Federated Learning for Classifying Network Intrusions

Anthony Peteros Thapa
B922168
Loughborough University

A thesis submitted for the degree of
Masters in Science

June 2023

Abstract

The thesis offers a comprehensive investigation into the strengths and weaknesses of federated learning, a decentralised approach to machine learning. It focuses on the feasibility and effectiveness of various federated learning strategies as an alternative to traditional machine learning methods, with a special emphasis on their application in classifying network intrusions. This is achieved by leveraging the distributed and heterogeneous data generated by Internet of Things (IoT) devices. Although centralised learning demonstrated marginally superior overall accuracy, federated learning proved to be more effective in identifying a wider range of network intrusions, especially within minority classes. Among the examined federated learning strategies, FedAvg delivered superior performance with feature selection, whereas FedProx showed strength when all features were considered. Challenges encountered included memory constraints posed by the use of virtual clients.

Acknowledgements

Foremost, I would like to express my gratitude and appreciation to Dr Posco Tso. His consistency and guidance have been an invaluable asset throughout this project. Thank you for giving up your time for constructive feedback throughout the year and ensuring my progress stayed on track. I would also like to thank my family and friends, whose support and encouragement have been a source of strength and inspiration during the times when the journey became demanding.

Contents

1 Introduction

The rapid growth of the Internet of Things (IoT) has led to an explosion in the amount of data generated. This provides rich opportunities for machine learning applications, such as Network Intrusion Detection Systems (NIDs), critical tools used for monitoring network traffic in order to identify potential threats [1]. Traditional machine learning models, such as neural networks and linear regression, rely largely on centralised learning. That involves collecting data from multiple devices and sending it to a central server for model training. However, the approach raises several concerns relating to privacy, regulatory compliance, and the computational and economic feasibility of handling high volumes of distributed data [2]. In the context of NIDS, these concerns are especially critical due to the sensitivity of network traffic data and the requirement for real-time protection against network intrusions. Network traffic data contains private information about user behaviour and the network. This means that the use of centralised learning could lead to a breach in data privacy, which results in severe security risks, emphasising the need for a better solution to preserving data privacy. Moreover, NIDS often deals with datasets that are large and diverse, due to the generation of network data from many devices with different characteristics. Along with the requirement to process data in real time, this gives central servers severe computational load to manage when using centralised learning. In response to these challenges, this thesis explores Federated Learning as an alternative novel approach. Federated learning is developed as a decentralised solution for handling data privacy and management issues, by allowing the learning process to be shifted from the central server to the devices where data resides. As a result, it has the capacity to process growing volumes of distributed heterogeneous data in real time without compromising privacy, making it especially beneficial for NIDS [3].

1.1 Thesis Objectives & Approach

The primary objective of this thesis is to assess the feasibility and effectiveness of different federated learning strategies for the classification of network intrusions using data derived from IoT devices. The focus on classification over other tasks within NIDS is motivated by the potential of this approach to more effectively and realistically determine network intrusions using decentralised and limited data. Firstly, a comprehensive review of existing theories, methods, and findings associated with federated learning will be conducted, highlighting the benefits and challenges. This thesis will establish a practically applied approach, focusing on the design, development, and optimisation of prototype federated learning models. The created federated learning models will then be evaluated against centralised learning based on performance and feasibility in relation to real-world scenarios.

1.2 Expectations

Consequently, it is expected that the findings of the thesis may reveal significant implications for the development of NIDS solutions that can effectively address the limitations of centralised learning. This research could shed light on the potential benefits that federated learning may offer, in terms of practicability and accuracy for network intrusion detection when using distributed data. Additionally, this thesis may uncover limitations and possible future directions for federated learning, such as challenges in the implementation of different learning or communication strategies, or managing a large number of clients.

2 Federated Learning

Federated learning is a decentralised approach to machine learning that allows the utilisation of distributed and heterogeneous data [4]. This approach to machine learning shifts the training process from a central location to the devices where the data resides. The following review will cover the many benefits and challenges that this novel strategy brings to practice. Moreover, the key components of federated learning can be explained in terms of the model's initialisation and distribution, local training within clients, and the transfer and aggregation of model updates.

2.1 Model Initialisation & Distribution

A global model is first initialised on a central server, similar to conventional centralised learning, using parameters that are arbitrary or previously saved. The initialised model is then distributed to participating clients to allow the model to be trained locally in the clients that they are shared with. These clients can include a wide range of devices such as IoT devices, smartphones, speakers, and even edge servers. Practically, not all clients are used for training the model simultaneously; a sample of clients is selected based on specified statuses [5]. This is to target devices with specific statuses, e.g., to select clients based on their availability, resource constraints, or data distribution. This approach enables more efficient training and avoids diminishing returns, for instance, training while the device is not in use.

2.2 Local training within clients

With the shared global model, the clients train the initial parameters using their own local dataset. Usually, in conventional centralised learning, the model would be trained until it converges. However, in federated learning, they are only trained for a specified amount of rounds. This is due to the different limitations each client has on computational resources and communication bandwidth. Moreover, training for too many rounds would result in overfitting the client's data, which could compromise their privacy [6]. Therefore, specifying the number of training rounds strikes a balance between the model convergence and the varying resources of each client.

2.3 Sending model updates

As the clients train the model on their local data, the original model parameters (weights) in each client will start to vary. These unique model updates are then sent to the server. The model updates are either a complete set of weights and biases of the model, or gradients, i.e., the partial derivatives of the model parameter's loss function computed during local training. The type of model updates used depends on what factors are important to take into consideration. Using gradients allow fewer communication costs due to greater compression compared to full model updates [6]. Moreover, gradients preserve privacy better as full model updates are more likely to reveal information about the local model, which sensitive data can be inferred from.

2.4 Aggregating model updates

To update the global model's parameters, we need to combine the results of all local training using the model updates that are sent to the central server through a process known as *aggregation*. The aggregation strategies utilised depend on factors such as the type and distribution of the local data, or whether model updates are gradients or full model parameters. In general, this means using simpler aggregating strategies on gradients and more advanced strategies on full model parameters, due to the need to consider the clients used and the size of their data.

2.5 Iteration

For the final model to converge, federated learning requires multiple rounds of sending the current model parameters, local training, sending the model updates back, and aggregating the model updates. To ensure that the final model achieves consistent accuracy across all client's local data.

2.6 Inherent Challenges in Federated Learning

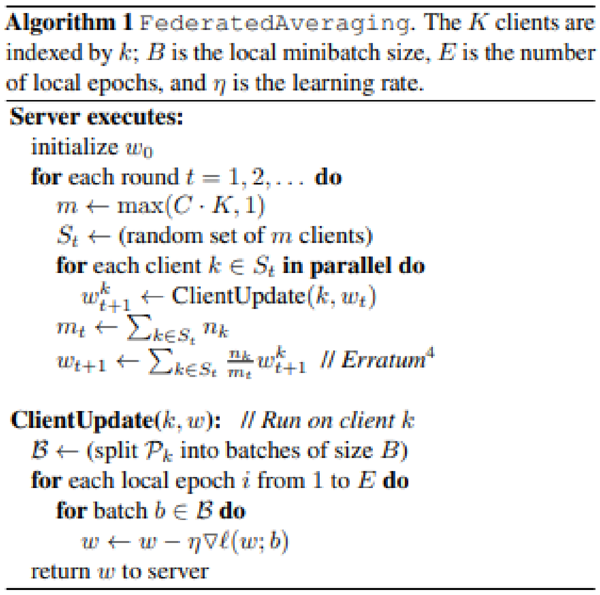
In conventional machine learning, fast communication is inherent due to data centralisation, especially with the use of GPUs. However, federated learning needs to account for the potential bottlenecks posed by the communication of model updates from separate client devices over the Internet. Moreover, not all client devices may be online for training concurrently. This could skew the learning process drastically if too many clients are unavailable. Furthermore, federated learning does not necessarily guarantee data preservation, as local model updates sent may still give context about the client's data. Additional security features such as secure aggregation or differential privacy, are usually required to ensure the protection of data, especially within third-party collaboration [2].

3 Federated Learning Strategies

Given that many different federated learning optimisation strategies have been developed to resolve the limitations of previous strategies. It is necessary to provide an overview of how existing methods work, their differences, and the rationale behind their innovation. This will allow us to identify the strengths and limitations of different existing solutions to make more informed decisions on what strategy to use in the context of NIDS.

3.1 FedAvg

Federated learning was made popular by *Google*, dating back to their earliest foundational works in the paper [7], which introduces the concept of the Federated Averaging (FedAvg) algorithm. The FedAvg algorithm trains a shared model on available local data across multiple clients without the need to transfer their data onto a centralised server. The local models are then sent to the central server, where a global model is produced from the average of the weights of all local models. This averaging is iterated for multiple rounds until the final global model converges. The overall global loss is a weighted average of the losses of all trained clients, and the performance of the model increases as this global loss is minimised. For further clarity, the figure 1 summarises the FedAvg algorithm itself:



Algorithm 1 FederatedAveraging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

```
Server executes:
  initialize $w_0$
  for each round $t = 1, 2, \dots$ do
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of $m$ clients)
    for each client $k \in S_t$ in parallel do
      $w_{t+1}^k \leftarrow$ ClientUpdate($k, w_t$)
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$   // Erratum[4]

ClientUpdate($k, w$):   // Run on client k
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
  for each local epoch $i$ from 1 to $E$ do
    for batch $b \in \mathcal{B}$ do
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return $w$ to server
```

Figure 1: Federated Averaging Algorithm

From Figure 1, we can see that the server executes the training for t number of rounds on a fraction C of the total amount of clients K specified. The weights w_t for the current round are sent to each client k, and the clients run Stochastic Gradient Descent (SGD) for E epochs using their local data. The clients then send the updated weights to the server where they are aggregated, and the process repeats for t rounds.

The FedAvg algorithm assumes all clients would complete E epochs of local SGD. However, some clients may take longer than others, which could increase the time it takes for the model to converge. These clients are known as stragglers, which the algorithm drops if they fail to send updated weights to the server in time. This means that if too many clients are stragglers, the final global model trained is not representative of the total amount of clients, which affects generalisation. Additionally, the FedAvg algorithm weighs the contribution of each client to the global model equally, ignoring the amount of data in each client [7]. This means that clients with more data will have a larger impact on the final model produced compared to clients with less data. This bias causes data imbalances if the clients with the most data are not representative of the overall dataset. Consequently, the final global model will not be able to generalise on new data. Therefore, these limitations are one of many grounds for the development of new federated learning strategies.

3.2 FedProx

FedProx is an algorithm designed with a focus on addressing data imbalances and stragglers in federated learning. The algorithm adds a regularisation (Proximal) term to the loss function, to penalise large changes in weights. This ensures that clients with more data or stragglers contribute less to the global model, in order to mitigate the negative effects data imbalance and stragglers have on the model performance, especially when the data is highly heterogeneous. Additionally, the algorithm further improves the model convergence by allowing more flexibility in the choice of local optimisers and learning rates. This is because the proximal term allows clients to adjust their learning rates and optimisers according to their local data more freely, without significantly affecting the global model [8]. We can examine how the proximal term is computed for the local optimisation function of each client below:

$$\min_{w} h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 .$$

Figure 2: Local Loss Function Minimisation for FedProx

Figure 2 taken from the FedProx optimisation research [8], defines the local optimisation function corresponding to each client k, where w is the global weight of the model, and w_k is the local weights for the current client. The function includes the proximal term added to the client loss $F_k(w)$, which takes the difference between the local updated weight for client k and the model's weight at the current round t. This is to prevent the local models from straying too drastically from the global model during training. The influence of the proximal term is controlled by the $\mu/2$ hyperparameter.

