learning rates for the client and server procedures. Similarly, as in traditional machine learning, the adaptive methods in a federated setting tend to have better convergence characteristics and necessitate less hyperparameter tuning.

It is important to note that in some cases, the clients may be unable to maintain a persistent state between the rounds. For example, when the client population is large, and the participants in each iteration are sampled. In this case, many clients may only be used once, and the local learning rates can not be applied.

Algorithm 1 Federated Averaging algorithm [15]. S_t is the set of clients in time t, and they are indexed by k; $\mathbf{w}$ are parameters of the model, η is the learning rate and l is the loss function.

```
procedure SERVER EXECUTION
    initialize $\mathbf{w}_0$
    for round $t \leftarrow 1, 2, ...T$ do
        for client $k \in S_t$ in parallel do
            $\mathbf{w}_{t+1}^k \leftarrow$ CLIENTUPDATE($k, \mathbf{w}_t$)
        end for
        $\mathbf{w}_{t+1} \leftarrow \frac{1}{k} \sum_{k=1}^{K} \mathbf{w}_{t+1}^k$
    end for
    return $\mathbf{w}_T$
end procedure

procedure CLIENTUPDATE($k, \mathbf{w}$)                                ▷ Run on client $k$
    $\mathcal{B} \leftarrow$ (Split local dataset $\mathcal{D}_k$ into batches of size $B$
    for local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla l\,(\mathbf{w}; b)$
        end for
    end for
    return $\mathbf{w}$ to server
end procedure
```

Regularization

In FL, clients often perform multiple local updates before sharing them with the aggregator server in order to save on communication overhead. This often leads to client drift where the models of the individual clients diverge. This can be mitigated by using regularization, which can be viewed as a term added to the loss function, which penalizes the drifting of the local model from the global one.

By choosing the function $\psi(\mathbf{w}, \mathbf{w}_g)$, a distance function of the local model $\mathbf{w}$, and the latest global model $\mathbf{w}_g$, we can ensure that the local models will not diverge significantly. Examples of the regularization functions can be found in the FEDPROX [25], which introduces the proximal term

$$\psi_i(\mathbf{w}, \mathbf{w}_g) = \frac{\mu}{2}(\|\mathbf{w} - \mathbf{w}_g\|)^2$$

where μ is a hyper-parameter of the method. The proximal term is added to the loss function as a penalty.

Algorithm 2 FedAdam algorithm[24]. S_t is the set of clients in time t, and they are indexed by k; $\mathbf{w}$ are parameters of the model, η_c and η_s are the client and server learning rates, l is the loss function, $\mathbf{m}$ and $\mathbf{v}$ are the momentum and velocity vectors, $\beta_1, \beta_2 \in [0, 1)$ are hyper-parameters controlling their decay and ϵ is a small non-zero constant

```
procedure Server execution($\mathbf{w}_0, \mathbf{m}_{-1}, \mathbf{v}_{-1}, \beta_1, \beta_2, \epsilon$)
    for round $t \leftarrow 1, 2, ...T$ do
        $\mathbf{w}_t^k \leftarrow \mathbf{w}_t$
        for client $k \in S_t$ in parallel do
            $\Delta_t^k \leftarrow$ClientUpdate($k, \mathbf{w}_t$)
        end for
        $n \leftarrow \sum_{k=1}^{K} n_k, \Delta_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \Delta_t^k$
        $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \Delta_t$
        $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \Delta_t^2$
        $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t \eta_s \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t} + \epsilon}$
    end for
    return $\mathbf{w}_T$
end procedure

procedure ClientUpdate($k, \mathbf{w}$)                                  ▷ Run on client $k$
    $\mathcal{B} \leftarrow$ (Split local dataset $\mathcal{D}_k$ into batches of size $B$
    for local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $\mathbf{w}' \leftarrow \mathbf{w} - \eta_s \nabla l(\mathbf{w}; b)$
        end for
    end for
    $\Delta_t^k \leftarrow \mathbf{w}' - \mathbf{w}$
    return $\Delta_t^k$ to server
end procedure
```

Model Weighting

During the model aggregation step on the central server, a weighting scheme is often employed. For example, in the FedAvg algorithm, the global model can be obtained as a weighted average of the clients' contributions. Weighting is most often employed in cases of data or performance heterogeneity, as described in Section 2.6.4. Its main purpose is usually to mitigate the tendency of the model to favor clients with higher volumes of training data or available resources. However, employing some weighting scheme may require the clients to share information about their data which might bring technical and privacy concerns. For example, when weighting is based on the number of samples used for training the model, the clients inevitably leak the amount of data they contain.

2.6.4 Challenges of Federated Learning

Federated Learning is a collaborative effort to extract knowledge from a set of clients while ensuring their data privacy. As this process is usually distributed, it shares some challenges with Distributed Learning while also introducing some unique ones. In this chapter, we will discuss some of those challenges and how they are usually addressed.

Data Heterogeneity

One of the most discussed and wildly studied challenges in Federated Learning is that the data across clients is often not identically and independently distributed (non-IID). This usually arises when each device contains data generated by a single user or a small set of them. It means that it is more common in the cross-device setting, but organizations in cross-silo settings can also each have uniquely skewed data.

The non-IID property of the data can manifest itself in a multitude of ways. The class balance can be different across clients, some labels being overrepresented on some sets of clients while non-present on others at all. The quality of labels can also vary across clients, and it is possible that the same features can have different labels across devices.

The non-IID data was presented as a key challenge in Federated Learning since its introduction[15], and it is often viewed as a requirement for realistic FL datasets.

Non-IID data is not unique to FL, and it is a challenge with which many ML models struggle. Robust statistics is an area of research focusing on methods applicable to data coming from a wide range of distributions. Using some robust techniques can help with FL on non-IID data. Examples of this can be extensions of FedAvg (see Section 2.6.3), which use a median or trimmed mean instead of a simple mean for weight aggregation [3]. These robust methods offer resiliency against outliers but at the cost of losing information, leading to slower convergence and worse performance.

A different approach is to relax the requirement of training a single global model. As the issues of non-IID data arise from the heterogeneity of the client's data, a possible solution might be to personalize the models for each client. A common approach is to use a method common in transfer learning - local fine-tuning[26, 27, 28]. In it, a shared FL model is first acquired, which is then fine-tuned on the client's own data to achieve personalization. Work has been done relating FL to Meta Learning [29] when viewing clients as heterogeneous tasks. This allows for the use of Model Agnostic Meta Learning (MAML) [30], which strives for general models which can be quickly adapted for specific tasks.

Device Heterogeneity

In many practical FL deployments, there exist significant differences among the clients. We discussed the differences in their data in the previous section, but apart from those, the clients can also vary in the available performance and communication resources as well as in the amount of data they can collect.

The practical challenge of training a federated model is that as the individual clients differ in their performance, it may take longer for some of them to finish iterating a batch of training data. This leads to some of the clients sitting idle and the wall-clock time of the training increasing [16].

A solution is to set a fixed time limit and allow the clients who would have finished sooner to continue training locally and omit the contributions of the struggling clients. Another option is to use asynchronous aggregation instead of conducting the training in distinct rounds. This allows the clients to share their contributions with the aggregating server as often as they are capable [31].

However, both of those approaches introduce bias favoring clients that are capable of finishing more batches. To mitigate this, a weighing schema is usually employed in the aggregation, decreasing the weights of contributions produced by the more active client [32]. A similar schema is also used to promote contributions from clients with fewer training samples available, to decrease the bias from the data volume heterogeneity.

Communication Efficiency

When deploying FL on IoT devices or mobile phones, communication overhead can become a significant obstacle. The main sources of bandwidth consumption is the aggregation server sharing the global model with the clients and the clients sending their gradients back to it. Some common techniques used to decrease the model size can be applicable here, e.g., sparsification or quantization.

Gradient compression is a popular technique that reduces the size of the gradients by only preserving the most important components and sparsifying them [26, 33].

Another approach is to employ a Multi-Epoch aggregation, where in each round the clients train their local models for multiple epochs, and only after that they share their updates with the aggregator [15]. This decreases the total number of communication rounds but also allows the local models to drift from each other.

These techniques usually represent a trade-off between the communication efficiency and the convergence time of the model and its accuracy.

Chapter 3

Previous Work

While Federated Learning is a relatively new area of research, it quickly found its applications in various fields. The focus of this chapter is to introduce some of the works relevant to applying FL in network security. In network security, there is a need to detect threats using a vast amount of data that is distributed between a large number of devices. Learning from this data can pose performance challenges and bring up privacy concerns. As the data is usually some representation of the network traffic, it may contain sensitive information. This might lead to Federated Learning being a good fit for these applications and, in some cases, the only option because of legal regulations.

Many of the published works focus on the area of the Internet of Things (IoT). One of the most relevant works to this thesis is by Rey et al. [3] on detecting malware in IoT devices. The authors compare both supervised and unsupervised methods and propose robust model aggregation techniques to handle non-IID data and to make the models more resilient to certain types of attacks. This work was tested on a static IoT dataset in which every device was infected with two types of malware (Mirai[34] and BASHLITE[35]). In contrast, in our work, we evaluated the developed methods on a dataset with a wider range of malware that uses HTTPS. Our dataset also spans multiple days, with significant changes in the data distribution. The malware is contained only in a subset of the clients, making it more challenging to train the supervised detection models.

Other works in IoT focused on anomaly detection using sequential models. The work done in [36] shows the importance of being able to capture the temporal information of the data for network security applications. They have evaluated the use of two types of recurrent models, a *Long Short-Term Memory* (LSTM) and *Gated Recurent Unit* (GRU). Their anomaly detection models were tested to detect various types of *Distributed Denial of Service* as well as *Man-in-the-Middle* attacks. Previous work has shown the viability of using the more demanding recurrent models in a resource-constrained FL environment [37] by employing techniques such as gradient compression.

De Carvalho Bertoli et al. [38] decided to focus on a narrow threat in network intrusion detection and designed a federated system for detecting port scanning attacks. They have evaluated a Logistic Regression trained using FedAvg on a dataset with 13 siloed agents, each with different distribution of benign and scanning traffic. In the majority of clients, the FL outperformed a locally trained model. However, their federated approach struggled with agents with significantly different class balances.

The heterogeneity of the clients' datasets is an inherent problem in Federated Learning, and it can be especially challenging in network security. This led some researchers to reject the FL goal of obtaining a single global model. Segmented Federated Learning [39, 40] was proposed for network intrusion detection. Segmentation allows for multiple global models to be maintained. Clients are periodically evaluated and reassigned to a group with the best fitting model.

In Segmented Federated Learning, all clients are assigned to a single segment at the beginning. In the following training rounds, the diverging clients split away from it to form new segments or join already existing ones. However, this often leads to a single group containing the majority of the clients and the rest scattered among groups of one or two. As a result, some clients do not benefit from the federation at all, as their models are only based on their local data. Neither work compared the results to models to models trained only on the client's local data.

A common variant of FL is so-called cross-silo learning which is characteristic for a small number of participants, usually large organizations, and each node containing data siloed from the organization's network. For this setting FedDICE was proposed [4] and used to detect the spreading of ransomware in an integrated clinical environment. Their goal was to enable collaborative training of machine learning models between multiple hospitals. They have shown that a federated approach can achieve similar results as fully-centralized learning, even in cases of non-IID data across the clients.

Federated Learning was also proposed as an effective method of sharing cyber threat intelligence across organizations [41]. This works utilized two distinct network intrusion detection datasets, each representing an organization. The datasets vary in the attack scenarios they contain as well as the number and class balance of the samples. They evaluated two neural network models (a Feed-Forward Deep Neural Network and an LSTM). They have shown that their approach can produce a robust model capable of a high detection rate across, through which it is capable of collaborative sharing of threat intelligence.

Works focusing on network security outside of Federated Learning may also be relevant for developing new FL methods in this field. For example, the authors of [42] use anomaly detection to recognize malware attacks in the network. They employ autoencoders trained on samples of realistic normal network traffic to learn its pattern. This model can then be used to mark any deviations in the testing traffic as anomalies and potentially malicious.

We also cannot omit work done by František Střasák [43], a former student and a member of Stratosphere Laboratory [8]. In their bachelor's thesis, they focused on developing methods of detecting malware that uses HTTPS to communicate. Their methodology groups traffic to each service and extracts features from it. They used XGBoost to classify with high accuracy which connections are malicious and which benign. We use a modified variant of their feature extraction in this work.

Chapter 4

Methodology

4.1 Proposed Solution

We use horizontal cross-device federated learning for detecting malicious activity in encrypted TLS network traffic. Cross-device in this context means, that the clients represent edge computers, monitoring and capturing their traffic. It is horizontal because the clients observe the same set of features, produced by different entities. The federated approach allows to distributively train a model using the client's observations, without having direct access to the data. This enables us to protect the privacy of the data, while still being able to learn from it. In addition, each client also benefits from cooperative training, as they use a global detection model that is averaged from all model updates sent by all the clients. The global model, therefore, had access to a larger and more diverse set of data coming from all clients, possibly leading to better performance and generalization, compared to a model trained only with each client's local data.

4.1.1 Solution Architecture

Our federated learning system consists of a central aggregator and multiple clients. The aggregator is a dedicated machine that initializes and coordinates the training process. There are ten clients in our system, representing the edge computers, each containing their data in the form of processed feature vectors used for training the detection models. The data spans multiple days, and each day is treated as a separate training process. The clients split each day's data into training and validation data using an 80/20 split. The clients use the data of the next day as testing data for the current day. This is functionally equivalent to training the model on yesterday's data and evaluating it on today's data as it is coming in.

At the start of the training, every client needs to adjust its features to a common range. For this purpose, they each fit a *MinMax* scaler, which finds

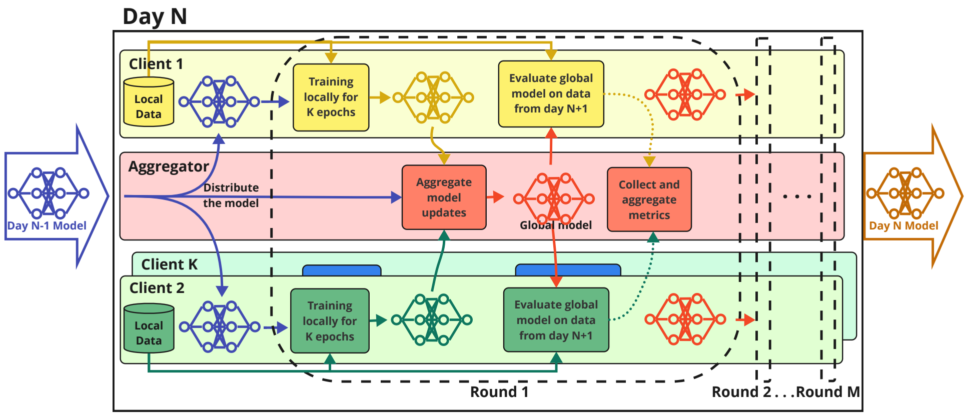

Day N
Client 1
Local Data
Training locally for K epochs
Evaluate global model on data from day N+1
Aggregator
Distribute the model
Aggregate model updates
Global model
Collect and aggregate metrics
Client K
Client 2
Local Data
Training locally for K epochs
Evaluate global model on data from day N+1
Day N-1 Model
Day N Model
Round 1
Round 2 . . . Round M

Figure 4.1: Diagram of the training process. Each day is treated as a separate training process with multiple federated training rounds. The aggregator coordinates the training process, distributes the global models, and creates new models using updates it receives from the clients. The clients distributively train the models using their local data. In our work, there are up to ten clients participating in the training.

the minimum and maximum values of every feature in the client's training data. Those extreme values are then shared with the aggregator, which combines them to produce a global MinMax scaler that is then distributed back to the clients. We choose the MinMax scaler, as it can be easily implemented in the federated setting. This scaler traditionally scales the values to 0-1 range. It is possible that some of the values transformed by the scaler may lay outside the fitted range of the scaler, as the scaler fit on the training data is also used to transform the validation and testing data. In this case, the values will be scaled outside of the 0-1 range proportionally to the values observed in the training data. This does not create issues for our work, as the used features are computed on 1-hour windows and are mostly consistent.

The training itself is initiated by the clients receiving an initial global model from the aggregator. The clients then train this model for a series of rounds. A round consists of the clients training the models locally for a number of epochs specified by the aggregator. After the local training, each client reports how the model's weights changed to the aggregator. In turn, the clients receive an updated aggregated global model, which they use in the next round. For the purposes of the experimental evaluation, the clients also evaluate the new *aggregated* global model on their testing data, which consists of the benign and malicious from the next day. They compute relevant metrics on this dataset and report them back to the aggregator. After which the next round of training starts.

The number of rounds and local epochs depends on the complexity of the models and the number of training iterations it needs to converge. Increasing the number of local epochs means that the model can be trained for the same number of epochs while decreasing the number of rounds. This effectively lowers the communication overhead. However, more local iterations may also lead to a higher risk of divergence of the models, as the clients receive the global models less often. We discuss the exact number of rounds and local epochs in the following sections describing the individual approaches.

The aggregator combines metrics using a weighted average with the number of clients' samples as weights. It also produces a global model after each round using a process described in Section 4.1.5. At the end of the day's training process, the aggregator selects the best-performing model using an aggregated validation loss of each model. This model is used as the initial model for the next day. We assume that when the model is reused, it can be trained for fewer rounds as it already possesses some domain knowledge. This can save on both computational and communication resources as well as enable the model to preserve previous knowledge.

4.1.2 Unsupervised Approach

For unsupervised learning, we use a Variational Autoencoder (VAE) to detect anomalies in the network traffic. One reason for using unsupervised learning, in this case, is that it might be difficult to obtain high-quality labels for malware traffic, which are needed for supervised learning approaches. Although our dataset is labeled, previous works have reported that unsupervised learning can be effective for detecting anomalies in network traffic data [3, 37].

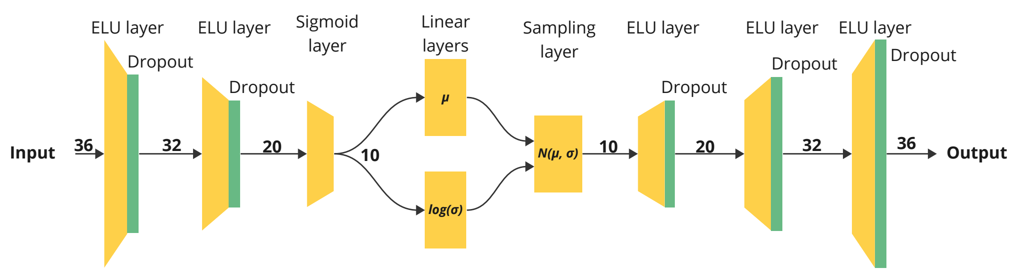
The anomaly detection model consists of an encoder and a decoder. The encoder of the model embeds the inputs into a 10-dimensional latent space. It fits multivariate normal distributions (with a diagonal covariance matrix) from the data to generate the embedded samples. The decoder then attempts to reconstruct the input vector from its compressed representation. The architecture of the model is shown in Figure 4.2a

The model is trained using a combined loss function consisting of the reconstruction loss $\mathcal{L}_{mse}$ (*Mean Square Error* of the input and output vectors), and the regularization *Kullback-Leibler* loss (KL), which penalizes the difference between the learned distribution and the standard normal distribution. The use of this penalty function was introduced together with the VAE, and ensures that the learned distributions do not diverge from each other and produce a generalizing embedding [14]. The loss function for each sample can be represented as:

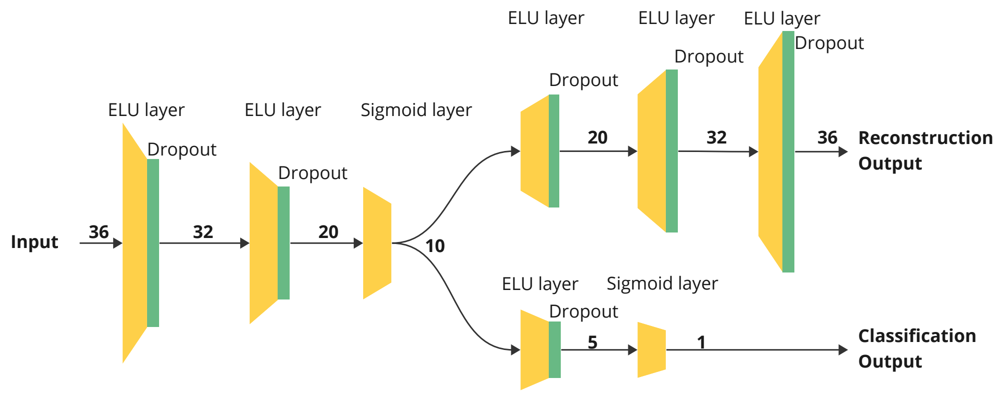
$$\mathcal{L} = \mathcal{L}_{mse}(\mathbf{x}, \hat{\mathbf{x}}) + KL(N(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x^2), N(\mathbf{0}, \mathbf{1}))$$

Where $\mathbf{x}$ and $\hat{\mathbf{x}}$ are the input and output vectors of the model, and $\boldsymbol{\mu}_x$ and $\boldsymbol{\sigma}_x^2$ are the mean and variance vectors produced by the encoder for a given input.

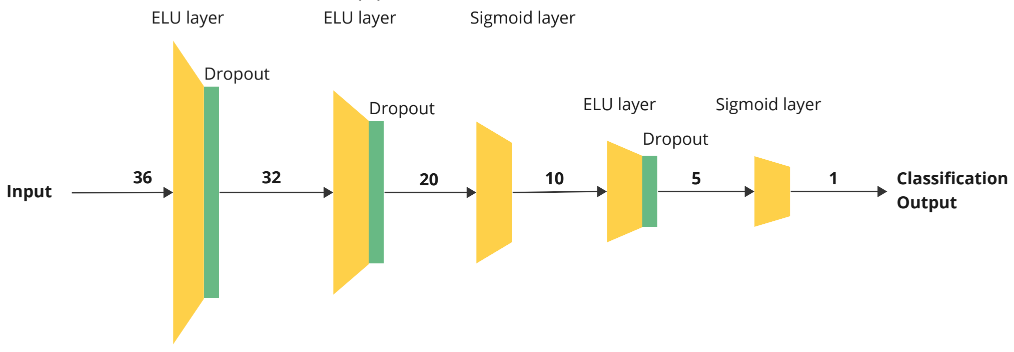
For detection, the reconstruction loss is used as an anomalous likelihood. Each client derives an anomaly threshold based on its validation data. The



(a) : Anomaly Detection Model



(b) : Multi-Head Model



(c) : Classifier-only

Figure 4.2: Architecture of the Neural Network models used in this work. The *Classifier-only* model is derived from the *Multi-Head* model by removing its reconstruction head.

reconstruction error on the normal data is often used for deriving the anomaly threshold [44, 45]. The clients compute reconstruction errors for every sample in their normal validation dataset. From those values, they found a threshold that classifies 1 % of their validation data as malicious. We use this approach in order to provide robustness to outliers in the benign validation data. If we would instead choose the maximum value on the validation dataset, it could select some non-malicious outlier of the dataset. Such a threshold would then result in more malware being missed. The specific value of 1% was chosen using an expert heuristic.

After computing individual thresholds, the clients send them back to the aggregator, which averages them and weights them by the ratio of the training data in each client to produce a global threshold. On the first day, when the model is trained from scratch, ten training rounds are used, and five when the model from the previous day is reused. Clients train the model locally for one epoch in the first two rounds and for two epochs in the remaining rounds. While more local rounds are more communication efficient, as discussed in Section 2.6.4, it may also lead to divergence in the individual client's updates. When using the momentum-based methods, the largest steps in the parameter space are generally made in the first few epochs [13]. We hope that by aggregating after each epoch in the first two rounds, the global model manages to converge into a state from which it reliably converges with less frequent aggregations.

4.1.3 Supervised Approach

For supervised learning, we use two types of models, a *Multi-Head* model and a *Classifier-only* model. They are shown in Figure 4.2b and Figure 4.2c, respectively. The *Multi-Head* model is derived from a regular autoencoder by adding a classification head after the encoder while also keeping the decoder part. We hope that by keeping the autoencoder components, the clients with only benign samples can contribute to the learning process by improving the embedding space. The model is trained to distinguish malicious traffic from benign; malicious being the positive class. Only benign samples are passed to the decoder part of the network so that the network does not learn to reconstruct the malicious samples well.

The model is trained on a combined classification - *Binary cross entropy* ($\mathcal{L}_{BNE}$) - and reconstruction loss - *Mean Square Error* ($\mathcal{L}_{MSE}$).

$$\mathcal{L} = \mathcal{L}_{BNE} + \mathcal{L}_{MSE} = (\mathbf{1} - \mathbf{y})log(\mathbf{1} - \hat{\mathbf{y}}) - \mathbf{y}log(\hat{\mathbf{y}}) + \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

The *Classifier-only* model was created to evaluate if the decoder part of the *Multi-Head* model brings any benefits. Its structure is identical to the *Multi-Head* model, but with the reconstruction part of the network missing. This effectively turns it into a regular classification model and is trained using only the *Binary cross entropy.*

The supervised models are trained for 75 rounds on the first day when the models are trained from scratch. On the following days, when the model from the previous day is reused, 25 training rounds are used. Clients train the model locally for one epoch every round.

Malware Vaccine

The motivation behind the *Multi-Head* model is to allow all clients to participate in supervised training, even if they do not have any malware data. However, it was observed that having clients participating without positive samples makes the supervised federated model unable to converge. To address this, we decided to send each client a set of malicious feature vectors, which we call a "vaccine", to help with the convergence. Traditionally in the security field, vaccines are a harmless part of the malware that is injected into the host machines to prevent infections [6]. Our vaccines differ in that they are not a passive mechanism but an aid in the learning process and a way to tackle data heterogeneity, as suggested in [46, Chapter 3]. The vaccines are comprised of only numerical values and, as such, do not pose any risk to the clients. In our setting, the central aggregator is responsible for gathering and distributing this set of data to the clients. This approach could be achieved in real deployments by using samples of malicious network traffic from publicly available datasets. In order to mitigate the convergence issues, the "vaccine" dataset has to be sufficiently large (more than 70 feature vectors).

4.1.4 Assumptions and Limitations

In this subsection, we discuss the assumptions and limitations of our work.