3.3 q-FedAvg

In federated learning, model performance can vary drastically due to highly heterogeneous data, as different data distributions may require different sets of features to accurately represent each data type. Some clients may have more representative data, while other clients have limited and biased data. This could lead to unbalanced improvements in model accuracies across these clients despite good overall accuracy, which led to the development of q-FedAvg [9]. Therefore, the aim of q-FedAvg is to create fairer training by distributing the accuracy evenly across all clients in a more uniform manner.

q-FedAvg improves upon the FedAvg algorithm by additionally accounting for the impact of highly heterogeneous data across clients. Instead of determining the client's weight by the proportion of their data, the algorithm calculates the weighted average sent to the global model with the training performance of each of the clients. The clients that perform worse during training will be penalised more, encouraging the model to specifically improve the performance of these clients. The following figures 3 and 4 proposes the minimisation of the global loss (objective) function $minf(w)$, that is the sum of the weighted local loss functions $F_k(w)$ for all m clients.

$$\min_{w} f(w) = \sum_{k=1}^{m} p_k F_k(w) \,,$$

Figure 3: Global Loss Function Minimisation for FedAvg

$$\min_{w} \; f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w) \,,$$

Figure 4: Global Loss Function Minimisation for q-FedAvg

The term P_k is the weight assigned to a client k, which is based on the client's proportion of data, performance, etc. We can see that q-FedAvg introduces the term $q + 1$ to the original FedAvg function. This adjusts the impact of each client's local loss function based on their performance. The larger the value for q, the more significant the impact of clients with larger loss values on the global loss function [9].

3.4 per-FedAvg

Another paper challenges the issues faced with highly heterogeneous data, through the use of FedAvg with principles of Model-Agnostic Meta-Learning (MAML) to create per-FedAvg [10]. The aim of the personalised federated learning approach is to allow for faster model training convergence while emphasising easier and minimal fine-tuning of the heterogeneous local data distributions across multiple clients. Requiring less data to produce models that better adapt to each client.

The key approach of the algorithm is to train an initial shared model for each client by using a few steps of gradient descent on local data. This is illustrated by the global loss function $F(w)$ to be minimised in figure 5, where n is the total number of clients that are being trained, f_i is the local loss function for client i.

$$\min_{w \in \mathbb{R}^d} F(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w - \alpha \nabla f_i(w)),$$

Figure 5: Global Loss Function Minimisation for per-FedAvg

When directly compared to the global loss function minimisation of FedAvg in figure 3, we can see that the loss of the current weights w for the round is computed after a single step of gradient descent $-\alpha \nabla f_i(w)$. The α denotes the learning rate, and the ∇ denotes the gradient of the local loss function $f_i(w)$.

The objective of the formula is to minimise the average loss after taking a step of gradient descent on each client's local loss function, allowing the shared model to quickly adapt to each client's local data through smaller steps of gradient descent. Consequently, retaining the benefits of federated learning while also providing a more personalised model to better fit the unique characteristics of each client's local data.

4 Methodology

The research methodology for this thesis will be established as a practical applied approach, focusing on the development and optimisation of prototype federated learning models. The study will follow a standard machine-learning workflow, including, but not limited to, collecting the data to be preprocessed, the selection and tuning of model architecture, model training, validation, optimisations, and evaluation of unseen data.

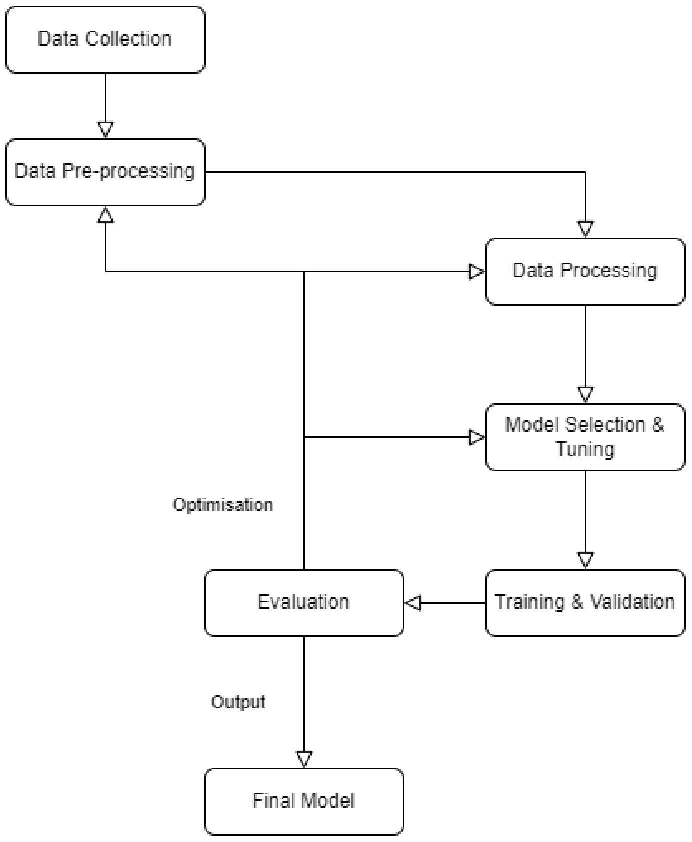


Figure 6: Workflow Diagram

The optimisation process will be inclusive and iterative to achieve optimal performance on the final models chosen for evaluation. This optimisation process involves experimenting and adjusting components of the data pipeline, model selection and architecture, as well as model training. Optimising the data pipeline may involve using different methods of resampling or feature extraction. For the model, hyperparameter tuning or the possible application of regularisation techniques will be considered. Moreover, training optimisation will involve experimenting with a different number of clients, and epochs, while exploring various federated learning strategies. Finally, a thorough experimental setup will be conducted to evaluate the performance and feasibility of the final federated learning models, in comparison to a baseline centralised learning model. Additionally, the results from using different federated learning strategies and client scaling will also be evaluated. This will identify and highlight the strengths and potential limitations associated with federated learning in classifying network attacks.

4.1 Data Collection

The dataset for this study is gathered from a novel real-time Network Intrusion Detection System (NIDS) research conducted by Mirsky et al [11]. It consists of nine different types of network attacks in the form of millions of network packets (see Figure E in Appendix E). These packets were captured using network attacks launched on either a network of IoT devices or an IP-based commercial surveillance system. In each dataset, the initial million packets captured represent benign behaviour captured prior to the execution of each type of network attack. Each of the nine network attack datasets includes a corresponding label file in CSV format, which indicates whether each captured network packet is benign or malicious. The network packets are available in either their raw original network captures in PCAP format or as a preprocessed CSV file. The preprocessed file is produced by extracting 115 features from raw network packets using the study's $AfterImage$ extractor, developed to enhance the efficiency of NIDSs. The 115 features essentially create a statistical representation of the packet currently in transit through the network, which includes context about the host activities and network behaviours [12]. This preprocessed data will be used instead of the raw network packets to facilitate supervised learning for network packet classification. This will allow the primary focus to be on essential aspects of federated learning instead of extensive preprocessing.

The dataset of all nine network attacks will all be utilised due to their quality and extensive coverage, which allows federated learning to accurately learn from a diverse range of patterns. Potentially leading to more effective generalisation over a broader spectrum of network intrusions. The simulation of non-independent and identically distributed (non-IID) data training is crucial in this study, as it allows the resilience of federated learning algorithms to be evaluated similarly to how data is distributed across devices in real-world scenarios. By aggregating nine separate datasets that are collected over different time periods, we effectively introduce non-IID characteristics into the data. These characteristics emulate data that often occurs in practice when collected from numerous devices over time. Moreover, the dataset's representation of real-world scenarios through its method of capture enhances the relevancy and practicability of the federated learning model's predictions for NIDs on IoT devices. Additionally, the large volume of network packets in the dataset ensures a substantial amount of normal and anomalous network packet data for the model to train, which is essential for accurately differentiating between benign and malicious network packets.

4.2 Data Processing

Cleaning. The data will have missing values and duplicates removed. Duplicates could possibly be helpful in identifying common patterns and repeated attacks, but can also impact the accuracy of the model and introduce bias to the data distribution in other cases. Since the objective is to classify network attacks, complete duplicates should be removed to avoid overrepresentation and ensure unique instances on the same network attack.

Splitting. Splitting the data into training, validation, and test sets before scaling them will ensure that any scaling or feature selection done is based on the training set alone. This prevents information leakage into the validation and test set, which could influence the model's performance and lead to overly optimistic results. Maintaining the sequence of data samples is vital for a time series analysis, as the chronological order holds significant information. Therefore, the data won't be shuffled, to prevent introducing bias and inaccurate model performance evaluations from breaking the temporal order.

Scaling. Although the dataset used in this study are extracted using the same feature extraction framework, it is likely that the features are in different scales. Therefore, the data will be scaled before any other preprocessing steps, including resampling. This will ensure that the scales are uniform, preserving the consistency of parameters such as mean and standard deviation across all data samples. This is especially important for imbalanced class distributions, as it will prevent samples of minority classes from being scaled differently, such as when resampling techniques are applied.

Due to the dataset containing many different ranges and variances, standardisation may be the most optimal method of scaling. Using normalisation may not be as effective, as the features are not constrained to a specific range with a natural maximum and minimum value. The loss of important information could also be possible by normalising the data. Moreover, normalisation does not guarantee zero mean and unit variance for the features like standardisation, and is sensitive to outliers.

Resampling. The marginal benefit of adding more labelled data will eventually decrease to a point of diminishing returns. Therefore, it is wasteful and computationally inefficient to use an excessive amount of labelled data on overrepresented classes. The diversity of the data is more important while maintaining enough data to train well. Therefore, it may be necessary to truncate larger network attack datasets. While resampling methods can address this, it is important to consider what method to use, as some methods can break the temporal order of the dataset from assuming that the data are independent and identically distributed. This is generally from resampling methods that involve random selection and placement or removing samples, such as oversampling, undersampling, SMOTE, etc.

Partitioning. To allow for a fair comparison between both federated learning and centralised learning, the same full dataset will be readily available to both learning approaches. This will be ensured in federated learning, by partitioning the overall dataset based on the number of clients used in training.

4.3 Neural Network Architecture

A Long Short-Term Memory (LTSM) architecture will be used to address the research question. LTSM is a type of recurrent neural network (RNN) that works exceptionally well at learning long-term dependencies from tasks that involve sequential data. This is due to their ability to prevent the vanishing gradient problem [13].

The primary objective of the study is not to develop and test complex models but rather to investigate federated learning. Therefore, a simple model architecture will be used, to ensure that the effective base capabilities are included while allowing the primary focus to remain on federated learning. The study's use of virtual clients could mean that computational resources may be limited. Using a simpler model architecture will reduce the computational load, ensuring that problems won't arise from exhausting the available resources. Additionally, using a simpler model will make it clear that any improvements come from federated learning and not the complexity of the model itself. This will allow better evaluation and understanding of the benefits of using federated learning in the study.