- **Supervised learning assumption:** In the supervised setting, we assume that the clients have the capability to label the data locally. This is a relatively strong assumption in real deployments, but our work aims at developing methods that would only require this from a subset of the participants.

- **Unsupervised learning assumption:** For the unsupervised setting, we assume that there are no malicious samples in the benign dataset. While we are confident that this assumption holds in this work, as the dataset was created using expert knowledge, it may be challenging to assure in future work or in real-world deployments.

- **Client trust:** We also assume that the clients who connect are not malicious and try to damage the training process or extract knowledge from other clients.

- **Client availability:** One limitation of our work is that we do not handle cases where some clients drop out or are unavailable during the training process. Although this is quite common in real-world settings, we believe that with the limited size of the dataset, we would not be able to evaluate this well.

Overall, our work has several assumptions and limitations that should be considered when interpreting the results and implications. These limitations do not invalidate our findings, but they should be taken into account when considering the generalizability and applicability of our approach.

4.1.5 Learning Algorithm

To train our models, we use a combination of FedAdam and FedProx algorithms. SGD with FedProx regularization is used to train the models in the clients. FedProx adds a term to the loss function penalizing the clients' divergence from the last received global model. This mitigates divergence in case of statistical differences between the clients' datasets. This client-side regularization is important when training with a small batch size (making a larger number of local steps) or when training locally for multiple epochs before sending the weight updates to the aggregator.

On the server side, the clients' contributions are aggregated using a weighted average based on the amount of clients' training data. Using this aggregate, a new update to the global model is created using the FedAdam algorithm provided in the flower framework. It is a federated variant of the Adam optimizer, and as such, it uses momentum when aggregating the client updates providing better convergence on the heterogenic data. These learning algorithms are described in more detail in Section 2.6.3.

4.2 Implementation

We chose to use a Python-based open-source federated learning framework called Flower [47] to implement our methods. Flower is a versatile framework that provides extendable implementations of both the server (aggregator) and the clients. It is designed to handle the communication between the aggregator and clients, enabling us to focus on developing the methods. In addition, Flower includes implementations of some of the common algorithms for aggregating client updates.

In the Flower framework the user is responsible for implementing the functionality, such as loading the local dataset, orchestrating the local fitting, and computing the metrics. To train the models in the clients, we have used TensorFlow [48] in combination with Keras [49]. On the server side, the aggregation of metrics must be implemented, as well as the initialization of the model and setting of training parameters. Flower allows for the sending of serializable data structures, which can be useful for exchanging configurations or other information between aggregator and clients. The serializable data has been used to distribute the vaccines from the aggregator to the clients.

The code for this work can be found in the repository of the FEEL project: `https://github.com/stratosphereips/feel_project`. This repository contains the implementation of the described methodology, including the code for orchestrating and running experiments and analyzing their results. It also includes the preprocessing of raw data into hourly feature vectors used by the models. The feature extraction is based on an in part reused from the work done by František Střasák in [43].

4.3 Experiment Setup

This section describes how the proposed methods are evaluated, how they are compared, and how the metrics are collected.

For the purposes of this work, one experiment is a set of conditions and parameters which are evaluated. Each experiment consists of ten federated runs with identical parameters, differing only in the random seed used for initializing the model and splitting the local datasets for training and validation. Each run in an experiment performs a *federated* run, a *local* run, and a *centralized* run; all using the same parameters and random seed.

On each run, the models are trained for a total of four days, producing a global detection model each day. These models are then evaluated on the *next* day's data resulting in four sets of evaluation metrics from each of the runs. Only four days are evaluated because the dataset has five days and the last day can not be evaluated since there is no next day.

4.3.1 Federated Training Process

The dataset on which our solution is evaluated has network traffic from five consecutive days for ten distinct clients. Only five clients have labeled malicious traffic which can be used for supervised training, the rest five clients only have benign traffic.

On each day a federated training process is done. The diagram in Figure 4.1 illustrates the federated training process. On the first day of training, the model is initialized randomly and trained from scratch on each client's data, while on the following days (days 2, 3 and 4), the previous day's model is **reused**. On a given day, the clients train using data from that day, split into training and validation data (using an 80/20 split). The validation dataset is used for selecting the best-performing global model (based on an aggregated validation loss). The testing is done on the next day's traffic, which is functionally equivalent to using the previous day's model for detection.

The model from the previous day is used, so that gained knowledge from the past is preserved while also not requiring the clients to keep data for longer periods of time. In general, when training the model from scratch, more rounds of training are necessary than when adjusting an already existing model to newer data.

4.3.2 Metrics

After each round, the clients evaluate the received global model on their test dataset by computing a set of metrics. Each of these values is aggregated on the server using a weighted average, where the weights are relative ratios of the sizes of clients' data used for generating the metrics. Meaning, that in the case of testing metrics, the sizes of the test datasets were used, with the vaccine samples included. The motivation for this is to produce similar metrics as if they were computed on a complete dataset from all clients.

To evaluate our methods, the metrics used are *Accuracy* (*Acc*), *True Positive Rate* (*TPR*), *False Positive Rate* (*FPR*) shown in eqs. (2.1) to (2.3) and *F-score* shown in eq. (2.5). *Accuracy* is a standard metric used to evaluate classification and detection models. However, it can not capture all the relevant information on its own. *TPR* indicates what ratio of the malicious samples was detected, and *FPR* shows how much of the benign samples are misclassified as malicious. The *F-score* is often advocated as a summarizing metric when comparing the performance of two classifiers [9, Chapter 11] [11]. We use its unweighted variant *F1*.

4.3.3 Comparison to Other Settings

All experiments are repeated ten times with a different random seed for initializing the model and splitting the dataset. Each run of the federated experiment is accompanied by training the same model with equivalent parameters in a local and centralized setting.

- **Local setting:** The local setting mimics the scenario when the client decides not to participate in the federated learning and instead creates a model using only its data. Comparing this to the federated results should show the benefits of joining the federated process. When evaluating the local setting, we use the datasets of all clients for the following day. This is to demonstrate how well the locally trained models would perform in other clients or when encountering unknown threats. The reported results are averages of the performance of the individual models.
- **Centralized setting:** The centralized setting represents a case where there would be no restrictions on the privacy of the data so that we could collect all datasets of the clients into a single one and use that for training the models. This should provide an ideal scenario for the model's performance.

Chapter 5

Dataset

For the evaluation of the proposed solution, we have considered existing datasets used by researchers studying network security. However, Federated Learning requires that the used dataset can be split in a specific manner to represent the local datasets of the individual participants of the federated process. Therefore a dataset for Federated Learning should include several different clients and possibly different days. Moreover, the split between classes should be realistic in the sense that its parts should be non-IID, meaning they should have different sizes, class balances, and the statistical distribution of the feature differ. Although some federated datasets in network security exist, they are mostly based on IoT malware, which is not complex and relatively easy to detect. IoT malware also usually does not use HTTPS traffic. We therefore decided to use the *CTU-50-FEEL* dataset[7] created for this thesis. CTU-50-FEEL is a dataset that contains very specific HTTPS features based on the more generic dataset CTU-50 created by the Stratosphere Laboratory [8].

CTU-50-FEEL consists of aggregated features for ten clients that generate traffic for five days. All the clients produce benign traffic, but only five of them also produce malware traffic mixed with the benign. The dataset also contains a sixth malware for testing. The following sections describe the process of creation, processing and subsequent feature extraction.

In particular for this thesis, the CTU-50-FEEL dataset was further modified to create a dataset variation called *CTU-50-FEEL-less-malware*. This variant contains much *less* malware per client, and less clients with malware, and therefore is much harder to detect. This was done to further test our supervised methods. More details about the differences between the variants is shown in Section 5.4.

5.1 Benign Traffic

The CTU-50-FEEL dataset has the traffic of 10 real human users (no simulations) over five consecutive days. The original format of the flows uses the Zeek logs to form the dataset [50]. Zeek is an open-source tool for monitoring and analyzing network traffic. It saves the network events into log files based on their type. The *conn.log* contains records about each connection, such as the used protocol, the originator, and the responder, as well as a unique identifier of the connection, which can be used to associate it with other types of logs. The *ssl.log* and *x509.log* log files are also relevant for this work, as they contain information about HTTPS connections and the certificate used to establish the encrypted connection. Zeek can generate other types of logs for different types of traffic or protocols, but those that are not used for feature extraction. All flows in the log files were labeled using the Stratosphere Lab's tool *netflowlabeler* [51].

Table 5.1 shows the number of TLS flows per client for each day. The traffic comes from real users, which are active for different periods each day and are using different operating systems and sets of applications. This results in realistic traffic where there are significant differences in the client data. The 10 real users used Linux, Windows and macOS operating systems.

The benign dataset was created by capturing the traffic of users in Stratosphere Laboratory [8]. The traffic was collected with their consent and for the purposes of security research. The CTU-50-FEEL dataset only consists of processed aggregated numerical features and does not contain any Zeek flow data or identifiable information.

Client	Day 1	Day 2	Day 3	Day4	Day 5
1	545	611	596	631	602
2	3,174	1,039	868	1,870	2,543
3	2,817	1,413	1,091	703	1,227
4	998	812	595	995	635
5	1,057	365	497	1,006	795
6	3,060	1,900	2,287	2,666	2,101
7	1,479	887	1,000	906	866
8	2,145	2,522	2,928	2,967	3,396
9	2,541	2,539	2,349	2,217	1,874
10	4,694	4,823	5,287	4,126	5,976

Table 5.1: Number of client TLS flows per day for benign traffic data

5.2 Malware Traffic

The malicious traffic comes from the deployment of real malware in the Stratosphere Laboratory [8]. All malware use TLS for command and control communication. As this traffic is encrypted, it is more difficult to distinguish it from normal traffic. Table 5.2 shows the number of TLS flows of each malware on individual days. The number of malware flows is much lower than the benign traffic, and two of them have TLS activity only on the first day.

Malware	Designation	Day 1	Day 2	Day 3	Day 4	Day 5
Dridex.A	M1	295	302	301	302	304
Trickbot	M2	88	89	88	89	87
NjRAT	M3	13	0	0	0	0
Cridex	M4	780	823	775	855	793
Kovter.B	M5	744	0	0	0	0
Dridex.B	M6	822	908	910	920	900

Table 5.2: Number of TLS flows per day in each of the malware datasets

5.3 Feature Extraction

To train neural network models on the data, we need to extract numerical features from the data. For that, we took advantage of the work done by František Strašák in [43]. The methodology used allows us to extract useful and proven features from TLS data to detect threats. However, in our work, we have decided to aggregate the traffic in one-hour windows, instead of per-day. This enables faster detection of possible threats, as the feature vectors and subsequent detection can be done within an hour of capturing the traffic.

Within the one-hour window, connections are aggregated based on a 4-tuple of source IP address, destination IP address, destination port, and protocol. This 4-tuple allows to group all related connections to one specific service. The flows inside the 4-tuple share a unique purpose and behavior. These values, along with information about the amount of data transferred and some temporal aspects of the connection, are extracted from the conn.log files. Features relating to the TLS traffic are computed from the values contained in ssl.log files using the certificate information located in the x509.log filse. The complete list of features can be seen in Table 5.3. Features mentioned as *not used* in the table were omitted because of low variance in their values. Both benign and malware data were processed in this manner. The processed dataset without the identifying information can be found at: `https://github.com/stratosphereips/feel_data`

Feature	Description
num_flows	Number of records in 4-tuples
avg_dur	Mean duration of connections
std_dev_dur	Stdev of connection duration
percent_stdev_dur	Percent of connections exceeding stddev
total_size_of_flows_orig	Bytes sent by the originator
total_size_of_flows_resp	Bytes sent by the responder
ratio_of_sizes	Ratio of responder bytes
percent_of_established_states	Percent established connections
inbound_pckts	Number of incoming packets
outbound_pckts	Number of outgoing packets
periodicity_avg	Mean periodicity of connection
periodicity_stdev	Stdev of the periodicity of connection
ssl_ratio	Not used
average_public_key	Mean public key length
tls_version_ratio	Ratio of records with TLS
avg_cert_length	Mean certificate length
stdev_cert_length	Stdev certifcate length
is_valid_certificate_during_capture	1 if certificate is valid
amount_diff_certificates	Number of certificates
num_domains_in_cert	Number of domains in certificates
cert_ratio	Ratio of valid certificates
num_certificate_path	Number of signed paths
x509_ssl_ratio	Ratio of SSL logs with x509
SNI_ssl_ratio	Ratio with SNI in SSL record
self_signed_ratio	Not used
is_SNIs_in_SNA_dns	Check if SNI is in the SAN DNS
SNI_equal_DstIP	Not used
is_CNs_in_SNA_dns	1 if all common names are in SAN
ratio_of_differ_SNI_in_ssl_log	Ratio SSL with different SNI
ratio_of_differ_subject_in_ssl_log	Ratio SSL with different subject
ratio_of_differ_issuer_in_ssl_log	Ratio SSL with different issuer
ratio_of_differ_subject_in_cert	Ratio of subjects
ratio_of_differ_issuer_in_cert	Ratio of issuers
ratio_of_differ_sandns_in_cert	Ratio of SAN DNS
ratio_of_same_subjects	Ratio SSL with same subject
ratio_of_same_issuer	Ratio SSL with same issuer
ratio_is_same_CN_and_SNI	Checks if CN is same as SNI
avg_certificate_exponent	Mean exponent value
is_SNI_in_top_level_domain	1 if SNI is a top level domain
ratio_certificate_path_error	Not used
ratio_missing_cert_in_cert_path	Not used

Table 5.3: Extracted features for a TLS connection, or 4-tuple. A TLS connection aggregates all the TLS flows that share a source IP, destination IP, destination port, and protocol.

5.4 Dataset Mixing

For the purpose of this work, we needed to mix the malware and benign traffic to form a supervised dataset. For the CTU-50-FEEL dataset, we have achieved that by assigning a different malware to each the first six clients. Two of the malware only had some HTTPS activity on the first day. This mixing was designed to fully utilize all captured malware traffic. The total number of feature vectors for each client can be seen in Table 5.5.

In Section 4.1.3 we described the malware vaccine which is a set feature vectors from the malware dataset, which is send to the clients to improve the convergence of the supervised federated methods. On each day we dedicated one day of a malware datasets to be used as a vaccine. Which malware dataset is used as a vaccine for a given day is shown in Table 5.4. When the clients receive the vaccines, they incorporate them into their local dataset and use them to train the supervised models.

	Day 1	Day 2	Day 3	Day 4
Vaccine malware	M4-1	M2-2	M1-3	M6-4

Table 5.4: Malware datasets used as a vaccine in the supervised experiments.

We also designed a scenario that enables us to assess how the proposed methods cope with more complicated setups. We have named this variant of the dataset *CTU-50-FEEL-less-malware*. Its benign data is identical to the CTU-50-FEEL, but we have removed malware infection on some of the days. The overview of the malware left in the CTU-50-FEEL-less-malware in each client is shown in Table 5.6. Note that the syntax M<malware number>-<day> references the malware captured by a client in the complete mix on a particular day. The table also indicates which vaccine was used each day.

Decreasing the number of malware present in the clients leads to increased class imbalance and amplifies the overall non-IID properties of the clients' datasets. While in the CTU-50-FEEL, around half of the clients did have some malware traffic on each day, in this variant, the number of infected clients at a given time is even smaller. By designing this more challenging scenario, we hope to demonstrate the capabilities of the developed methods to handle the following challenges:

- Challenge one: The situation where on the first day only the "vaccine" malware is present, with some malware appearing on the second day in some of the clients. As the models are evaluated on the next day's data, it test how well it is able to generalize from one type of malware to others, as well as how well the models can adapt to new malware appearing.

		Day 1		Day 2		Day3		Day 4		Day 5	
Client	Malware	Benign	Malware	Benign	Malware	Benign	Malware	Benign	Malware	Benign	Malware
1	M1	346	48	356	48	361	48	396	48	395	48
2	M2	748	72	468	72	430	72	600	72	902	72
3	M3	598	4	297	0	281	0	380	0	568	0
4	M4	417	108	337	105	264	109	402	105	255	106
5	M5	341	29	223	0	216	0	371	0	454	0
6	M6	1,115	96	943	96	891	96	1,042	96	805	96
7	—	415	0	334	0	361	0	344	0	312	0
8	—	1,203	0	1,315	0	1,470	0	1,515	0	1,593	0
9	—	1,180	0	1,283	0	1,147	0	1,044	0	783	0
10	—	1,213	0	1,263	0	1,338	0	1,410	0	1,466	0

Table 5.5: Number of benign and malicious samples in each client in the raw CTU-50-FEEL on a particular day. The malware indications reference malware in Table 5.2

- Challenge two: The situation where on the third and fourth days, malware that was previously used as a vaccine reappears after not being present on the previous day. This evaluates how well the knowledge is preserved after retraining on a different set of threats.
- Challenge three: The situation where the last day contains all available malware to test the final model on as much malicious data as possible.

	Day 1		**Day 2**		**Day 3**		**Day 4**		**Day 5**	
Client	**Vaccine Malware**	**Client Malware**	**Vaccine Malware**	**Client Malware**	**Vaccine Malware**	**Client Malware**	**Vaccine Malware**	**Client Malware**	**Vaccine Malware**	**Client Malware**
1	M4-1	—	M2-2	M1-2	M1-3	—	M6-4	M1-4	—	M1-5
2	M4-1	—	M2-2	—	M1-3	—	M6-4	M2-4	—	M2-5
3	M4-1	—	M2-2	—	M1-3	—	M6-4	—	—	M3-1
4	M4-1	—	M2-2	—	M1-3	M4-3	M6-4	M4-4	—	M4-5
5	M4-1	—	M2-2	M5-1	M1-3	—	M6-4	—	—	—
6	M4-1	—	M2-2	—	M1-3	M6-3	M6-4	—	—	M6-5
7	M4-1	—	M2-2	—	M1-3	—	M6-4	—	—	—
8	M4-1	—	M2-2	—	M1-3	—	M6-4	—	—	—
9	M4-1	—	M2-2	—	M1-3	—	M6-4	—	—	—
10	M4-1	—	M2-2	—	M1-3	—	M6-4	—	—	—

Table 5.6: The used malware in the *CTU-50-FEEL-less-malware* dataset for each client. The *M<malware number>-<day>* references a particular day of malware from the malicious data of CTU-50-FEEL, shown in Table 5.2. Benign data is identical to CTU-50-FEEL, as shown in the Table 5.5.

Chapter 6

Experiments

This chapter describes the experiments conducted to evaluate the methods proposed in Chapter 4 using the dataset described in Chapter 5. We provide a description of the parameters used when running the experiments and an the results.

6.1 Unsupervised Experiments

Recall that the unsupervised experiments are designed to measure how well FL can be used for clients without labels for their traffic. In these experiments, we use the Autoencoder architecture described in Section 4.1.2 for anomaly detection. Each client trains the model using its benign local data of a particular day according to the CTU-50-FEEL dataset. The data for that day is split into training and validation datasets using an 80-20 split (80% for training).

The validation data is used for selecting the anomalous threshold. Each client selects as threshold a 99 percentile of their validation losses, i.e. value, which will mark 1 % of their validation samples as anomalous. The thresholds on each client are then weighted on the aggregator based on the number of training samples in the clients and finally averaged to produce the global detection threshold. The aggregator uses the combined weighted validation loss to selecting the best-performing model. By default, the model is saved and reused on the next day's training.

If the model is trained from scratch, it is trained for ten rounds. If the model from the previous day is re-used, then it is only trained for five rounds. The model is trained at the clients for one epoch in the first two rounds and for two epochs for the remaining rounds. The batch size used is 64. Both global and local learning rates are set to 10^{-3} and the first and second momentums of FedAdam are set to 0.9 and 0.99 respectively. The μ parameter of the proximal term is set to 10.

One run of the experiment consists of training on the first four days and always testing on the next day's dataset. Each run is repeated ten times with a different random seed, with the values reported in this section corresponding to the mean values across the runs. Each run of the experiment is also accompanied by a centralized and local run.

6.1.1 Experiment A1: Reusing the model on the next day

This type of experiment reuses the previous day's model on each next day. This allows for fewer training rounds on the subsequent days as well as preserving knowledge from across the days, as the model is effectively exposed to a larger amount of data.

Figure 6.1a shows results of these type of experiments. When the model is trained in the federated setting, it outperforms the models trained locally in almost every metric, while the centralized models achieve better results than both of them.

Although on some days the local models detect marginally more malicious samples, their average False Positive Rate is almost one and a half that of the federated one. This results in significantly better accuracy and F1 for the federated model.

Although it is expected that a centralized model would perform better due to access to all data simultaneously, these results show that clients using the federated model would anyway greatly benefit from participating in the federated learning. This shows that federated learning can be a valuable alternative when direct data sharing is not feasible or desired.

6.1.2 Experiment A2: New model on each day

In contrast to experiments A1, it is important to evaluate the effects of training a new model completely each day. This requires training for the whole ten rounds each day and therefore can mitigate the degradation of the model.

The summarized results in Table 6.1 show that training a model from scratch every day marginally but marginally improves the performance. This may hint that the performance of the reused model is degrading, which seems to be the case for testing days 3 and 4 according to the results in Appendix A.1. However, on the final day, the A1 model outperforms the A2 model. An explanation for this observation might be a change in the distribution of the clients' benign data between day 4 (the training day) and day 5 (testing day). Table 5.5 shows the differences in amounts of benign data between these days. Possibly the A1 model may be more resilient after multiple rounds of fine-tuning.

		Cent.	Fed.	Local
A1	$Acc \pm std$	96.65 ± 0.31	96.03 ± 0.47	95.24 ± 0.66
	$TPR \pm std$	46.14 ± 2.24	45.46 ± 0.89	45.12 ± 0.70
	$FPR \pm std$	1.00 ± 0.31	1.66 ± 0.45	2.46 ± 0.68
	$F1 \pm std$	55.14 ± 2.49	50.09 ± 2.83	45.56 ± 3.31
A2:	$Acc \pm std$	96.65 ± 0.32	96.07 ± 0.43	95.60 ± 0.67
	$TPR \pm std$	46.12 ± 2.25	45.29 ± 0.60	45.29 ± 0.71
	$FPR \pm std$	0.99 ± 0.32	1.61 ± 0.43	2.09 ± 0.71
	$F1 \pm std$	55.18 ± 2.53	50.22 ± 2.60	47.63 ± 3.28
A3: 6 clients	$Acc \pm std$	96.58 ± 0.40	94.78 ± 1.10	94.97 ± 0.82
	$TPR \pm std$	46.11 ± 2.21	45.52 ± 0.61	45.32 ± 0.69
	$FPR \pm std$	1.06 ± 0.44	2.98 ± 1.09	2.76 ± 0.80
	$F1 \pm std$	54.71 ± 2.94	43.70 ± 4.41	44.35 ± 3.37
A3: 3 clients	$Acc \pm std$	96.31 ± 0.68	94.16 ± 1.13	93.37 ± 2.13
	$TPR \pm std$	46.05 ± 2.14	45.35 ± 0.59	44.91 ± 0.37
	$FPR \pm std$	1.34 ± 0.68	3.62 ± 1.14	4.42 ± 2.14
	$F1 \pm std$	53.01 ± 4.28	40.82 ± 4.25	38.57 ± 6.66

Table 6.1: Summary of the anomaly detection experiments. The provided values are averages over the four testing days with ten runs each.

6.1.3 Experiments A3: Effect of fewer participants

In previous type of experiments A1 and A2 all ten clients always participated in the training. In this type of experiment, we evaluate the performance of models when only training on six and three of the original clients, while still evaluating using the complete next day's dataset.

As expected, with the decreased number of clients used for training, the obtained models perform worse. However, the gap between the centralized and federated models increases significantly with the decreased number of participants. Analogically, this could mean that with more clients, the performance gap between centralized and federated could shrink.

6.2 Supervised Experiments

Recall that the supervised experiments are designed to use the knowledge and labels in the clients, independently of how these labels were obtained. In the supervised setting, some clients have malicious samples in addition to benign samples. The distribution of the malware dataset among the clients is described in Section 5.4. Most of the parameters of the training remained unchanged from the unsupervised experiments, except of the number of training rounds which we increased to 75 if training the model from scratch and to 25 if reusing the model from the previous day. In this set of experiments, each client trains the models for one epoch each round. The vaccine used for the supervised experiments is described in Section 5.4.

Two types of architectures are evaluated - the *Multi-Head* model with both a classification head and a decoder head and a *Classifier-only* model. As with unsupervised cases, each experiment is run in a federated, centralized, and global setting and is repeated ten times with a different random seed.

6.2.1 Experiment S1: CTU-50-FEEL dataset and reusing the model

This scenario represents the base case for the supervised experiments. The benign and malicious datasets of the CTU-50-FEEL are split amongst the 10 clients. In this scenario, every client is used for both training and evaluation.

The summarized results are shown in Table 6.2, and the breakdown of the performance on individual days can be seen in Figure 6.1. The figure also compares the performance of supervised models to that of the anomaly detection model. As expected, the supervised models achieve much better results, and even the locally trained supervised models outperform centralized AD models. Most of the improvements come from improved recall (TPR). Previously it hovered below 50%, but the supervised models reached nearly 100% recall meaning almost all malware samples are detected.

The two variants of the supervised models reach comparable results, with the *Classifier-only* model achieving marginally better average accuracy and F1 metrics. However, the performance margin is minor and does not provide enough evidence to claim the superiority of one of the model variances.

Compared to the unsupervised models, both classifying variants come closer to the performance of the centralized model event outperforming it in some metrics. However, the relative gap between local and federated settings increased compared to the AD model. This is likely caused by some of the clients only containing benign data, and all malicious samples used in training originate from the vaccine.