4.4 Training & Validation

While the validation and test sets serve similar purposes of evaluating the model on unseen data, they are used at different stages of the model development process. The validation set is used during the training phase to tune the model and make decisions about its learning process, while the test set is used on the final trained models to get an unbiased performance estimate. Furthermore, the validation process will take place on each client locally to yield more realistic and representative results, as this is often the case in real scenarios. However, unstable validation results should be expected, as it is possible for the validation results to change over time if clients are unavailable.

4.5 Federated Learning

To allow for federated learning, evaluation, and analytics, Flower will be used to facilitate the movement of machine learning models, local training and evaluation data, and the aggregation of updated models [14]. Flower provides a framework with a focus on simple, scalable, and secure federated learning. The framework allows federated learning using any machine learning framework and programming language due to its emphasis on a unified and agnostic approach to federated learning, evaluation, and analytics. It will allow the study to implement different federated learning strategies with ease, which will make it efficient to compare and evaluate their performances [15].

4.5.1 Strategies

The implementation is explored with the strategies, federated averaging (FedAvg), adaptive federated optimisation using Adagrad (FedAdagrad), and federated proximal (FedProx). The FedAvg strategy is chosen to provide a simple yet effective baseline for starting, while FedAdagrad and FedProx are implemented explicitly to evaluate their training performance using non-IID data. FedAdagrad allows learning rates to adapt to the model parameters by performing larger updates for parameters associated with infrequent features and smaller updates for parameters associated with frequent features.

4.5.2 Virtual Engine Client

The Flower framework's ability to simulate virtual clients is vital to this study, where there is a lack of sufficient physical devices to emulate federated learning accurately. When simulating multiple client instances for federated learning, utilising a single machine can quickly exhaust available memory. To manage memory efficiently, the framework allows client instances to be created only when they are required for training and validation to prevent memory overload, especially on a single machine. Although the Framework uses hardware capabilities efficiently such as parallelising client tasks, the federated learning simulation is still tied to the computational resources of the host machine. This means that the speed, efficiency, or client scaling of the simulation could still be impacted. There are also significant limitations that need to be acknowledged in authentically replicating real-world federated learning complexities. Unlike physical devices, virtual clients are homogeneous and do not fully reflect the diversity of a federated learning scenario. Factors in physical devices such as varying computational capabilities, network conditions, user behaviours, or availability cannot be accounted for when evaluating virtual clients. Moreover, the communication of virtual clients with the server does not reflect real-world network latencies.

5 Implementation

5.1 Programming Tools & Platforms

The implementation is carried out on a cloud-based IDE, specifically Google Collab for Python development. It was selected for its robust computational resources to reduce the computational constraints of running federated learning. Additionally, its integration with Google Drive allowed for easier saving and loading of data. PyTorch was the chosen machine learning framework, due to previous experience, to facilitate a more seamless federated learning implementation.

5.2 Data Preprocessing

Dataset Creation. The CSV file for the network packets and labels are gathered from their URLs [12]. Before the network packets are read using the Python library *pandas*, the first one million rows of benign network packets are skipped in every network attack type except Mirai Botnet. This is to reduce the excruciatingly slow processing of large file sizes and manage the over-representation of benign network packets. The library $fastml$ will also be used to call a function that reduces memory usage every time a CSV file is read or saved. This will reduce the memory footprint of the data, to speed up computations performed on large file sizes. Each row of the network packet is then concatenated with its corresponding labels, and the labels are mapped to their classes using a predefined dictionary. This represents 10 classes in total, 9 types of network attacks and benign networks. The data is cleaned by dropping any exact duplicates and samples with empty values.

Altogether, the creation and saving of each network attack dataset are done in parallel using a thread pool executor. This is to further optimise the time taken to process such large file sizes. From examining the distribution of classes within the training set, it becomes clear that the samples of benign classes significantly outnumber other classes, resulting in severe under-representation in minority classes (see Figure 11 in Appendix A). To address the imbalance, benign network packets are further dropped from each network attack-type dataset, taking care not to remove malicious packets by accident. Finally, a single dataset of all network attacks is made by concatenating each of the nine network attack datasets.

Splitting. The machine learning library *scikitlearn* is used to split the dataset into the training, validation, and test subsets with stratified sampling. The training and test subsets split were created from the full dataset, while the validation set was created from the training set. To strike a balance between computational efficiency and ensuring data for validation, an 80:20 split was used for the training and validation subsets. Additionally, the creation of the subsets is done with a random state parameter, which generates a seed to ensure that the same sequence of randomness is generated every time the function is called. This is to allow the results to be reproducible, which will aid in comparing, testing, and debugging.

Feature Reduction. Using the $fast - ml$ library, constant and quasi-constant features are selected to be removed from the dataset. Features that are constant have the same value across all samples, whereas quasi-constant features have nearly the same value across samples. To reduce feature dimensionality they are removed from the dataset as they provide little to no meaningful information for training. This will prevent wasting computational resources from factoring in these features and improve computational efficiency. The effects of this will be measured empirically to see whether it makes a significant difference.

Scaling. The *scikitlearn* library is used to scale the datasets, testing the effects of both normalisation and standardisation on the performance of the model. The scalers are fit exclusively on the training set, to prevent information leakage and overly optimistic results on the validation and test set.

Resampling. In an attempt to improve model performance by addressing the imbalanced class distribution, re-sampling methods are applied through the $imbalanced - learn$ library. Random under-sampling is applied to the benign network class to reduce its over-representation by a specified amount relative to the other classes. To maintain the diversity of the resampled dataset, the method is specified to only remove the same instance once. Synthetic Minority Over-sampling (SMOTE) is applied to increase samples from three of the most under-represented classes by a specified amount. A random state is specified for the resampling methods to maintain the same resampled dataset every creation, which ensures reproducible results. To ensure the model is evaluated on unseen data and prevent data leakage, the under-sampling is done only on the training set. Additionally, the dataset should be scaled beforehand to ensure that under-sampling is applied uniformly across all features.

Feature Selection. Feature selection is performed on the dataset with the use of ANOVA F-test scoring and SelectKBest from the $scikit-learn$ library. The ANOVA F-test score is calculated for all features of the dataset, and the results are then plotted on a bar graph using the *matplotlib* library. This helps identify the number of features with the highest scores to determine the most important features for the model. The training set is then used to train a Decision Tree Classifier. To find the optimal number of top features for the final model, the classifier is trained for multiple numbers of top features k selected through SelectKBest. Through using the test set, the accuracy of using different k top features is evaluated. Finally, a graph of the classifier's accuracy is plotted against each k value, to determine the best number of features to use. The feature selection is implemented after scaling the data, as some feature selection methods rely on the use of scaled data. This ensures that the selected features will be of the same scale and range.

5.3 Data Partitioning & Batching

To simulate the effects of using local data from each device for federated learning, a function is defined to partition the training and validation dataset for distribution using Python $NumPy$ array functions. The partitions created depend on the number of clients available to ensure all data is used. Additionally, a function is defined to test the effects of an extreme case of non-IID data where each client represents one class. This is 10 clients, representing the 10 classes in total. Using a custom-defined $PyTorch$ class, each numpy array dataset is converted into PyTorch tensors. These tensorised datasets are then batched and loaded into memory in the form of PyTorch DataLoaders for the training process. For the centralised learning baseline, non-partitioned training and validation datasets are batched and loaded as well.

5.4 Neural Network Architecture

A simple LTSM model is originally implemented using the PyTorch library. The forward pass is defined to initialise the hidden states and cell states, process input through an LTSM layer, and then through two fully connected layers. In each layer, a rectified linear unit activation function (ReLU) is applied to introduce non-linearity. A dropout layer is applied between the two fully connected layers for regularisation, to prevent the model from overfitting the data. The sequence length is not specified for this LTSM implementation to give flexibility in processing different sequence lengths without needing to modify the model. Following the subsequent disruption of the dataset's temporal order, a deep neural network (DNN) model is implemented to process the data in a non-sequential approach. In the forward pass function defined, the input data is flattened and propagated through an input layer, a specified number of layers, and two fully connected layers. Additionally, ReLU activation functions and dropout layers are also applied between the layers.

5.5 Training & Validation

Using the PyTorch library, a train and test function is defined to train and evaluate the model using the dataloaders created. Both functions use cross-entropy to measure the loss, but the Adam optimiser is used only on the train function to adjust the model parameters. The Adam optimiser is used for its popularity in balancing the trade-off between efficiency and low memory requirement. For each batch of the dataloader, the gradient values are set to zero, the forward pass computes an output for the model, and the loss is calculated. In the train function, a backward pass is additionally performed to compute gradients for updating the model parameters. A running total loss and the number of correctly classified samples are tracked for each epoch to compute the overall output loss and accuracy for both functions. A variable is set to perform the computations on an NVIDIA GPU if it is available instead of the CPU. This is to utilise CUDA's parallel computing feature on NVIDIA GPUs to significantly speed up computations.

5.6 Centralised Learning

Using the same model initialisation parameters as with federated learning, the model is trained for a specified number of epochs on the full available dataset. For each epoch, the model trains and evaluates the full training and validation data respectively. The losses and accuracies are computed and printed for both the training and evaluation for each epoch. They are then stored in arrays to plot graphs for the loss and accuracy of the training and evaluation over epochs. This allows performance over epochs to be visualised to detect potential overfitting or underfitting.

5.7 Federated Learning

5.7.1 Model Parameter Updates

For client devices participating in training, a function is defined to get model parameter updates from the state dictionary of a client's local model and return them as numpy arrays. This dictionary object contains model layers and their associated parameter values (weights) and is sent to the central server for aggregation. Another function is then defined for clients to update their local model with received parameter updates from the server. This is through pairing each key in the model's current state dictionary with corresponding new parameters from the numpy arrays and loading the updated state dictionary into the model.

5.7.2 Federated Training & Evaluation

Using the Flower framework, the behaviour of a single client participating in federated learning is implemented in a class. The class provides the previously defined function to get model parameter updates and update model parameters within a model fitting and evaluation function. Each client is initialised with a unique ID, local model, training data, and validation data. The model fitting function updates the local model with the received model parameter updates, performs local training, and returns the result along with the number of samples used for training. The model evaluation function uses the received model parameter updates to evaluate using the validation data to return a loss, accuracy, and the number of samples used for evaluation. Finally, a function is defined to create instances of the client class, with a client ID, initialised model, and their respective data. Each model and data are unique to the client, to simulate the decentralised property of federated learning. To prevent exhausting memory, the Flower framework allows instances of the client to be created only when they are necessary for training or evaluation and discards them after. To fairly compare the performance of federated learning to centralised learning, the visualisation of the federated training and validation metrics is not necessary. Both models will be trained for the same number of epochs to ensure a consistent and unbiased experimental setup. It is important to note that in terms of federated learning, an epoch would be a round of communication between the server and the clients, instead of a full pass through an entire dataset as with centralised learning.

5.7.3 Tracking Metrics

The Flower framework can automatically aggregate the losses of each client, but custom metrics such as accuracy need to be manually aggregated. The federated learning strategy would then be configured to call this custom aggregation function when metrics are received from clients. The function is defined to aggregate the accuracy metric from participating clients. A weighted average is calculated by taking a list of tuples containing the number of samples and the metrics (dictionary) and summing the products to divide against the total number of samples. This allows clients that trained on more samples to have greater influence over the accuracy.