6.2.2 Experiment S2: Only training the model using clients with malicious data

In experiments of type S1 all clients are used for federated training of the model even if the client does not possess any malicious samples. In order to evaluate if their inclusion brings any benefit to the final model, the S2 experiment is designed in which clients non-infected clients are excluded from training and only used for evaluation. Specifically, Clients 1-6 are used to

		Cent.	Fed.	Local
S1 *Multi-Head* model	$Acc \pm std$	99.76 ± 0.07	99.61 ± 0.17	98.29 ± 0.78
	$TPR \pm std$	99.10 ± 0.81	99.78 ± 1.37	84.94 ± 9.23
	$FPR \pm std$	0.21 ± 0.06	0.39 ± 0.18	1.08 ± 0.55
	$F1 \pm std$	97.37 ± 0.78	94.96 ± 2.13	81.72 ± 8.13
S1 *Classifier-only* model	$Acc \pm std$	99.77 ± 0.07	99.62 ± 0.16	98.11 ± 0.81
	$TPR \pm std$	99.05 ± 0.71	99.77 ± 1.37	83.02 ± 9.89
	$FPR \pm std$	0.20 ± 0.06	0.38 ± 0.17	1.19 ± 0.59
	$F1 \pm std$	97.44 ± 0.75	95.06 ± 1.98	79.76 ± 8.41
S2 *Multi-Head* model	$Acc \pm std$	99.78 ± 0.08	99.62 ± 0.15	98.08 ± 1.28
	$TPR \pm std$	99.73 ± 0.54	100.00 ± 0.00	90.72 ± 8.58
	$FPR \pm std$	0.22 ± 0.08	0.40 ± 0.15	1.56 ± 1.10
	$F1 \pm std$	97.57 ± 0.78	95.03 ± 1.72	81.66 ± 11.05
S2 *Classifier-only* model	$Acc \pm std$	99.79 ± 0.08	99.61 ± 0.16	98.37 ± 0.97
	$TPR \pm std$	99.70 ± 0.53	99.92 ± 0.49	91.76 ± 8.40
	$FPR \pm std$	0.21 ± 0.07	0.40 ± 0.17	1.31 ± 0.76
	$F1 \pm std$	97.67 ± 0.83	94.98 ± 1.97	83.81 ± 9.12
S3 *Multi-Head* model	$Acc \pm std$	99.76 ± 0.06	99.60 ± 0.21	97.60 ± 0.47
	$TPR \pm std$	99.42 ± 0.71	100.00 ± 0.00	64.87 ± 11.15
	$FPR \pm std$	0.22 ± 0.06	0.41 ± 0.22	0.88 ± 0.57
	$F1 \pm std$	97.37 ± 0.61	94.85 ± 2.60	70.39 ± 5.86
S3 *Classifier-only* model	$Acc \pm std$	99.77 ± 0.06	99.19 ± 1.10	97.67 ± 0.54
	$TPR \pm std$	99.32 ± 0.73	87.50 ± 33.49	64.40 ± 13.78
	$FPR \pm std$	0.21 ± 0.05	0.38 ± 0.25	0.79 ± 0.64
	$F1 \pm std$	97.41 ± 0.65	82.79 ± 31.78	70.60 ± 7.13

Table 6.2: Summary of the S1, S2, and S3 experiments all conducted on the CTU-50-FEEL dataset. The provided values are averages over the 4 testing days.

train on the Day 1 and Clients 1, 2, 4, 6 on the following days, as clients 3, and 5 only have malware data on the first day.

The summary results of the S2 experiments are shown in Table 6.2. Although the number of training clients decreased, the models managed to preserve a similar performance to those of S1. In the federated setting, the only observable difference is a statistically insignificant improvement in the *Multi-Head* models's F1 and a similarly inconclusive decrease of the same metric of the *Classifier-only* model.

A more pronounced difference is in the TPR of the local models, which improved when trained only on the clients with their own malware. This supports the claim from the previous subsection that when training locally, using malicious data only from the vaccine results in worse performance.

6.2.3 Experiment S3: New model on each day

In the S1 and S2 scenarios, we save the model at the end of training to be reused on the next day. This saves computational and communication resources since the reused models can be trained for fewer epochs. It might also enable the model to remember threats it saw on previous days and transfer this knowledge into the future.

However, there is also a risk of gradual degradation of the model and worse adaptability to shifting distribution of data. In this type of experiments, we train a new model every day to measure the impacts of these potential risks.

The summary of results in Table 6.2 shows that while the performance of the *Multi-Head* S3 model is comparable to the S1 type, the S3 *Classifier-only* models's average TPR is much lower with higher variance. The detailed results in Table A.5 show that on the first testing day the model performs the same as the other models, and only on the following days it degrade. Since the first training day contains the most malicious samples, it suggests that the *Classifier-only* model needs more positive samples in the clients.

The similar performance of the *Multi-Head* model in the S1 and S3 experiments suggests that reusing the model does not lead to degraded performance over time while requiring fewer training rounds.

6.2.4 Experiment S4: Using the CTU-50-FEEL-less-malware dataset

Section 5.4 describes a second type of malware mix which we have created in order to test our proposed models. *CTU-50-FEEL-less-malware* is designed to be both more challenging and more realistic by including less positive class samples. None of the clients have any malware data on the first day and have to rely on the vaccine provided by the aggregator to train the classification model. In the following days, only two to three clients have their own malware samples, and the final day, which is only used for evaluation, contains as much malware as possible. As the amount of malware and its types change every day, it should prove to be much more challenging for the model and its ability to adapt.

Figure 6.2a presenting the results of this experiment shows that this scenario is much more challenging for the supervised methods. Although the overall accuracy is still higher than in the unsupervised experiments, there are measurable differences between the two classification models.

Day 2 results are produced by models trained only on the first day when no client has any malware data except the vaccine. However, on the second day on which the model is tested, two different malware are present in the client's dataset, which the model struggle to detect. The third day contains the same malware as the first day's vaccine (M4) and the traffic of M6 malware, which is the same type as the M1 present on the second day. This results in overall better performance on Day 3.

On the following day, the M2 appears in Client 2 while only being used as a vaccine on Day 2 previously. However, it seems that the model was not able to preserve its knowledge well since the true positive rate decreased on that day. The final day is only used for testing and contains a larger amount of malware samples. Despite that, the models perform best on it with an average F1 score above 90%.

This more challenging data scenario shows that with small amounts of positive samples, the models might struggle to converge to a solution that is able to detect the malware. When the models are trained with only the benign data and the vaccine, it occasionally resulted in models which did not mark any of the test samples as malicious. This seems to affect the *Multi-Head* model more than the *Classifier-only* model. Although on the next day, all of the *Classifier-only* models mostly recovered, some of the *Multi-Head* ones still performed significantly worse. This effect seems to disappear in the following days.

6.2.5 Experiment S3+S4: New model on each day, CTU-50-FEEL-less-malware dataset

The *CTU-50-FEEL-less-malware* dataset variant also provides a good setting to test, how well the model preserves information from previous days. In the scenario, there are instances where a dataset is used for training on one day, is not present on the next day, and reappears on one of the following ones. In order to evaluate if the reuse of the model brings any benefits in these cases, we combined the S4 data scenario and S3 in which the model is trained from scratch each day.

The results are shown in Figure 6.2b. Day 2's results are not really relevant as the setting is functionally identical to the S4 experiment. The differences in the results on this are given by some of the models failing to converge to a solution that is able to detect any malware. Similar issues can be observed on Day 3. It seems that reusing the model can mitigate it, as all of the S4 models are detecting at least some malware.

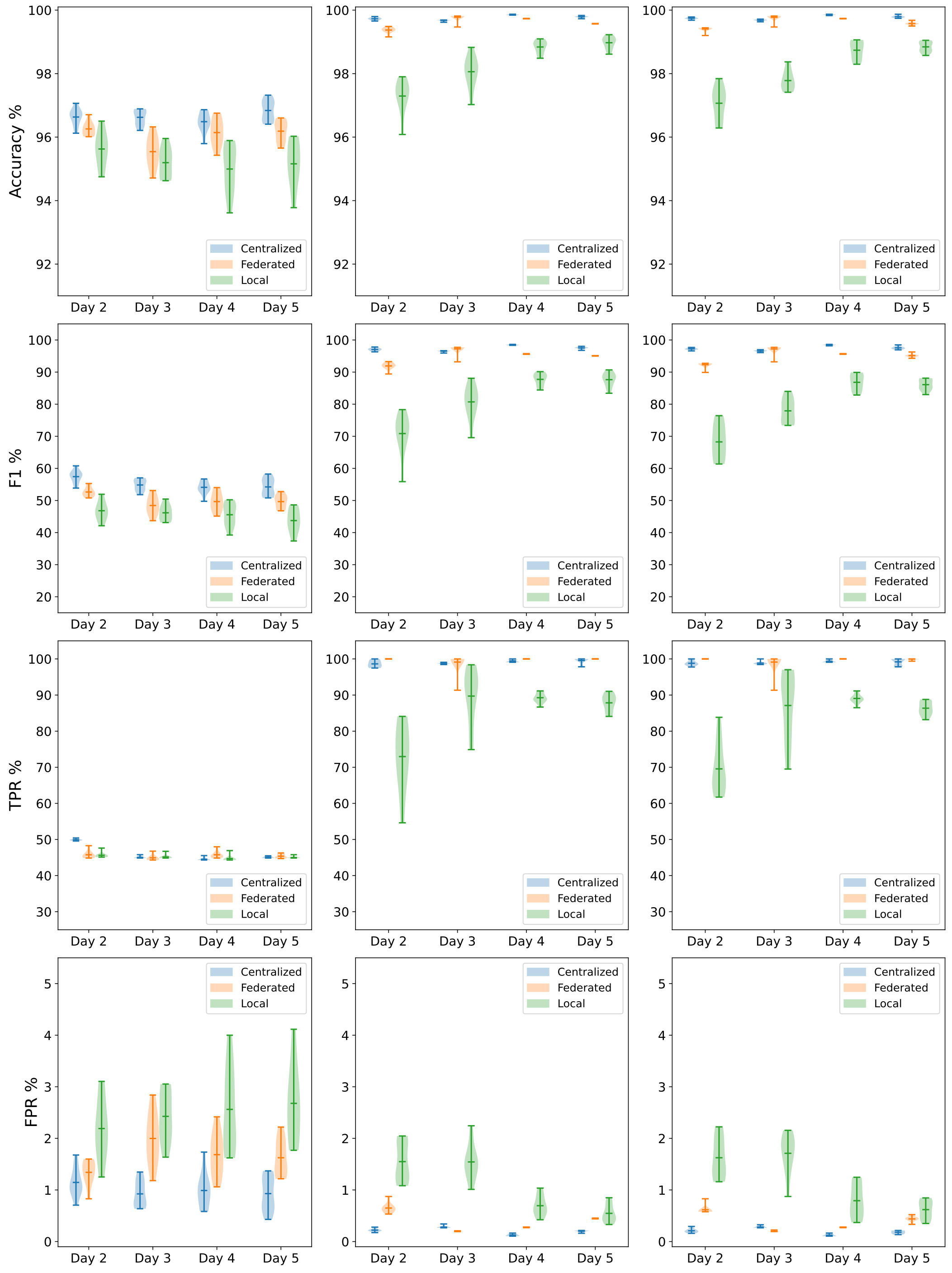
On Day 4, when more clients have malware, all models converged to a working state. However, the average recall of the *Classifier-only* model is significantly worse than that of the *Multi-Head* model and both of the models which reused models from previous days.

The last testing day with the highest amount of malware data again shows a worse performance of the *Classifier-only* model. When comparing the S4 and S3+S4 *Multi-Head* model outperforms its variant, which reuses the model from the previous day. There is also a much smaller variance in the results.

Overall, this more difficult data scenario shows that with fewer malicious samples in the clients' datasets, both evaluated supervised models face issues with convergence. In multiple cases, the models converged to local minima, where they classified every sample as benign. However, when reusing the model from the previous day, the model is able to recover from it.

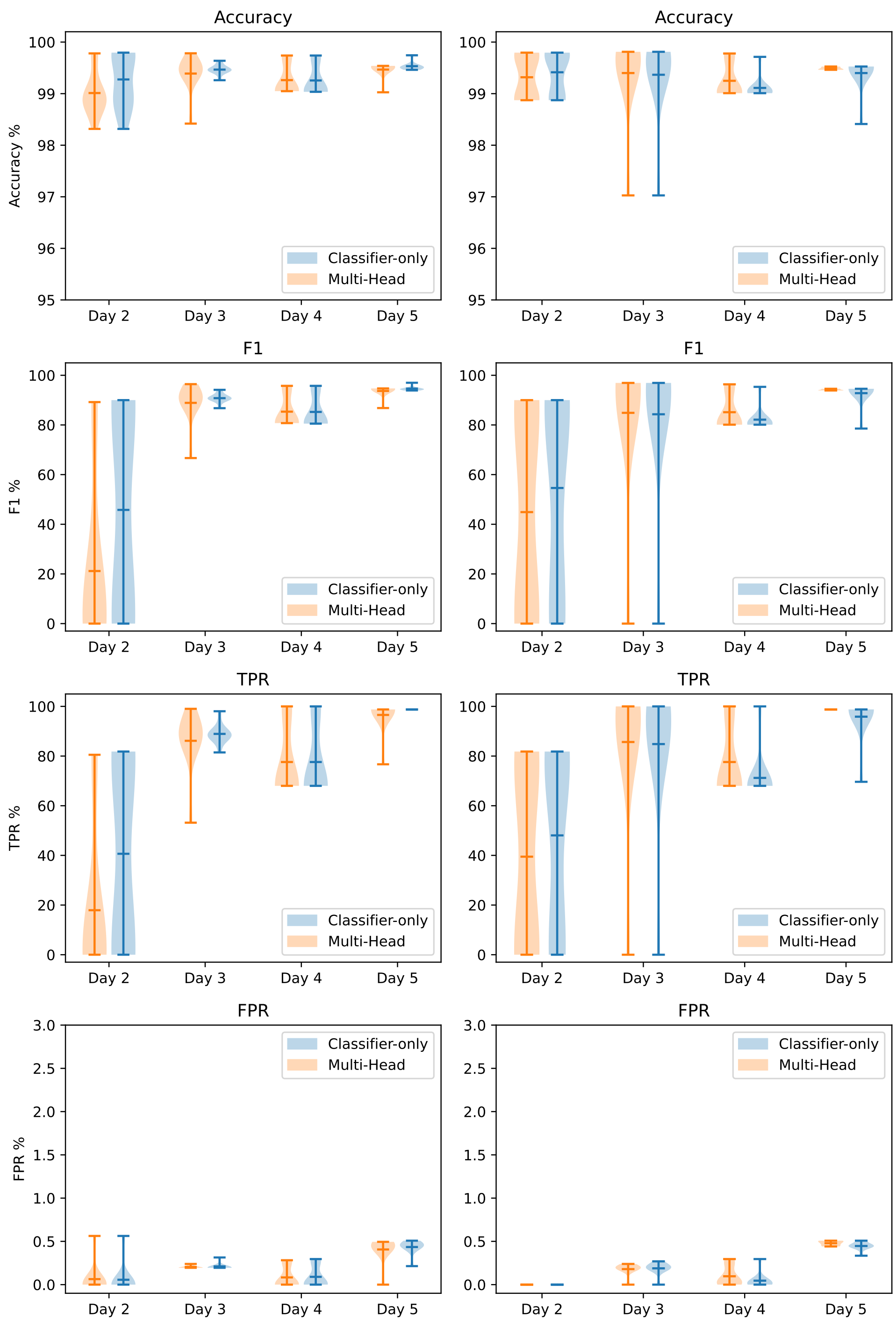
A shortcoming of the supervised methods is that it relies on the clients to label their data correctly and also to be infected to have malicious samples. In feature works, we would like to investigate a setting in which the aggregator provides bigger and more diverse vaccines as a form of threat intelligence, and Federated Learning would be used as a means to learn from clients' benign data. This approach would mitigate privacy concerns while taking advantage of public malware datasets created by the security community.

In this work, the models are always learning from a single day of network traffic. As having access to more training data at once is generally better, we would evaluate how the models would benefit from being trained on longer spans of activity. However, this might require a new dataset containing data for more days. While we did not observe any degradation in the performance of the models when they were reused for multiple days, it would be beneficial to test whether this holds for longer deployments. It would also be interesting to investigate whether the models need to be periodically retrained from scratch to maintain their performance. A longer dataset would be useful for this purpose.



(a) : Anomaly Detection Model **(b) :** *Multi-Head* model **(c) :** *Classifier-only* model

Figure 6.1: Comparison of the three types of models on the A1 (for unsupervised) and S1 (supervised) scenarios. The models trained on the CTU-50-FEEL dataset, split amongst ten clients. The days refer to the days on which the models were tested, being trained on the previous day. At the end of the day's training, the model was saved to be reused the next day.



(a) : S4 Experiment **(b) :** S3+S4 Experiment

Figure 6.2: Results of the supervised models on a CTU-50-FEEL-less-malware dataset. The models were trained from scratch each day in the S4+S3 experiment and the previous day's model was reused in the S4 experiment.

Chapter 7

Discussion

Our results showed that federated-trained models consistently outperformed those trained solely on local data. While centralizing data can often produce the most accurate models, it is not always possible due to technical limitations, privacy concerns, or legal requirements. In these cases, Federated Learning offers a way to train effective malware detection models while taking these considerations into account.

We evaluated the models on a series of training and testing days to somehow mimic an actual real deployment and the challenges related to it. In particular the challenge of how the distribution of data may shift between individual days. The datasets used in this work reflect this well, as the benign traffic was collected from actual users using different operating systems and sets of applications with notable differences in activity in the span of the five days.

One way to handle the situation of different distribution of data is to train a new model for each day, which ensures that the daily models are independent and do not suffer from degraded performance due to changes in data distribution.

The experiments were successful in showing that the models from the previous day could be reused, and its training could be resumed on a new day of data. This leads to lower computational and communication overhead, as fewer training rounds were necessary to adapt the model to new data than to train it from scratch. This approach also has the potential to enable the model to *remember* threats or types of data it encountered in the past. We attempted to evaluate the long-term memory capabilities of the model, but we did not observe any significant improvements in this aspect when reusing the model.

Obtaining reliable labels in the network security setting can be difficult as it usually requires deep domain knowledge from the user. To address this issue, we have developed a fully unsupervised method for detecting threats in the client's traffic. This method uses a variational auto-encoder and utilizes it for anomaly detection. The model is trained on a dataset of benign data, which allows it to learn the distribution of normal network traffic. By comparing new data to this distribution, the model can identify data that is unusual or out of the ordinary and mark it as potentially malicious. However, our experiments showed that this method was not always able to accurately detect all types of malware, as some of it was indistinguishable from normal traffic for the model.

To take advantage of labels, we also run supervised methods of detection. In this setting, we assume that only a subset of clients was able to observe the malicious activity. Having malicious samples only in some clients make the setting more realistic, as in the real world, it is unlikely that every client would be infected. That also means that the non-infected clients only have training samples of one class. Including these clients in the federated training process often results in the models not converging correctly and learning the trivial classification of marking every sample as benign. Although we were successful in addressing these convergence issues, it is clear that this is a particularly challenging aspect of Federated Learning.

The solution to this problem was to introduce the concept of "vaccine". A vaccine, for us, is a set of malicious data which is distributed to the clients, which would incorporate it into their own local dataset and use it for training. It is the responsibility of the aggregator to obtain this set of data by, for example, capturing the traffic of malware, processing it, and extracting features. These features can be safely distributed among the clients, as they are purely numerical and do not contain any of the malware's actual traffic or code. We have shown that when utilizing vaccines, the supervised models are able to converge to a state in which they accurately detect a majority of the malware. However, our experiments also showed that occasionally when the amount of malicious data available to the clients is small (for example, only the vaccine), the models may still fail to converge properly. In these cases, we suggest implementing a fallback system that would automatically restart the training process with a differently initialized model.

In the supervised experiments, we have evaluated two types of models: a *Multi-Head* model and a *Classifier-only* model. The *Multi-Head* model was trained on two tasks: (i) reconstructing an input feature vector similar to an autoencoder; (ii) classification of the traffic. The *Classifier-only* model was derived from the *Multi-Head* model by discarding the reconstruction part of the network. The rationale behind training the *Multi-Head* model on two tasks is that even clients with mostly benign data can contribute to the learning by improving the embedding of the model.

Our experiments showed that the simpler *Classifier-only* model achieves better results. However, in scenarios with fewer malicious samples, we were observed the convergence issues more often in the case of *Classifier-only* model than in the case of *Multi-Head* model.

Chapter 8
Conclusion

This thesis developed and evaluated both supervised and unsupervised Federated Learning methods for network security. We showed that these methods allow the clients participating in the training to distributively train models which detect a wider range of threats with higher accuracy compared to models trained only on their own local data. The used federated methods also ensure that the privacy of clients' data is not violated, as the training happens locally in the clients and their data never leaves their machines.

To evaluate the methods we created a new CTU-50-FEEL dataset and a dataset variant, both containing benign and malicious traffic. The benign data was created by capturing traffic of the people in the Stratosphere Laboratory[8] for five consecutive days. The malicious traffic originates from six distinct malware, all of which use HTTPS. Using HTTPS makes them more difficult to detect than malware that uses unencrypted connections. This results in a challenging federated dataset with realistic differences between the individual clients.

Our experiments showed that by utilizing the federally trained models, the clients were able detect a wider range of threats with better accuracy than if they relied only on their own data to create them. Although the model's performance (including its variance) improved slightly if trained on a centralized dataset, doing so may not be always possible due to privacy issues. In cases where the security of the data is important or regulation prevents sharing the data, the gained privacy may outweigh the loss in performance.

We have evaluated both supervised and unsupervised methods for detecting threats. The unsupervised approach utilizes a Variational Autoencoder trained to reconstruct the benign data. This model is then used for anomaly detection, marking the malicious feature vector as anomalous. The main benefit compared to the supervised approach is that it eliminates the need to label the data in the clients. However, it is not able to reliably detect all of the malware on which it was tested.

The supervised approach uses label information to train classification models. Two supervised models were used in our work. A *Multi-Head* model, which was trained for both classification and reconstruction of the samples, and a *Classifier-only* model. The main challenge in the supervised approach was that the dataset was non-IID. Specifically, only some of the clients had malware samples, and there were significant differences in the amounts of clients benign traffic. However, these properties are to be expected in real network security applications, and that is why we found it important to focus on them.

When developing the supervised methods, we observed that the inclusion of clients without any malicious samples impairs the convergence characteristics of the federated model. One approach to tackle the issue of the non-infected clients participating in the training is to provide them with some malicious samples. We did this in the form of a *vaccine*, which consists of numerical feature vectors of malware's activity. The aggregator sends these samples to the clients, where they can be incorporated into their local dataset, enabling the convergence of the supervised models. The vaccines always consisted in the traffic of a single malware. With the help of vaccines the models were also able to learn to distinguish other types of malware.

One of the goals of this thesis was to evaluate the methods under realistic conditions, such as an imbalance in the participants' computational resources and data volume. While we successfully created significant differences in the data volume of the clients, our setup did not allow for a straightforward way to simulate differences in computational resources between clients. However, the Flower framework used for Federated Learning allows for dropouts of individual clients, and in our experiments, we did examine the effects of fewer participants on the quality of the model.

As the dataset spanned five successive days, we were able to evaluate the methods in a simulated multi-day deployment, where it would be trained on data from the previous day and tested on the current one. We have evaluated if the models trained on the previous day can be repurposed to accelerate the training on the next day. We have shown that by doing so, the models required fewer training rounds to reach comparable performance to the models trained on that day from scratch. We also observed that in cases where the models did not converge successfully, they often recovered on the next day when reused.

8.1 Future Work

A shortcoming of the supervised methods is that it relies on the clients to label their data correctly and also to be infected to have malicious samples. In future work, we would like to investigate a setting in which the aggregator provides larger and more diverse vaccines. We would also like to investigate how Federated Learning would be used as a means to learn from clients' benign data. This approach would mitigate privacy concerns while taking advantage of public malware datasets created by the security community.

In this work, models always learnt from a single day of network traffic. Since having more data is usually better, we believe the models would benefit from being trained on longer spans of activity. However, this might require a new dataset containing data for more days. While we did not observe any degradation in the performance of the models when they were reused for multiple days, it would be beneficial to test whether this holds for longer deployments. It would also be interesting to investigate whether the models need to be periodically retrained from scratch to maintain their performance.

Further ideas for future research in this area might include taking advantage of model personalization[26, 27, 28]. In this approach, at the end of the federated training process, the clients train locally on the global models. By fine-tuning them using their local data, they can better adapt it for their purpose. This might be beneficial in the unsupervised anomaly detection approach, where the model has to learn to represent normal traffic. We observed the normal traffic of the clients could differ significantly across the clients, and personalizing the models on the normal data of each client might lead to better performance. In this case, a new approach to learning the anomaly threshold would have to be developed.

Appendix A

Detailed Experiment Results

A.1 Detailed Anomaly Detection Results

		Cent.	Fed.	Local
Day 2	$Acc \pm std$	96.63 ± 0.27	96.26 ± 0.22	95.63 ± 0.56
	$F1 \pm std$	57.36 ± 1.96	52.61 ± 1.50	46.61 ± 3.00
	$FPR \pm std$	1.15 ± 0.28	1.34 ± 0.24	2.19 ± 0.60
	$TPR \pm std$	49.92 ± 0.26	45.76 ± 1.07	45.63 ± 0.74
Day 3	$Acc \pm std$	96.62 ± 0.25	95.54 ± 0.49	95.20 ± 0.48
	$F1 \pm std$	54.77 ± 1.81	48.29 ± 2.90	46.04 ± 2.45
	$FPR \pm std$	0.92 ± 0.26	2.00 ± 0.50	2.43 ± 0.51
	$TPR \pm std$	45.05 ± 0.33	44.92 ± 0.77	45.17 ± 0.57
Day 4	$Acc \pm std$	96.49 ± 0.32	96.14 ± 0.42	94.99 ± 0.76
	$F1 \pm std$	54.00 ± 2.09	49.52 ± 2.99	45.28 ± 3.58
	$FPR \pm std$	0.99 ± 0.35	1.68 ± 0.42	2.56 ± 0.81
	$TPR \pm std$	44.52 ± 0.41	45.76 ± 0.95	44.70 ± 0.80
Day 5	$Acc \pm std$	96.84 ± 0.33	96.19 ± 0.33	95.16 ± 0.73
	$F1 \pm std$	54.10 ± 2.68	49.56 ± 2.08	43.45 ± 3.71
	$FPR \pm std$	0.93 ± 0.34	1.62 ± 0.36	2.68 ± 0.76
	$TPR \pm std$	45.08 ± 0.26	45.37 ± 0.54	45.00 ± 0.29

Table A.1: Results of the A1 experiment comparing the federated anomaly detection model to the centralized and local models.