5.7.4 Federated Learning Strategy

The Flower framework allows FedAvg, FedAdagrad, and FedProx to be implemented as a built-in class. Initial model parameters are directly implemented into the federated learning strategy to enable server-side parameter initialisation. This prevents the Flower framework's default action of randomly initialising the parameters using a random client. The strategy used is customised to take the instantiation of a model as input, to save the model updates after each epoch of training. This is done by inheriting the parent strategy class, where a method is overridden to save the model on the server side after the results of training are received from all participating clients. Finally, the strategy used is customised to use a specified number of clients for fitting and evaluation, and the number of available clients required to start federated learning.

5.7.5 Simulation

To start the federated learning simulation a function is called with various arguments. The arguments include the function to create client instances, the number of clients used in the simulation, the number of federated learning epochs, and the federated learning strategy used. Additionally, if a CUDA device is available, the virtual client resources will be specified to include the use of a GPU.

5.8 Collecting Evaluation Results

The evaluation records the performance of all of the trained federated learning models and the baseline centralised learning model on the unseen test data. The previously defined test function is modified to also track the true class and the class predicted by the model. Using the $scikit-learn$ library, the tracked classes are then used to calculate the precision, recall, and F1 score. Additionally, the *matplotlib* and *seaborn* libraries are used alongside scikit-learn to create a confusion matrix. The seaborn library allows the prediction performance of the model on each class to be visualised using a heatmap function.

6 Experimentation & Optimisation

This section covers any changes made and testing carried out across various areas of the machine learning workflow. The main objective was to maintain implementations that gave the most balanced trade-off between memory efficiency and training performance. This is because memory constraints became a significant issue when running federated learning, causing memory exhaustion errors and crashing the development environment. Additionally, modifications to the implementation are also made to maintain an unbiased evaluation between federated learning and centralised learning on the unseen test data.

6.1 Data Processing

Splitting. The initial strategy was to manually split the dataset into training, validation and test sets, to preserve their original temporal order. However, the approach led to an imbalanced underlying class distribution across the subsets, as each network attack dataset was stacked one after another when creating the dataset. This led to biased model evaluations and overfitting. Moreover, the model may learn misleading patterns from the connecting packets of different time periods. Ultimately, degrading performance by disrupting the model's ability to learn correct sequential dependencies.

Due to how challenging it was to address the class imbalance across the subsets while preserving the temporal order, stratified sampling was used as an alternative solution. The technique selects samples in proportion to their importance from subgroups, created by dividing the dataset based on characteristics. This allowed each subset of training, validation, and test data to contain a representative proportion of each network attack type, at the cost of breaking the temporal order of the data. Given the primary objective is a classification task and the negative impact of imbalanced classes is more severe than preserving the data's temporal order, a compromise was made. Additionally, subsets were created using 10% of the overall dataset to fit the resource constraints. This alleviated the slow training times and reduced errors from memory exhaustion when running federated learning.

Scaling. Through experimenting with different combinations of normalisers and standardisers, it is observed that scaling only with the normaliser yielded the most model accuracy. Therefore, the normaliser is used throughout the implementation and used for the final evaluation.

Resampling. In an attempt to increase model performance, both random undersampling and SMOTE were originally considered for the implementation to address the severely imbalanced class distribution. However, the model's performance during training has significantly degraded when experimenting with the combined use of both methods and individually. This could be due to the loss of valuable information in the major class from using undersampling and the creation of noise from using SMOTE. Therefore, both resampling methods were excluded from the final implementation. The class imbalance is not further addressed, since the main objective is to compare how federated learning handles non-IID data compared to centralised learning. This meant that the comparison of results will rely heavily on the proposed evaluation metrics to address the class imbalance.

Feature Reduction. In federated learning, the size of model updates that clients communicate to the central server depends on the number of features the model has to process from the data given. Feature reduction can reduce the size of these model updates which in turn reduces the memory load required to store them. Therefore, implementing feature selection is necessary to reduce the memory footprint of the high-dimensional dataset to resolve issues with memory constraints.

From implementing the removal of constant and quasi-constant features, it was observed that it did not have a significant effect on the model's accuracy. Therefore, the removal is retained in the implementation due to its advantage in reducing the memory requirements of federated learning. To further address the memory constraints and potentially improve training performance by reducing noise, feature selection methods are explored to remove trivial features. Due to the large size of the dataset, experimenting with scoring methods such as mutual information took too much time. Therefore, the ANOVA F-test scoring method was chosen for simplicity and efficiency, to select the top k most informative features from the dataset. Additionally, the issue with slow training times was also why the decision tree classifier is chosen over other tested methods, such as random forest, to evaluate the training performance of using different k features.

The ANOVA F-test scoring of all features in figure 7, showed that a majority of the features did not have a strong association with the target class. Moreover, the results of evaluating different k features on the test set in figure 8, show that the increase in accuracy was very minimal above using more than 50 features. Therefore, the rest of the implementation tested the feature selection of only 50 features.

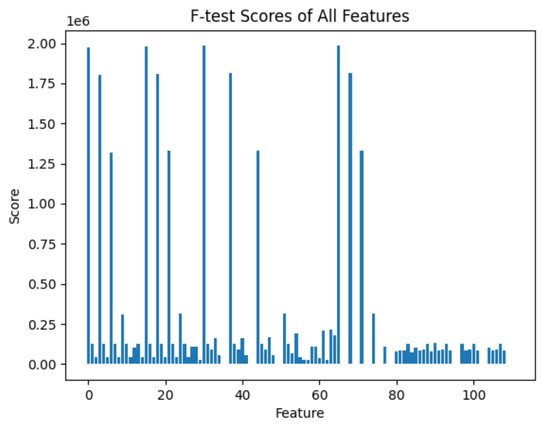


Figure 7: ANOVA F-test Score of All Features

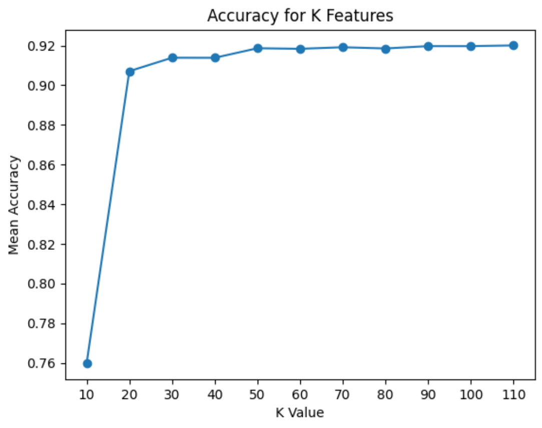


Figure 8: Test Set Accuracy for k Features

During both federated and centralised learning, it was discovered that the model performance when using feature-selected datasets was worse than using all features. However, the difference in performance loss was very minimal. When considering that the benefits of reducing the memory footprint far outweighed the insignificant loss in performance, the feature-selecting process is included for the final chosen model.

Partitioning & Batching. When attempting federated learning with a highly non-IID dataset where each client represented a class, the training performance was significantly reduced. Therefore, the final implementation partitions the overall dataset based on the number of clients participating to retain better model performance. Due to errors from memory exhaustion, a batch size of 16 is used instead of the default value of 32, at the cost of increasing training time.

6.2 Neural Network Architecture

Initially, an LTSM model was implemented to consider the sequence length of the data for capturing temporal dependencies. However, the data was no longer in temporal order and the LTSM's complexity was too high for the given memory constraints of the experiment. Therefore, a DNN model was implemented as an alternative with a simpler architecture and better computational efficiency. This allowed the experiment to maintain reliable results while handling the memory constraints. Additionally, the initial model parameters of the DNN were also adjusted since memory limitations were still an issue. This allowed for the training process to achieve similar accuracy and maintain lower memory consumption, while also reducing training time.

6.3 Training & Validation

After the initial training with 20 epochs, the results in figure 9 and 10 show a minimal decrease in loss and an increase in accuracy after 10 epochs. Therefore, training for both models was capped at 10 epochs to reduce training times and save resources. Moreover, the results do not show any occurrence of underfitting and overfitting. The validation loss and accuracy fluctuated throughout the training process but were generally similar to the training loss and accuracy. This indicates that the model is generalising well to unseen data.

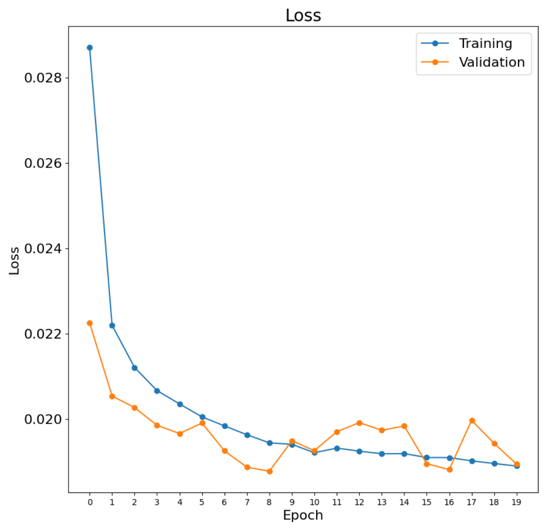


Figure 9: Loss Over Epoch

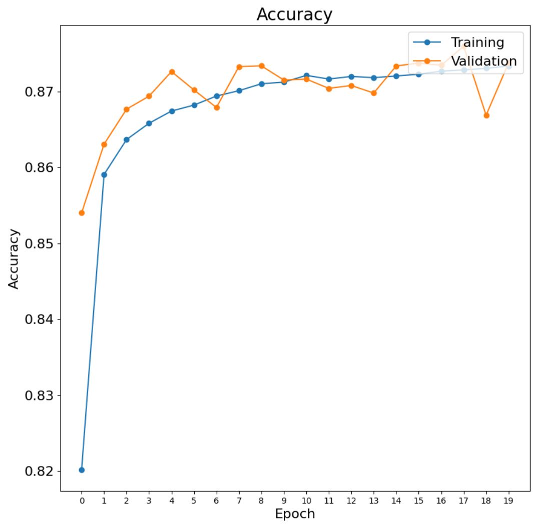


Figure 10: Accuracy Over Epoch

Regularisation Techniques. In the initial implementation, regularisation techniques such as dropout and L2 regularisation were implemented to prevent overfitting from occurring during training. However, it was necessary to simplify the model complexity over the course of the implementation due to memory constraints. Therefore, the use of regularisation techniques was ultimately dropped, since overfitting did not occur after the model simplification.

6.4 Virtual Client Scalability

When running Federated Learning using the number of clients recommended by the Flower framework, the development environment crashes due to memory exhaustion. Therefore, 4 clients were used for training and validation initially. However, despite efforts to improve memory efficiency, the development environment still crashes when scaling clients with all features of the data. Therefore, the impact of scaling the number of clients is tested on the feature-selected data, while maintaining optimisations to the implementation in addressing the memory constraints. As a result, the training performance of federated learning could be tested with more clients, staying within the memory limit. Nonetheless, FedAdagrad was ultimately excluded when testing the scaling performance since the development environment crashes from memory exhaustion when utilising more virtual clients for the strategy.