		Cent.	Fed.	Local
Day 2	$Acc \pm std$	96.66 ± 0.33	96.20 ± 0.22	95.76 ± 0.64
	$F1 \pm std$	57.55 ± 2.26	52.02 ± 1.52	47.41 ± 3.34
	$FPR \pm std$	1.12 ± 0.35	1.38 ± 0.23	2.05 ± 0.69
	$TPR \pm std$	49.92 ± 0.34	45.36 ± 0.30	45.63 ± 0.70
Day 3	$Acc \pm std$	96.60 ± 0.38	95.75 ± 0.49	95.73 ± 0.60
	$F1 \pm std$	54.56 ± 2.72	49.46 ± 2.97	49.01 ± 3.11
	$FPR \pm std$	0.95 ± 0.40	1.78 ± 0.50	1.87 ± 0.66
	$TPR \pm std$	44.98 ± 0.26	44.89 ± 0.64	45.22 ± 0.58
Day 4	$Acc \pm std$	96.51 ± 0.28	96.39 ± 0.19	95.24 ± 0.87
	$F1 \pm std$	54.19 ± 1.90	50.94 ± 1.24	46.67 ± 3.78
	$FPR \pm std$	0.96 ± 0.29	1.41 ± 0.21	2.32 ± 0.96
	$TPR \pm std$	44.49 ± 0.22	45.39 ± 0.53	45.01 ± 0.94
Day 5	$Acc \pm std$	96.84 ± 0.21	95.95 ± 0.47	95.69 ± 0.47
	$F1 \pm std$	54.10 ± 1.60	48.11 ± 2.76	46.48 ± 2.66
	$FPR \pm std$	0.93 ± 0.22	1.88 ± 0.51	2.14 ± 0.50
	$TPR \pm std$	45.08 ± 0.21	45.53 ± 0.73	45.29 ± 0.50

Table A.2: Results of the A2 experiment comparing the federated anomaly detection model to the centralized and local models.

		Cent.	Fed.	Local
Day 2	$Acc \pm std$	96.46 ± 0.48	94.75 ± 0.74	95.33 ± 0.61
	$F1 \pm std$	56.10 ± 3.52	44.10 ± 3.32	45.09 ± 3.05
	$FPR \pm std$	1.32 ± 0.51	2.92 ± 0.79	2.51 ± 0.64
	$TPR \pm std$	49.83 ± 0.31	45.64 ± 0.57	45.82 ± 0.71
Day 3	$Acc \pm std$	96.33 ± 0.46	93.71 ± 1.29	94.07 ± 0.55
	$F1 \pm std$	52.74 ± 3.12	40.02 ± 5.23	40.97 ± 2.51
	$FPR \pm std$	1.24 ± 0.48	3.94 ± 1.37	3.62 ± 0.56
	$TPR \pm std$	45.14 ± 0.37	45.29 ± 0.80	45.37 ± 0.62
Day 4	$Acc \pm std$	96.77 ± 0.18	95.36 ± 0.67	94.81 ± 0.76
	$F1 \pm std$	56.07 ± 1.40	44.87 ± 3.77	44.49 ± 3.58
	$FPR \pm std$	0.69 ± 0.19	2.50 ± 0.70	2.76 ± 0.79
	$TPR \pm std$	44.49 ± 0.33	45.67 ± 0.51	44.87 ± 0.64
Day 5	$Acc \pm std$	96.77 ± 0.23	95.29 ± 0.81	95.65 ± 0.34
	$F1 \pm std$	53.51 ± 1.79	44.35 ± 4.13	46.21 ± 1.95
	$FPR \pm std$	1.00 ± 0.24	2.56 ± 0.85	2.18 ± 0.36
	$TPR \pm std$	44.98 ± 0.22	45.47 ± 0.53	45.20 ± 0.48

Table A.3: Results of the A3 experiment with six clients comparing the federated anomaly detection model to the centralized and local models.

		Cent.	Fed.	Local
Day 2	$Acc \pm std$	96.28 ± 0.55	93.88 ± 0.85	94.61 ± 0.63
	$F1 \pm std$	54.78 ± 3.77	40.21 ± 3.12	41.29 ± 2.95
	$FPR \pm std$	1.50 ± 0.58	3.81 ± 0.90	3.24 ± 0.66
	$TPR \pm std$	49.66 ± 0.14	45.33 ± 0.40	45.34 ± 0.18
Day 3	$Acc \pm std$	96.55 ± 0.69	93.61 ± 1.27	92.65 ± 1.94
	$F1 \pm std$	54.18 ± 4.52	39.46 ± 4.65	35.72 ± 6.50
	$FPR \pm std$	1.00 ± 0.73	4.03 ± 1.31	5.09 ± 2.04
	$TPR \pm std$	44.98 ± 0.26	44.95 ± 0.53	45.02 ± 0.21
Day 4	$Acc \pm std$	95.71 ± 0.68	94.03 ± 0.92	90.99 ± 1.43
	$F1 \pm std$	49.05 ± 3.90	38.68 ± 3.71	31.36 ± 3.51
	$FPR \pm std$	1.80 ± 0.72	3.88 ± 0.96	6.75 ± 1.50
	$TPR \pm std$	44.52 ± 0.33	45.58 ± 0.71	44.42 ± 0.19
Day 5	$Acc \pm std$	96.70 ± 0.38	95.11 ± 0.95	95.25 ± 1.02
	$F1 \pm std$	53.00 ± 2.74	43.46 ± 4.22	43.84 ± 5.04
	$FPR \pm std$	1.07 ± 0.40	2.76 ± 1.01	2.58 ± 1.06
	$TPR \pm std$	45.02 ± 0.22	45.53 ± 0.57	44.88 ± 0.07

Table A.4: Results of the A3 experiment with six clients comparing the federated anomaly detection model to the centralized and local models.

A.2 Detailed Classification Results

		Day 2			Day 3			Day 4			Day 5		
		Cent.	Fed.	Local	Cent.	Fed.	Local	Cent.	Fed.	Local	Cent.	Fed.	Local
S1 Multihead Model	$Acc \pm std$	99.73 ± 0.04	99.37 ± 0.09	97.29 ± 0.52	99.68 ± 0.02	99.78 ± 0.11	98.06 ± 0.50	99.85 ± 0.01	99.73 ± 0.01	98.84 ± 0.20	99.79 ± 0.03	99.58 ± 0.00	98.97 ± 0.19
	$TPR \pm std$	98.63 ± 0.99	100.00 ± 0.00	72.96 ± 9.35	98.75 ± 0.15	99.13 ± 2.74	89.72 ± 7.68	99.45 ± 0.48	100.00 ± 0.00	89.26 ± 1.41	99.56 ± 0.92	100.00 ± 0.00	87.84 ± 2.43
	$FPR \pm std$	0.22 ± 0.03	0.65 ± 0.09	1.55 ± 0.39	0.28 ± 0.02	0.20 ± 0.00	1.54 ± 0.35	0.13 ± 0.02	0.28 ± 0.01	0.70 ± 0.20	0.20 ± 0.02	0.44 ± 0.00	0.55 ± 0.17
	$F1 \pm std$	97.04 ± 0.46	91.90 ± 1.06	70.99 ± 6.28	96.51 ± 0.24	97.26 ± 1.42	80.76 ± 5.21	98.43 ± 0.10	95.58 ± 0.09	87.69 ± 1.91	97.50 ± 0.35	95.11 ± 0.04	87.61 ± 2.20
S1 Classifier Only Model	$Acc \pm std$	99.75 ± 0.03	99.41 ± 0.07	97.07 ± 0.52	99.68 ± 0.02	99.77 ± 0.11	97.78 ± 0.31	99.85 ± 0.01	99.73 ± 0.01	98.74 ± 0.30	99.79 ± 0.04	99.58 ± 0.04	98.84 ± 0.17
	$TPR \pm std$	98.80 ± 0.67	100.00 ± 0.00	69.55 ± 7.24	98.85 ± 0.42	99.13 ± 2.74	87.10 ± 9.64	99.35 ± 0.45	100.00 ± 0.00	89.06 ± 1.33	99.19 ± 1.06	99.94 ± 0.20	86.36 ± 1.98
	$FPR \pm std$	0.21 ± 0.04	0.62 ± 0.08	1.62 ± 0.40	0.28 ± 0.02	0.20 ± 0.01	1.71 ± 0.38	0.12 ± 0.02	0.28 ± 0.01	0.79 ± 0.33	0.18 ± 0.03	0.44 ± 0.05	0.62 ± 0.19
	$F1 \pm std$	97.24 ± 0.34	92.29 ± 0.87	68.28 ± 5.53	96.55 ± 0.27	97.22 ± 1.42	78.08 ± 3.89	98.43 ± 0.13	95.62 ± 0.10	86.73 ± 2.66	97.55 ± 0.45	95.12 ± 0.48	86.05 ± 1.77
S2 Multihead Model	$Acc \pm std$	99.73 ± 0.04	99.45 ± 0.13	96.96 ± 0.51	99.69 ± 0.02	99.75 ± 0.04	96.96 ± 1.00	99.84 ± 0.01	99.73 ± 0.02	99.39 ± 0.09	99.85 ± 0.03	99.54 ± 0.04	99.02 ± 0.41
	$TPR \pm std$	99.50 ± 0.62	100.00 ± 0.00	79.62 ± 5.40	99.81 ± 0.39	100.00 ± 0.00	89.90 ± 6.58	99.82 ± 0.39	100.00 ± 0.00	93.35 ± 2.34	99.78 ± 0.69	100.00 ± 0.00	100.00 ± 0.00
	$FPR \pm std$	0.26 ± 0.04	0.57 ± 0.13	2.21 ± 0.47	0.32 ± 0.03	0.26 ± 0.05	2.71 ± 0.92	0.15 ± 0.02	0.28 ± 0.02	0.32 ± 0.09	0.15 ± 0.03	0.48 ± 0.04	1.02 ± 0.43
	$F1 \pm std$	97.10 ± 0.46	92.82 ± 1.49	70.40 ± 4.39	96.67 ± 0.24	96.97 ± 0.51	72.84 ± 7.36	98.35 ± 0.15	95.60 ± 0.37	93.41 ± 0.96	98.18 ± 0.37	94.71 ± 0.44	89.43 ± 3.61
S2 Classifier Only Model	$Acc \pm std$	99.74 ± 0.03	99.39 ± 0.11	97.19 ± 0.32	99.69 ± 0.02	99.77 ± 0.04	97.86 ± 0.57	99.86 ± 0.02	99.73 ± 0.02	99.40 ± 0.11	99.86 ± 0.01	99.56 ± 0.04	99.05 ± 0.30
	$TPR \pm std$	99.30 ± 0.66	100.00 ± 0.00	81.84 ± 4.18	99.60 ± 0.59	100.00 ± 0.00	91.42 ± 9.66	99.91 ± 0.29	99.69 ± 0.98	93.85 ± 2.60	100.00 ± 0.00	100.00 ± 0.00	99.93 ± 0.16
	$FPR \pm std$	0.24 ± 0.03	0.63 ± 0.12	2.08 ± 0.28	0.31 ± 0.03	0.24 ± 0.04	1.84 ± 0.41	0.14 ± 0.01	0.27 ± 0.04	0.33 ± 0.04	0.14 ± 0.01	0.45 ± 0.04	0.99 ± 0.31
	$F1 \pm std$	97.16 ± 0.34	92.15 ± 1.30	72.55 ± 2.95	96.64 ± 0.20	97.23 ± 0.45	79.47 ± 5.93	98.48 ± 0.16	95.55 ± 0.29	93.59 ± 1.27	98.38 ± 0.16	94.99 ± 0.39	89.65 ± 2.71
S3 Multihead Model	$Acc \pm std$	99.75 ± 0.03	99.35 ± 0.25	97.28 ± 0.45	99.68 ± 0.03	99.80 ± 0.03	97.62 ± 0.51	99.83 ± 0.02	99.72 ± 0.01	97.62 ± 0.40	99.78 ± 0.02	99.55 ± 0.02	97.87 ± 0.36
	$TPR \pm std$	98.57 ± 0.55	100.00 ± 0.00	69.52 ± 8.00	99.38 ± 0.66	100.00 ± 0.00	75.70 ± 11.29	99.82 ± 0.39	100.00 ± 0.00	54.62 ± 6.43	99.91 ± 0.30	100.00 ± 0.00	59.63 ± 2.05
	$FPR \pm std$	0.20 ± 0.03	0.67 ± 0.26	1.40 ± 0.28	0.30 ± 0.04	0.21 ± 0.03	1.34 ± 0.22	0.17 ± 0.02	0.29 ± 0.01	0.29 ± 0.12	0.22 ± 0.03	0.47 ± 0.02	0.48 ± 0.41
	$F1 \pm std$	97.25 ± 0.36	91.63 ± 3.18	69.89 ± 5.62	96.61 ± 0.26	97.50 ± 0.37	74.23 ± 6.73	98.15 ± 0.17	95.38 ± 0.21	68.03 ± 6.35	97.45 ± 0.22	94.80 ± 0.25	69.86 ± 3.20
S3 Classifier Only Model	$Acc \pm std$	99.76 ± 0.04	99.30 ± 0.13	97.24 ± 0.73	99.69 ± 0.01	99.05 ± 1.60	97.81 ± 0.54	99.84 ± 0.02	99.19 ± 1.12	97.58 ± 0.23	99.78 ± 0.02	99.20 ± 1.17	98.02 ± 0.15
	$TPR \pm std$	98.94 ± 0.79	100.00 ± 0.00	71.13 ± 12.09	98.75 ± 0.00	80.00 ± 42.16	76.78 ± 12.32	99.82 ± 0.39	80.00 ± 42.16	51.75 ± 7.57	99.78 ± 0.69	90.00 ± 31.62	57.94 ± 4.78
	$FPR \pm std$	0.20 ± 0.03	0.72 ± 0.13	1.52 ± 0.37	0.27 ± 0.01	0.16 ± 0.08	1.19 ± 0.31	0.16 ± 0.03	0.23 ± 0.12	0.19 ± 0.16	0.22 ± 0.03	0.41 ± 0.14	0.25 ± 0.18
	$F1 \pm std$	97.37 ± 0.44	91.08 ± 1.48	70.07 ± 8.96	96.65 ± 0.10	87.02 ± 41.17	76.12 ± 7.16	98.29 ± 0.16	85.35 ± 40.28	66.49 ± 5.25	97.36 ± 0.26	90.25 ± 30.05	70.77 ± 3.06
S4 Multihead Model	$Acc \pm std$	99.99 ± 0.02	99.01 ± 0.45	99.18 ± 0.23	99.50 ± 0.22	99.39 ± 0.36	98.23 ± 0.37	100.00 ± 0.01	99.26 ± 0.33	99.27 ± 0.24	99.78 ± 0.02	99.47 ± 0.16	98.96 ± 0.18
	$TPR \pm std$	100.00 ± 0.00	17.92 ± 33.21	100.00 ± 0.00	93.96 ± 15.35	86.15 ± 12.07	79.49 ± 14.26	100.00 ± 0.00	77.60 ± 15.46	93.25 ± 6.99	100.00 ± 0.00	96.56 ± 6.98	83.88 ± 1.98
	$FPR \pm std$	0.01 ± 0.02	0.06 ± 0.18	0.83 ± 0.24	0.38 ± 0.16	0.20 ± 0.02	1.36 ± 0.42	0.00 ± 0.01	0.08 ± 0.13	0.50 ± 0.08	0.23 ± 0.02	0.41 ± 0.14	0.39 ± 0.23
	$F1 \pm std$	99.59 ± 0.66	29.02 ± 36.51	77.64 ± 4.98	89.00 ± 7.79	89.35 ± 8.02	65.94 ± 7.25	99.94 ± 0.13	86.05 ± 7.13	90.36 ± 3.80	97.44 ± 0.18	93.80 ± 2.43	86.95 ± 1.86
S4 Classifier Only Model	$Acc \pm std$	99.97 ± 0.02	99.28 ± 0.55	99.09 ± 0.33	99.60 ± 0.07	99.47 ± 0.10	98.16 ± 0.37	100.00 ± 0.00	99.26 ± 0.33	99.18 ± 0.25	99.78 ± 0.02	99.53 ± 0.08	98.75 ± 0.38
	$TPR \pm std$	100.00 ± 0.00	40.65 ± 41.36	99.00 ± 3.16	98.93 ± 0.85	88.93 ± 4.08	78.62 ± 16.53	100.00 ± 0.00	77.60 ± 15.46	94.50 ± 3.00	99.69 ± 0.72	98.77 ± 0.00	83.51 ± 2.32
	$FPR \pm std$	0.03 ± 0.02	0.06 ± 0.18	0.91 ± 0.36	0.39 ± 0.06	0.21 ± 0.04	1.40 ± 0.56	0.00 ± 0.00	0.09 ± 0.13	0.64 ± 0.27	0.21 ± 0.02	0.43 ± 0.08	0.59 ± 0.37
	$F1 \pm std$	98.86 ± 0.62	55.84 ± 45.45	75.44 ± 6.55	91.44 ± 1.48	90.83 ± 1.86	64.90 ± 6.40	100.00 ± 0.00	85.95 ± 7.11	89.41 ± 2.86	97.43 ± 0.19	94.64 ± 0.86	84.70 ± 4.07
S3+S4 Multihead Model	$Acc \pm std$	99.97 ± 0.02	99.32 ± 0.46	99.30 ± 0.17	99.67 ± 0.02	99.40 ± 0.85	98.24 ± 0.37	100.00 ± 0.00	99.25 ± 0.34	98.41 ± 0.15	99.78 ± 0.02	99.49 ± 0.03	97.80 ± 0.39
	$TPR \pm std$	100.00 ± 0.00	39.48 ± 40.41	99.00 ± 3.16	98.52 ± 1.30	85.66 ± 30.59	79.67 ± 16.36	100.00 ± 0.00	77.60 ± 15.46	60.76 ± 6.61	99.78 ± 0.69	98.77 ± 0.00	59.82 ± 2.76
	$FPR \pm std$	0.03 ± 0.02	0.00 ± 0.00	0.69 ± 0.19	0.30 ± 0.03	0.18 ± 0.06	1.35 ± 0.40	0.00 ± 0.00	0.10 ± 0.12	0.17 ± 0.11	0.22 ± 0.03	0.48 ± 0.03	0.57 ± 0.40
	$F1 \pm std$	99.04 ± 0.66	56.61 ± 44.95	80.11 ± 3.60	92.88 ± 0.38	89.45 ± 29.96	66.19 ± 8.19	100.00 ± 0.00	85.84 ± 7.38	73.58 ± 4.01	97.44 ± 0.30	94.18 ± 0.30	69.16 ± 4.12
S3+S4 Classifier Only Model	$Acc \pm std$	99.98 ± 0.02	99.41 ± 0.44	99.10 ± 0.27	99.66 ± 0.01	99.37 ± 0.84	98.24 ± 0.27	100.00 ± 0.00	99.11 ± 0.21	98.40 ± 0.18	99.78 ± 0.01	99.40 ± 0.35	97.90 ± 0.41
	$TPR \pm std$	100.00 ± 0.00	48.05 ± 39.31	100.00 ± 0.00	96.91 ± 0.85	84.83 ± 30.31	81.60 ± 11.37	100.00 ± 0.00	71.20 ± 10.12	62.15 ± 7.08	100.00 ± 0.00	95.86 ± 9.22	58.79 ± 3.43
	$FPR \pm std$	0.02 ± 0.02	0.00 ± 0.00	0.91 ± 0.28	0.28 ± 0.01	0.19 ± 0.07	1.39 ± 0.27	0.00 ± 0.00	0.05 ± 0.09	0.23 ± 0.11	0.23 ± 0.02	0.45 ± 0.05	0.41 ± 0.44
	$F1 \pm std$	99.31 ± 0.72	64.91 ± 43.15	75.98 ± 5.40	92.39 ± 0.31	88.84 ± 29.74	66.72 ± 5.77	100.00 ± 0.00	82.45 ± 4.65	73.85 ± 4.56	97.42 ± 0.17	93.02 ± 5.02	69.84 ± 4.29