During the training process, it was discovered in both FedAvg and FedProx that the training loss increases and the training accuracy decrease as the number of clients used for training increases. This implies that the performance of the learning strategies diminishes as we scale up the number of clients. The performance degradation is most likely caused by the termination of client tasks due to memory exhaustion, which leads to disruptions in the learning process such as the loss of local data and computations, or unexpected system behaviours. This may have led to inconsistent or incomplete results that negatively affect the model's classification performance. Consequently, exploring the scalability of federated learning had serious limitations under these circumstances.

7 Evaluation

During the evaluation phase, all model evaluations will be performed on unseen test data. The evaluation will be performed on both the feature-selected dataset and also the dataset with all features, to assess whether discrepancies come from the feature selection or the strategies themselves. To facilitate the comparison of the implemented federated learning strategies, all models under evaluation are trained using four clients. This is due to the memory constraints of training FedAdagrad with more clients. The implemented federated learning strategies will be compared to determine the model that yields the best performance in our setup. This comparison aims to find the most effective model for the specific characteristics of the dataset used. The centralised learning model will then serve as a benchmark against which we compare the federated learning models. Centralised learning provides a reference point to measure the performance of federated learning, as it leverages all data directly without partitioning. Comparing federated learning to this will allow us to better understand the trade-offs between the two approaches more thoroughly. Additionally, it may highlight situations where federated learning may be more beneficial, such as in handling class imbalance with distributed datasets.

7.1 Evaluation Metrics

For evaluation of the unseen test dataset, accuracy will be used to measure the model's overall performance, by calculating the total correct predictions made by the model from the total number of predictions performed. However, measuring solely the accuracy is not enough, especially since the classification involved imbalanced datasets. If the model correctly predicts a class that contains the most samples in the dataset, it will achieve high accuracy despite predicting other classes wrong. Therefore, metrics such as precision, recall, and F1 score will be used to thoroughly evaluate the classification of each class. This will ensure unbiased and comprehensive comparisons between the learning strategies.

Precision measures how much of the predictions made are actually correct, by calculating the number of true positives over the total number of positive instances predicted, which includes both true and false positives. Recall measures the number of actual instances the model correctly identified, by calculating the number of true positives divided by the total actual instances, which includes true positives and false negatives. In general, ensuring high precision where the model correctly identifies true positives is desirable when false positives are costly. Conversely, high recall from ensuring the model identifies more positive instances is more suited when false negatives are more costly [16]. In terms of classifying network packets for NIDS, emphasis on either recall or precision will depend on the consequences of misclassification. Focusing on higher recall is necessary if the emphasis is on the repercussion of failing to detect actual network attacks. This means correctly identifying as many attacks as possible, at the cost of producing false positive instances. Contrarily, high precision would be more desirable if the objective is to reduce resource waste on incorrect alerts from false positive instances and prevent disruptions of legitimate network traffic. If the dataset used has low noise and distinct differences between classes, the model may potentially achieve both high precision and recall. If data patterns are clearer, this will allow more accurate learning and generalisation to enhance detection accuracy and reduce false alarms.

Additionally, it is important to be mindful of the inherent trade-off between precision and recall since optimising a model to maximise one may often decrease the other metric [17]. This is because of the different focuses of the metrics, if precision aims to minimise false positives, it leads to increased false negatives and lower recall. Conversely, recall's aim to capture all actual positives risks more false positives and lower precision. Therefore, to give significance to both false positives and false negatives, the F1 score will be used to measure the harmonic mean of precision and recall. Thus, ensuring the balance between optimising precision and recall.

7.2 Overall Performance of All Learning Strategies

The results in Table 1 show the overall accuracy, and the macro-averages of all precision, recall, and F1 score of each class. The use of macro-averages over weighted averages is necessary to treat all classes independently before the averaging to ensure minority classes are not underrepresented. This provides a less biased performance measure where the results will not lean towards the class with the higher number of instances in the imbalanced dataset.

Table 1: Test Set Results of The Learning Strategies

Learning Strategy	Accuracy	Precision	Recall	F1 Score
		All Features		
Centralised	0.878	0.82	0.71	0.73
FedAvg	0.876	0.83	0.78	0.78
FedAdagrad	0.598	0.65	0.63	0.53
FedProx	0.876	0.84	0.78	0.78
		Feature Selection		
Centralised	0.875	0.83	0.75	0.77
FedAvg	0.873	0.83	0.76	0.77
FedAdagrad	0.580	0.50	0.50	0.44
FedProx	0.874	0.82	0.75	0.76

The results show that FedAdagrad performed noticeably worse than the other learning strategies in all of the metrics for both the feature-selected and non feature-selected datasets. The centralised learning strategy achieved the best accuracy among all of the other strategies for both data scenarios, which implies that it is the best at accurately classifying most of the instances. Nevertheless, this performance difference was only about 0.2% at most for the dataset with all features. The results also show that centralised learning has similar precision, recall, and F1 score to FedAvg and FedProx for the feature-selected dataset. This suggests that FedAvg and FedProx are viable alternatives to the centralised approach.

However, it appears that FedAvg and FedProx performed better than centralised learning on the F1 score using the non feature-selected dataset. From closer inspection, we can see that this is due to both higher precision and recall performance than centralised learning. This suggests that FedAvg and FedProx were better than centralised learning at balancing the minimisation of false positives while accurately classifying a large portion of the instances. Consequently, this implies that centralised learning may have achieved higher accuracy than FedAvg and FedProx due to its better performance in classifying classes with the most instances in the dataset. Moreover, it suggests that FedAvg and FedProx were better than centralised learning at classifying instances of the minority classes.

Both FedAvg and FedProx seem to perform almost identically with the non feature-selected dataset and very similar with the feature-selected dataset. Although, FedAvg seems to perform better on precision, recall, and F1 score than FedProx for the feature-selected dataset, this difference is only 0.1% with FedProx having the higher overall accuracy. Therefore, the difference in performance between FedAvg and FedProx is too minimal to draw definitive conclusions on which of the two achieved better performance when evaluating the overall performance.

Overall, the decrease in performance from using the feature-selected dataset instead of all features is relatively negligible in all results of the learning strategies. This implies that the selected features correctly captured most of the relevant information.

7.3 Class Performance Comparison of FedAvg & FedProx

To evaluate whether FedAvg or FedProx performs better, the precision, recall, and F1 score of both strategies will be compared for each class, with the feature-selected dataset and the dataset with all features. This will allow us to evaluate their performance on the class-imbalanced dataset with greater depth. The results shown in the tables are ordered by most instances for each class, which allows us to see trends between the results of the majority classes and minority classes. The support is also shown, which measures the number of instances of each class within the test dataset.

7.3.1 Performance on Feature-Selected Dataset

Table 2: Class Performance Metrics of FedAvg (Feature-Selection)

Class	Precision	Recall	F1 Score	Support
Benign	0.79	0.87	0.83	50928
SSDP Flood	0.99	1.00	0.99	28792
ARP MitM	0.91	0.68	0.78	22905
Active Wiretap	0.94	0.97	0.96	18464
Mirai	1.00	0.89	0.94	12850
Fuzzing	0.81	0.90	0.85	8656
Video Injection	0.82	0.56	0.67	2050
SSL Renegotiation	0.67	0.58	0.62	1853
OS Scan	0.49	0.94	0.65	1313
SYN Dos	0.89	0.23	0.37	141

Table 3: Class Performance Metrics of FedProx (Feature-Selection)

Class	Precision	Recall	F1 Score	Support
Benign	0.79	0.87	0.83	50928
SSDP Flood	0.99	1.00	1.00	28792
ARP MitM	0.91	0.68	0.78	22905
Active Wiretap	0.94	0.98	0.96	18464
Mirai	1.00	0.89	0.94	12850
Fuzzing	0.81	0.90	0.85	8656
Video Injection	0.87	0.52	0.65	2050
SSL Renegotiation	0.67	0.58	0.62	1853
OS Scan	0.50	0.88	0.63	1313
SYN Dos	0.69	0.23	0.35	141

We can see from Table 2 and 3 that both FedAvg and FedProx performed almost similarly in classifying most of the classes. Both strategies achieved relatively high precision, recall, and F1 score in the classes Benign, SSDP Flood, Active Wiretap, Mirai, and Fuzzing. However, the strategies demonstrated varying performance levels, mostly across the underrepresented classes. In the Video Injection class, FedProx outperforms FedAvg in precision but exhibits a lower recall value, which indicates that a large number of Video Injection instances were missed. Similarly for the OS Scan class, FedProx slightly outperforms FedAvg in precision which implies fewer false positives but also displays a lower recall value. Regarding the SYN Dos class, FedAvg seems to significantly perform better than FedProx in precision. However, the performance of both strategies on SYN DoS is significantly worse than all of the other classes altogether. Moreover, the precision, recall, and F1 score seem to decrease the more underrepresented the classes are. This suggests that the strategies struggle to classify instances of minority classes. Overall, FedAvg seems to achieve higher F1 scores than FedProx on the minority classes. This suggests that FedAvg is the most effective at balancing the accurate classification of minority classes over a larger portion of actual instances of those classes when using the feature-selected dataset. For a detailed visualisation of these performance results, see the confusion matrices Figure 12 and 13 in Appendix B.

7.3.2 Performance on Non Feature-Selected Dataset

Table 4: Class Performance Metrics of FedAvg (All Features)

Class	Precision	Recall	F1 Score	Support
Benign	0.79	0.87	0.83	50928
SSDP Flood	0.99	1.00	1.00	28792
ARP MitM	0.91	0.68	0.78	22905
Active Wiretap	0.94	0.98	0.96	18464
Mirai	1.00	0.89	0.94	12850
Fuzzing	0.81	0.90	0.85	8656
Video Injection	0.89	0.65	0.75	2050
SSL Renegotiation	0.66	0.66	0.66	1853
OS Scan	0.49	0.93	0.65	1313
SYN Dos	0.85	0.23	0.37	141

Table 5: Class Performance Metrics of FedProx (All Features)

Class	Precision	Recall	F1 Score	Support
Benign	0.79	0.87	0.83	50928
SSDP Flood	0.99	1.00	1.00	28792
ARP MitM	0.91	0.68	0.78	22905
Active Wiretap	0.94	0.97	0.95	18464
Mirai	1.00	0.89	0.94	12850
Fuzzing	0.81	0.90	0.85	8656
Video Injection	0.88	0.74	0.80	2050
SSL Renegotiation	0.67	0.59	0.63	1853
OS Scan	0.49	0.92	0.64	1313
SYN Dos	0.90	0.26	0.41	141

From the results in Table 4 and 5 both FedAvg and FedProx seem to perform similarly in classifying the same classes as the results on the feature-selected dataset, still achieving the same high scores. However, the difference in the performance of both strategies in the minority classes varies more than with the feature-selected dataset. This does not include the OS Scan class, which shows to perform similarly using both FedAvg and FedProx. FedProx achieves a significantly higher F1 score than FedAvg for the Video Injection and SYN Dos classes, whereas FedAvg achieves a better F1 score on SSL Renegotiation. The higher F1 score performance using FedProx appears to be influenced by an increase in both the precision and recall in the Video

Injection and SYN Dos classes. In contrast, FedAvg outperforms FedProx in terms of F1 score for the SSL Renegotiation class. This seems to be due to a higher recall, which indicates FedAvg is better at identifying more actual instances of the class than FedProx. Nonetheless, it seems that FedProx has a better overall performance in classifying minority classes than FedAvg when feature selection is not considered. Although both of the strategies have better performance on the underrepresented classes using the dataset without feature selection, they still do not perform as well as classifying overrepresented classes. For a detailed visualisation of these performance results, see the confusion matrices Figure 14 and 15 in Appendix C.

In summary, the performance of FedAvg and FedProx appears to be comparable after all, despite their similarity in results. It is demonstrated by the results that whether FedAvg or FedProx performs better depends on if the dataset is feature selected or not. With all features considered, FedProx seems to outperform FedAvg in the minority classes. However, this trend seems to disappear when the dataset is feature-selected, where FedAvg performs better on the minority classes. In addition, it is discovered that the lower performance in the minority classes is the main factor responsible for holding back the overall accuracy from reaching higher levels. This implies that there are still limits to how well both federated learning strategies perform on the imbalanced dataset.

7.4 Class Performance of Centralised Learning

Table 6: Class Performance Metrics of Centralised Learning (Feature-Selection)

Class	Precision	Recall	F1 Score	Support
Benign	0.8	0.86	0.83	50928
SSDP Flood	0.99	1.00	1.00	28792
ARP MitM	0.89	0.72	0.80	22905
Active Wiretap	0.93	0.98	0.96	18464
Mirai	0.98	0.91	0.94	12850
Fuzzing	0.82	0.90	0.85	8656
Video Injection	0.87	0.63	0.73	2050
SSL Renegotiation	0.66	0.67	0.67	1853
OS Scan	0.50	0.64	0.56	1313
SYN Dos	0.85	0.23	0.37	141

Table 7: Class Performance Metrics of Centralised Learning (All Features)

Class	Precision	Recall	F1 Score	Support
Benign	0.81	0.85	0.83	50928
SSDP Flood	0.99	1.00	1.00	28792
ARP MitM	0.91	0.72	0.81	22905
Active Wiretap	0.93	0.98	0.95	18464
Mirai	0.88	0.98	0.92	12850
Fuzzing	0.82	0.90	0.85	8656
Video Injection	0.85	0.76	0.80	2050
SSL Renegotiation	0.66	0.67	0.67	1853
OS Scan	0.50	0.04	0.07	1313
SYN Dos	0.89	0.23	0.36	141

To see a detailed visualisation of these performance results, view the confusion matrices Figure 16 and 17 in Appendix D. The results in Table 6 and 17 show that centralised learning closely matches the performance of FedAvg and FedProx, achieving high precision and recall rates in most of the classes. It seems to perform better than both FedAvg and FedProx in classifying ARP MitM instances with higher recall, which means that a large portion of actual instances was correctly identified. Although centralised learning demonstrates marginally better performance for classifying SSL Renegotiation on the feature-selected dataset, the results are fairly similar across all of the strategies when both datasets are considered. The results also show that centralised learning yields a higher overall F1 score than FedAvg and FedProx in classifying Video Injection for both datasets due to its higher recall rates. However, it has noticeably worse performance than FedAvg and FedProx in classifying OS Scan, where the recall and F1 scores are extremely low. This indicates that it struggles more at classifying minority classes than FedAvg and FedProx.

Overall, the results show that centralised learning starts to decrease in performance for the minority classes in both types of datasets, which suggests that all of the strategies universally struggled with addressing the class imbalance. This means that the poor performance of classifying minority classes may not be due to the strategies themselves but the severity of the class imbalances in the dataset itself. However, the results also imply that FedAvg and FedProx were more robust than centralised learning at balancing the trade-off between false positives and false negatives for more of the classes. Additionally, FedAvg and FedProx performed relatively similarly and closely matched the performance of centralised learning.

7.5 Discussion

7.5.1 Performance of Federated Learning Strategies

FedAdagrad has more hyperparameters to tune than FedAvg or FedProx. The negative performance could be due to the improper tuning of these hyperparameters, such as the learning rate or initial accumulator value. Moreover, FedAdagrad does not contain explicit mechanisms to handle non-IID data in particular. The strategy's ability to adapt updates based on the frequency of features makes it more suited for use on data with feature sparsity. Therefore, its use may have been inappropriate in the first place, since the strategy lacked the ability to handle imbalanced classes within different clients. Given that FedAdagrad uses past gradient information to update its parameters, the model may have underperformed on the minority classes because most of the gradient information comes from the majority class. As a result, the bias may have skewed the learning process which led to its poorer performance.

FedProx adds a proximal term to the objective function to penalise local models that diverge drastically from the global model, providing robustness to the imbalanced data. In the absence of feature selection, FedProx's proximal term acts as a safeguard against the higher dimensionality and potential noise. This may have been the reason for its better performance over FedAvg when feature selection wasn't considered. In contrast, the feature selection of data reduces the dimensionality of the dataset, which may have caused the data distribution to be more homogeneous across the clients. As a result, the benefits of FedProx may have backfired by causing the model to underfit, due to unnecessarily constraining it on more homogeneously distributed data. Therefore, simpler weight averaging techniques such as FedAvg might have performed better due to their computational efficiency. Additionally, the data may have been easier to model as a result of the feature selection, which made FedAvg more effective since its assumptions of IID data hold more accurately the simpler data became.

7.5.2 Impact of Network Intrusion Types on Classification

The type of network intrusion can influence classification performance, due to factors, such as their inherent structure or patterns from features and characteristics, and the complexity of the network intrusions. Certain types of network intrusions may manifest as unique patterns of packet size, time intervals, protocol types, or source and destination IPs. For instance, SSDP Flood is a type of DDoS attack that uses a distinct pattern of network traffic flooding, which potentially contributed to its effective classification. This indicates that the abundance of certain classes may have been unnecessary for successful classification. However, benign network traffic being non-malicious, naturally has a higher representation, which resulted in its high classification performance.

SYN Dos is another type of DDoS attack where the attacker sends a succession of SYN requests to the target system. This attack is often identifiable due to recognisable patterns created by the SYN requests but all the models performed poorly in classifying it. Moreover, some attacks rely on being discreet to avoid detection such as active wiretapping or man-in-the-middle attacks, which causes them to be more challenging to classify. However, all of the models achieved high performance classifying Active Wiretap and ARP MitM classes. This implies that difficulties in classifying certain intrusions may not necessarily be due to their nature, but rather their underrepresentation in the training data.

Additionally, the minority classes may have been inherently more difficult to classify correctly due to their similarity to the features of classes with more representation. For example, fuzzing and SYN Dos both involve sending large numbers of packets over a short time period to overwhelm a target. This could have potentially confused the model as their similarities may have outweighed the subtle distinct patterns between the classes, especially since the imbalance is severe.

7.5.3 Research Limitations

Before concluding the overall implications of using federated learning, the results of the study must be evaluated while taking into account its limitations to enable a more complete interpretation. These include elements of federated learning that were not explored, which may provide explanations for suboptimal performances. There are several constraints in the research design that needs to be acknowledged:

Virtual Clients. In a realistic federated learning setting, hundreds to even millions of devices are typically used. However, this research was conducted with a maximum of eight clients. As a result, the model's ability to generalise may be limited, because the setup does not accurately scale to represent how devices in the real world would utilise the distributed data. With significantly fewer clients, each of their local data has a larger impact on the model. This meant that any noise or biases in the local data of a single device has the ability to significantly skew the model, potentially leading to biased or inaccurate results. Additionally, with the use of virtual clients, the impact of clients that are slow in contributing their model updates (stragglers) cannot be measured accurately. Similarly, the diversity and heterogeneity of real-world IoT devices, each with different computational capabilities aren't fully represented. Therefore, these factors may significantly bias the robustness and applicability of the study's findings in a real-world federated learning scenario.

Memory Constraints. Throughout the training phase, memory constraints were a predominant issue that not only posed a challenge but also dictated many of the design and implementation choices made. It had a significant impact on the effective implementation of complex strategies such as FedProx and FedAdagrad. Due to the computational requirements of the algorithms, the memory pressure led to the premature termination of worker processes. As a result, models may have performed poorly or unreliably due to incomplete or ineffective learning from the disruption of the training process. Moreover, the dataset has to be significantly subsetted to allow the federated learning to run without running out of memory and to reduce the slow training time. The complexity of the neural network model also had to be severely simplified to further optimise memory utilization. Consequently, these modifications may have resulted in poor performance, especially when classifying the minority classes. This performance decline could be due to the diminished opportunity for the model learning from fewer instances of the minority classes, along with potential underfitting caused by the oversimplified model architecture.

Training. Regularisation techniques such as L2 regularization were not used in the final implementation, which could have provided a certain degree of robustness against class imbalance. It could have potentially mitigated the model from being biased towards the majority class by preventing overfitting the majority class. Moreover, in practice, federated learning typically requires more communication rounds or epochs than traditional machine learning to reach optimal performance. Therefore, by maintaining the same number of epochs as centralised learning, the study may have inadvertently reduced the potential of federated learning to achieve better results.

8 Conclusion

8.1 Summary of Research Findings & Implications

The evaluation revealed centralised learning to have marginally superior overall accuracy over the federated learning strategies. However, federated learning has shown promising results with its overall performance, notably in classifying instances of minority classes, making it a viable alternative. At a class level, the performance of FedAvg and FedProx varied with respect to the type of dataset used. For the feature-selected dataset, FedAvg seemed to perform better on minority classes, while FedProx demonstrated better overall performance when all features were considered. Thus, the best strategy depends on the nature of the dataset and the importance of classifying minority classes accurately. In both dataset scenarios, all of the strategies universally underperformed in classifying minority classes, which suggests that the negative results are most likely due to the severely imbalanced dataset and not the strategies themselves.

While federated learning may not have achieved better overall accuracy, its value lies in its ability to identify minority network intrusions from distributed data. Therefore, this could be seen as a trade-off that is especially beneficial to network intrusion detection systems. Despite the encouraging results, it is imperative to remember the inherent challenges and limitations that came with implementing federated learning using virtual clients. Particularly the memory constraints that came with scaling on a single machine. Further investigation may be required due to the many changes in initial design and implementation to address these challenges. Overall, the evaluation offers insights into the strengths and limitations of the different federated learning strategies which may hopefully serve as a beneficial reference for future research and improvements.

8.2 Proposed Directions for Future Research

While this study has shed light on the inherent benefits of federated learning over traditional machine learning, it also opens the door to numerous considerations for future advancement and deeper understanding in this research. The use of data generated in real time from actual IoT devices to train NIDS is a potential approach to create more realistic future scenarios. This could potentially pave the way for the anticipation and mitigation of new attacks or anomalies. Moreover, research can be explored with various network settings, topologies, and traffic conditions to optimise bandwidth utilisation and reduce communication overhead inherent in federated learning. An effective approach could be through configuring client-side executions from the server to optimise model updates prior to their dispatch or guide clients to better select what data segments to use.

Additionally, more optimisation techniques could be explored to improve the model performance of federated learning strategies. This could include the use of more diverse sets of hyperparameters, advanced aggregation techniques, or other novel strategies to address heterogeneous data. Given the challenges faced in this research, efficient solutions for memory constraints could be explored through the effective implementation of data or model compression. Conversely, a larger scale of physical client devices could be used instead to more accurately reflect the heterogeneity of real scenarios. Lastly, an important aspect that was not explored in this thesis is the effectiveness of federated learning in preserving privacy. The evaluation of privacy preservation techniques could be another crucial area for future investigation.