Table A.5: Detailed results of the supervised experiments comparing the federally trained detection model to the centralized and local models.

Appendix B

Bibliography

[1] Check Point. Check Point Research: Third quarter of 2022 reveals increase in cyberattacks and unexpected developments in global trends - Check Point Software — blog.checkpoint.com. `https://blog.checkpoint.com/2022/10/26/third-quarter-of-2022-reveals-increase-in-cyberattacks/#:~:text=Global%20attacks%20increased%20by%2028,organization%20worldwide%20reached%20over%201%2C130`, 2022. [Accessed 08-Jan-2023].

[2] Steve Morgan. Cybercrime To Cost The World $10.5 Trillion Annually By 2025 — cybersecurityventures.com. `https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/`, 2020. [Accessed 08-Jan-2023].

[3] Valerian Rey, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and Gérôme Bovet. Federated learning for malware detection in IoT devices. *Computer Networks*, 204, 2022. Publisher: Elsevier B.V.

[4] Chandra Thapa, Kallol Krishna Karmakar, Alberto Huertas Celdran, Seyit Camtepe, Vijay Varadharajan, and Surya Nepal. FedDICE: A ransomware spread detection in a distributed integrated clinical environment using federated learning and SDN based mitigation. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2021.

[5] W3Techs. Historical yearly trends in the usage statistics of site elements for websites, January 2023 — w3techs.com. `https://w3techs.com/technologies/history_overview/site_element/all/y`, 2023. [Accessed 08-Jan-2023].

[6] Andre Wichmann and Elmar Gerhards-Padilla. Using infection markers as a vaccine against malware attacks. In *2012 IEEE International*

Conference on Green Computing and Communications, pages 737–742, 2012.

[7] Pavel Janata, Maria Rigaki, and Sebastian Garcia. Ctu-50-feel. `https://zenodo.org/record/7515406`, Jan 2023.

[8] Sebastian Garcia, Veronica Valeros, et al. Stratosphere research laboratory. `https://www.stratosphereips.org/`. *Cybersecurity group of the Artificial Intelligence Centre, Faculty of Electrical Engineering at the Czech Technical University.* Last Accessed: 2023-01-07.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* Adaptive computation and machine learning. The MIT Press, 2016.

[10] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2018.

[11] David M. W. Powers. Evaluation: from precision, recall and f-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2021.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. Number: 6088 Publisher: Nature Publishing Group.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *preprint arXiv:1312.6114*, 2014.

[15] H Brendan McMahan, Eider Moore, Daniel Ramage, and Seth Hampson. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[16] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, et al. Towards federated learning at scale: System design. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019.

[17] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions, 2018.

[18] Chen Zhu, Zheng Xu, Mingqing Chen, Jakub Konečný, Andrew Hard, and Tom Goldstein. Diurnal or nocturnal? federated learning of multi-branch networks from periodically shifting distributions. In *International Conference on Learning Representations*, 2022.

[19] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. In *Proceedings on Privacy Enhancing Technologies*, volume 2017, pages 345–364, 2017.

[20] Ruei-Hau Hsu, Yi-Cheng Wang, Chun-I Fan, Bo Sun, Tao Ban, Takeshi Takahashi, Ting-Wei Wu, and Shang-Wei Kao. A privacy-preserving federated learning system for android malware detection based on edge computing. In *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 128–136, 2020.

[21] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated gradient boosting decision trees. In *AAAI-20*, 2019. Number: arXiv:1911.04206.

[22] Yang Liu, Zhuo Ma, Ximeng Liu, Zhuzhu Wang, Siqi Ma, and Ken Ren. Revocable federated learning: A benchmark of federated forest. *arXiv preprint arXiv:1911.03242*, 2019. Number: arXiv:1911.03242.

[23] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[24] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *2021 The International Conference on Learning Representations*, 2021.

[25] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Sixth Conference on Machine Learning and Systems*. arXiv, 2020.

[26] Kaan Ozkara, Navjot Singh, Deepesh Data, and Suhas Diggavi. QuPeL: Quantized personalization with applications to federated learning. In *Advances in Neural Information Processing Systems*, 2021.

[27] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758 [cs, stat]*, 2022.

[28] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization. *arXiv:1910.10252 [cs, stat]*, 2019.

[29] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *Advances in Neural Information Processing Systems*, volume 33, pages 3557–3568. Curran Associates, Inc., 2020.

[30] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 2017.

[31] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, 2018.

[32] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Neural Information Processing Systems*, volume abs/2007.07481, 2020.

[33] Irem Ergun, Hasin Us Sami, and Basak Guler. Sparsified secure aggregation for privacy-preserving federated learning. *Computers and Security*, 2021.

[34] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[35] Siddharth Sharma. Mirai code re-use in Gafgyt. `https://www.uptycs.com/blog/mirai-code-re-use-in-gafgyt`.

[36] 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*, 9(4):2545–2554, 2022. Conference Name: IEEE Internet of Things Journal.

[37] 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.

[38] Gustavo de Carvalho Bertoli, Lourenço Alves Pereira Júnior, and Osamu Saotome. Improving detection of scanning attacks on heterogeneous networks with federated learning. *ACM SIGMETRICS Performance Evaluation Review*, 49(4):118–123, 2022.

[39] Yuwei Sun, Hideya Ochiai, and Hiroshi Esaki. Intrusion detection with segmented federated learning for large-scale multiple LANs. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. ISSN: 2161-4407.

[40] Geet Shingi, Harsh Saglani, and Preeti Jain. Segmented federated learning for adaptive intrusion detection system. *arXiv preprint arXiv:2107.00881*, 2021.

[41] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. Cyber threat intelligence sharing scheme based on federated learning for network intrusion detection. *Journal of Network and Systems Management*, 31(1):1–23, 2023.

[42] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[43] František Střasák. *Detection of HTTPS malware traffic.* Bachelor's thesis, Czech Technical University Prague, Czech Republic, 2017.

[44] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*, pages 1–5, 2018.

[45] SungJin Kim, WooYeon Jo, and Taeshik Shon. APAD: Autoencoder-based payload anomaly detection for industrial IoE. *Applied Soft Computing*, 88:106017, 2022.

[46] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. *Advances and Open Problems in Federated Learning.* Now Publishers, Inc., 2021.

[47] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[49] François Chollet et al. Keras. `https://keras.io`, 2015.

[50] Aashish Sharma, Christian Kreibich, Fatema Bannat, Johanna Amann, Keith Lehigh, et al. Zeek: An open source network security monitoring tool. `https://zeek.org/`. Last Accessed: 2023-01-07.

[51] Sebastian Garcia. Netflow Labeler: A configurable rule-based labeling tool for network flow files.