References

[1] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019.

[5] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[6] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017.

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[8] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2020.

[9] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning," 2020.

[10] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3557–3568, 2020.

[11] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[12] Y. Mirsky, "*Kitsune Network Attack Dataset*," 2022, https://www.kaggle.com/datasets/ymirsky/network-attack-dataset-kitsune.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] F. F. Learning, "*What is Federated Learning?*" 2022, https://flower.dev/docs/tutorial/Flower-0-What-is-FL.html.

[15] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2022.

[16] S. Ghoneim, "*Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?*" 2019, https://towardsdatascience.com/accuracy-recall-precision-f-scorespecificity-which-to-optimize-on-867d3f11124.

[17] Google, "*Classification: Precision and Recall*," 2022, https://developers.google.com/machine-learning/crash-course/classification/precision-andrecall#:~:text=Unfortunately%2C%20precision%20and%20recall%20are,by%20an%20email%20classification%20model.

A Training Data Class Distribution Imbalance

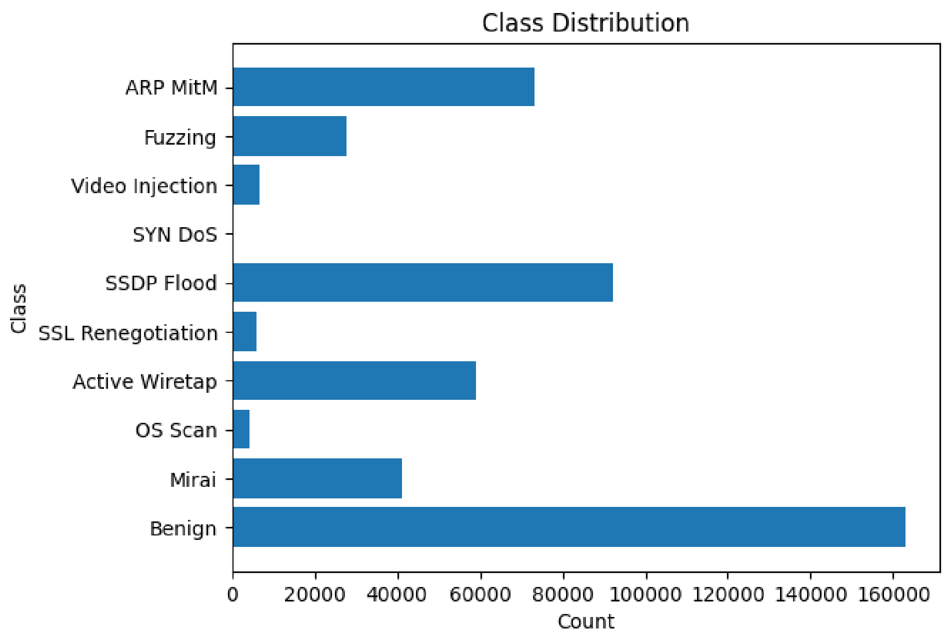
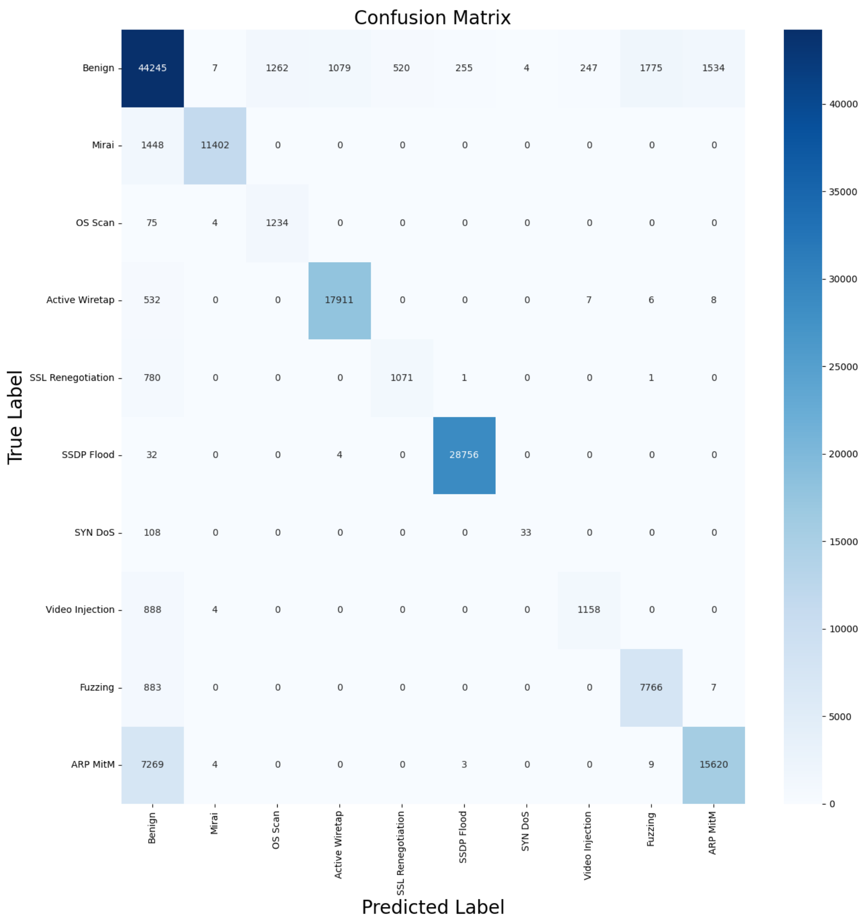


Figure 11: Imbalanced Class Distribution of the Training Data Used

B Confusion Matrices of Performance on the Feature-Selected Dataset



True Label \ Predicted Label	Benign	Mirai	OS Scan	Active Wiretap	SSL Renegotiation	SSDP Flood	SYN DoS	Video Injection	Fuzzing	ARP MitM
Benign	44245	7	1262	1079	520	255	4	247	1775	1534
Mirai	1448	11402	0	0	0	0	0	0	0	0
OS Scan	75	4	1234	0	0	0	0	0	0	0
Active Wiretap	532	0	0	17911	0	0	0	7	6	8
SSL Renegotiation	780	0	0	0	1071	1	0	0	1	0
SSDP Flood	32	0	0	4	0	28756	0	0	0	0
SYN DoS	108	0	0	0	0	0	33	0	0	0
Video Injection	888	4	0	0	0	0	0	1158	0	0
Fuzzing	883	0	0	0	0	0	0	0	7766	7
ARP MitM	7269	4	0	0	0	3	0	0	9	15620

Figure 12: Confusion Matrix of FedAvg on the Feature-selected Dataset

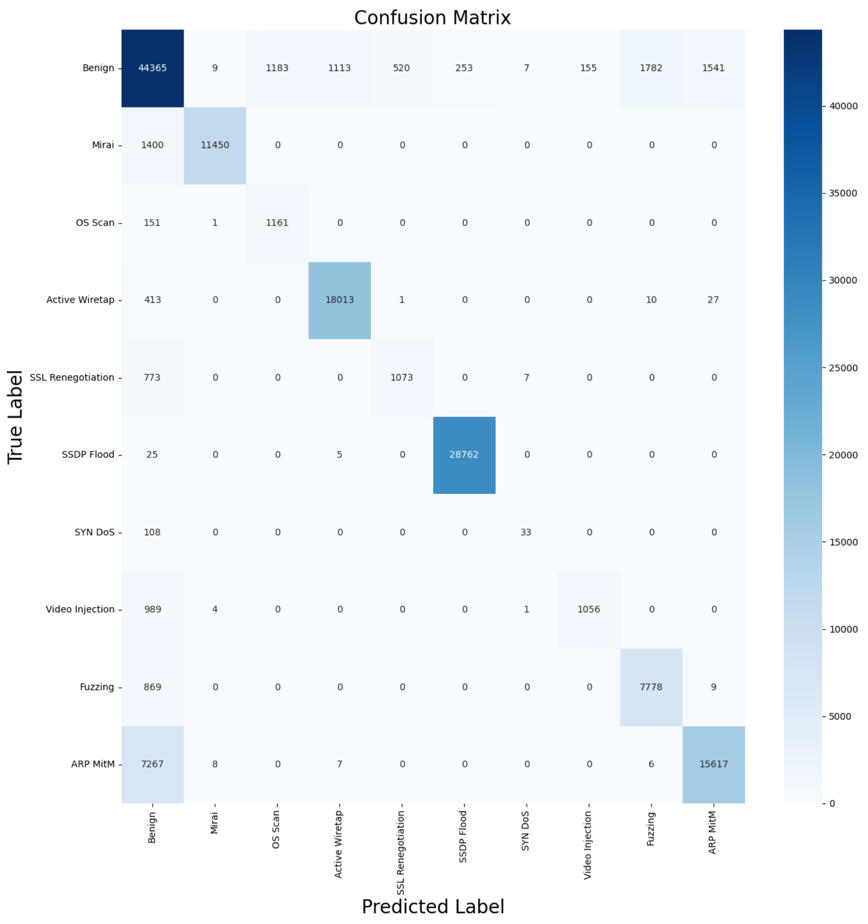
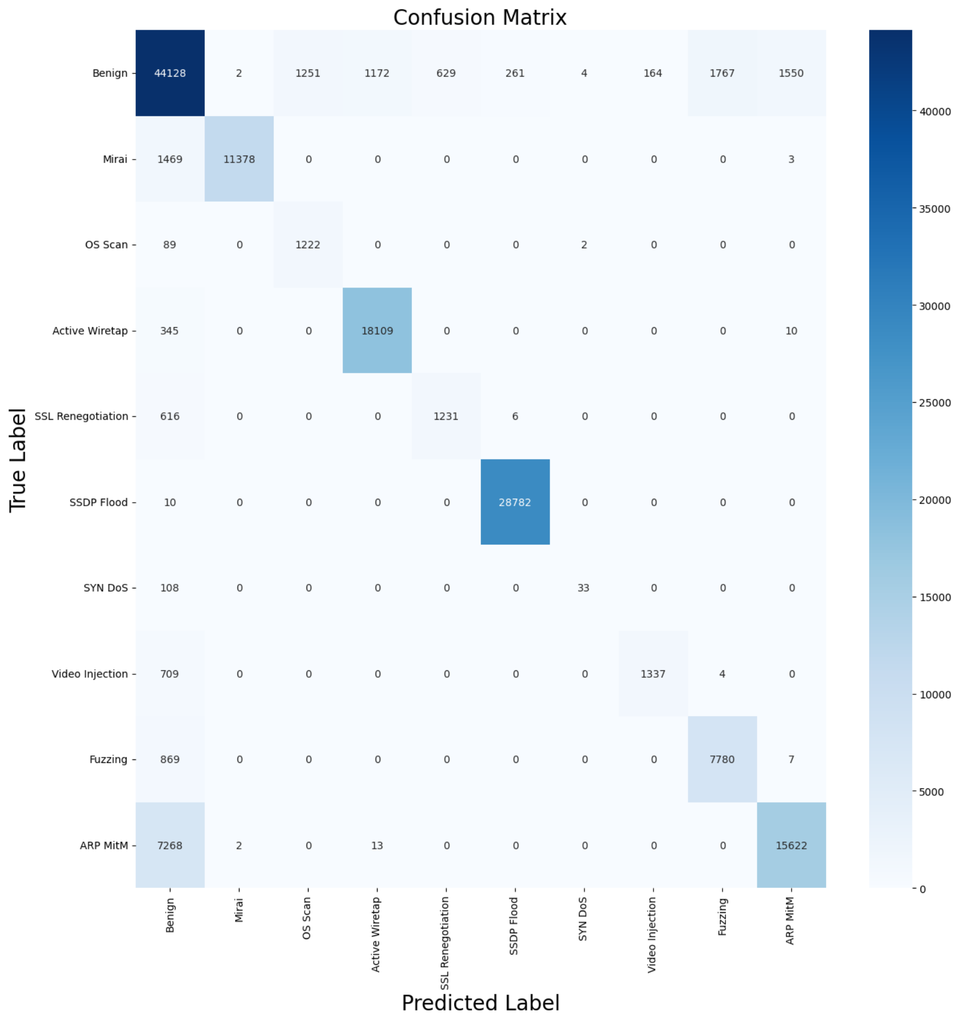


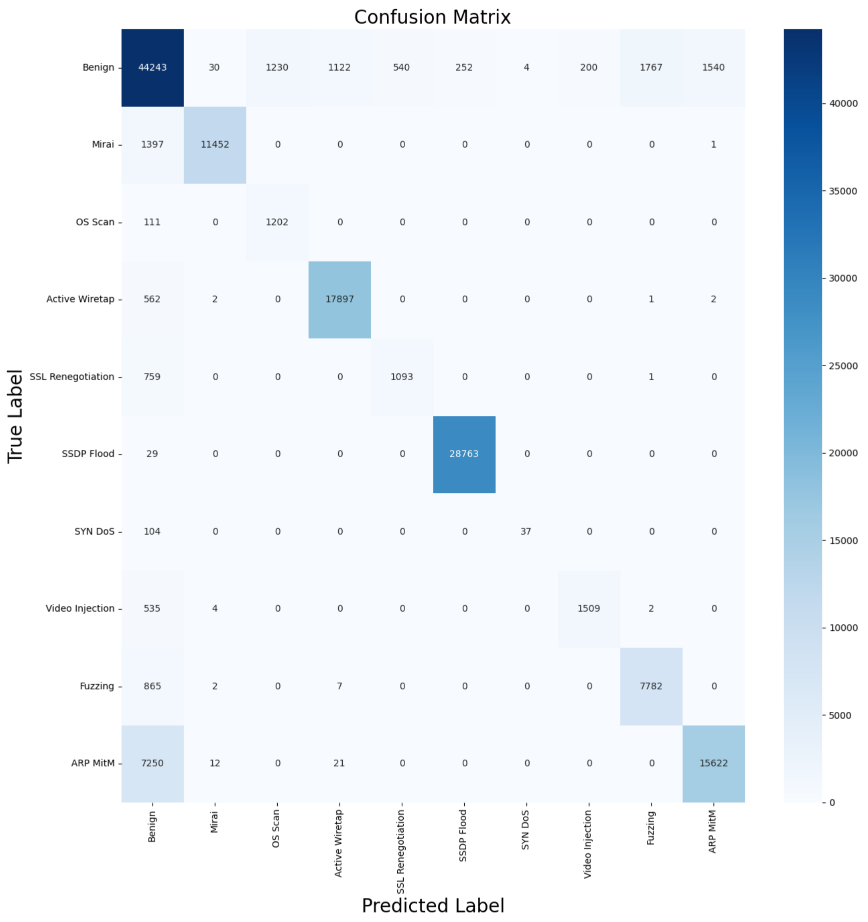
Figure 13: Confusion Matrix of FedProx on the Feature-selected Dataset

C Confusion Matrices of Performance on the Non Feature-Selected Dataset



	Benign	Mirai	OS Scan	Active Wiretap	SSL Renegotiation	SSDP Flood	SYN DoS	Video Injection	Fuzzing	ARP MitM
Benign	44128	2	1251	1172	629	261	4	164	1767	1550
Mirai	1469	11378	0	0	0	0	0	0	0	3
OS Scan	89	0	1222	0	0	0	2	0	0	0
Active Wiretap	345	0	0	18109	0	0	0	0	0	10
SSL Renegotiation	616	0	0	0	1231	6	0	0	0	0
SSDP Flood	10	0	0	0	0	28782	0	0	0	0
SYN DoS	108	0	0	0	0	0	33	0	0	0
Video Injection	709	0	0	0	0	0	0	1337	4	0
Fuzzing	869	0	0	0	0	0	0	0	7780	7
ARP MitM	7268	2	0	13	0	0	0	0	0	15622

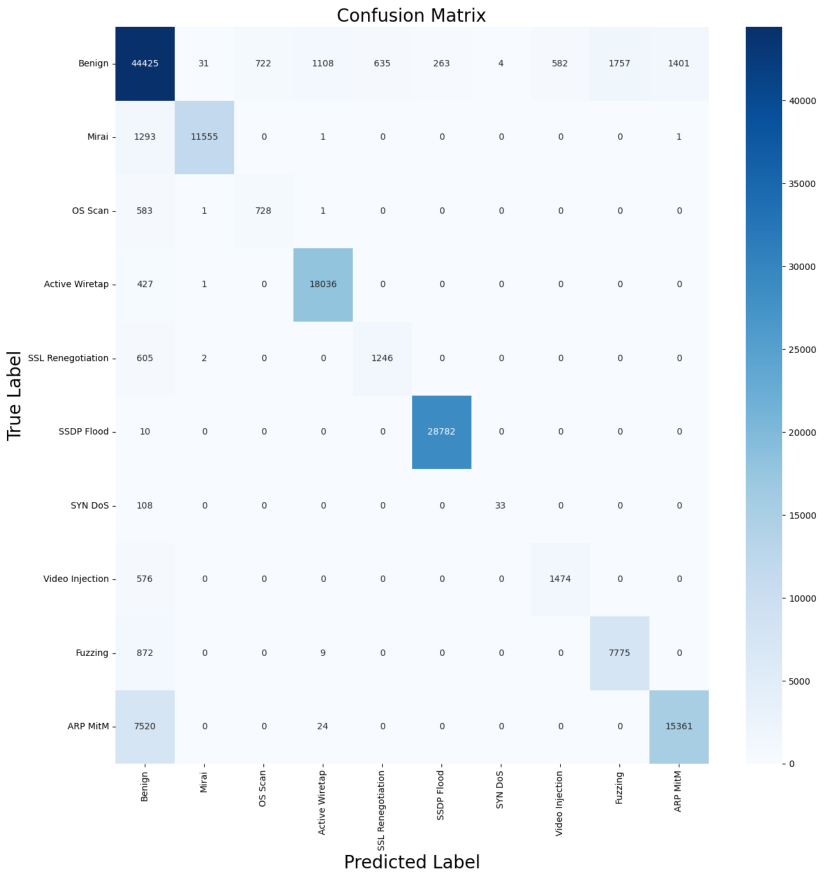
Figure 14: Confusion Matrix of FedAvg on the Non Feature-selected Dataset



True Label \ Predicted Label	Benign	Mirai	OS Scan	Active Wiretap	SSL Renegotiation	SSDP Flood	SYN DoS	Video Injection	Fuzzing	ARP MitM
Benign	44243	30	1230	1122	540	252	4	200	1767	1540
Mirai	1397	11452	0	0	0	0	0	0	0	1
OS Scan	111	0	1202	0	0	0	0	0	0	0
Active Wiretap	562	2	0	17897	0	0	0	0	1	2
SSL Renegotiation	759	0	0	0	1093	0	0	0	1	0
SSDP Flood	29	0	0	0	0	28763	0	0	0	0
SYN DoS	104	0	0	0	0	0	37	0	0	0
Video Injection	535	4	0	0	0	0	0	1509	2	0
Fuzzing	865	2	0	7	0	0	0	0	7782	0
ARP MitM	7250	12	0	21	0	0	0	0	0	15622

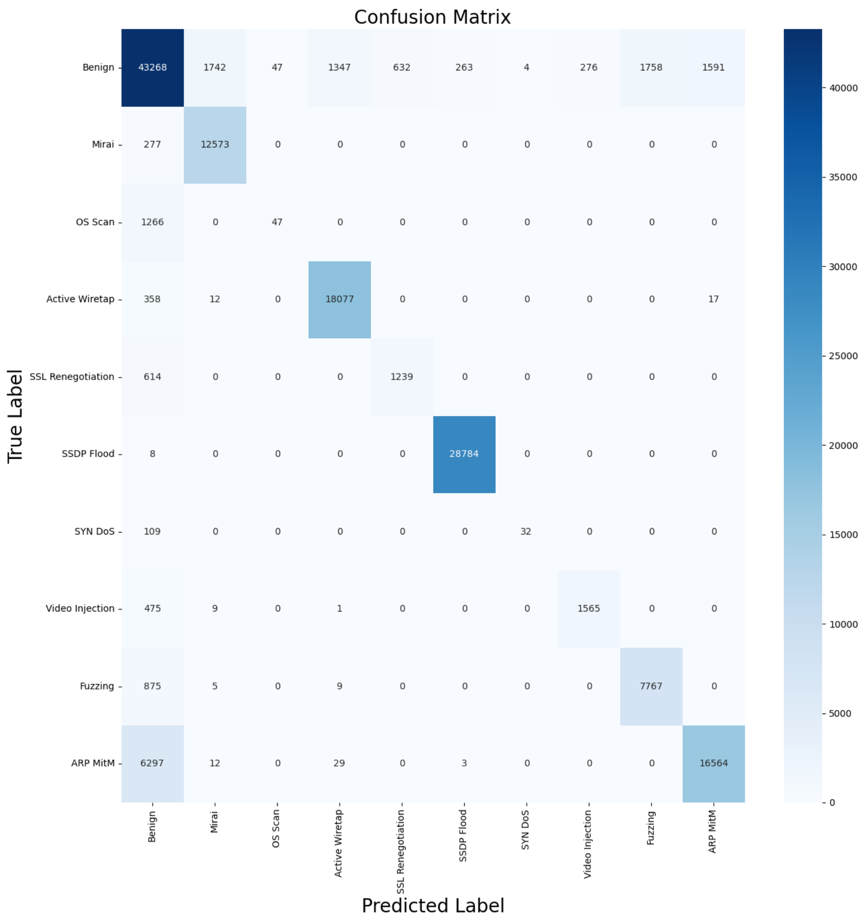
Figure 15: Confusion Matrix of FedProx on the Non Feature-selected Dataset

D Confusion Matrices of Centralised Learning Performance



True Label \ Predicted Label	Benign	Mirai	OS Scan	Active Wiretap	SSL Renegotiation	SSDP Flood	SYN DoS	Video Injection	Fuzzing	ARP MitM
Benign	44425	31	722	1108	635	263	4	582	1757	1401
Mirai	1293	11555	0	1	0	0	0	0	0	1
OS Scan	583	1	728	1	0	0	0	0	0	0
Active Wiretap	427	1	0	18036	0	0	0	0	0	0
SSL Renegotiation	605	2	0	0	1246	0	0	0	0	0
SSDP Flood	10	0	0	0	0	28782	0	0	0	0
SYN DoS	108	0	0	0	0	0	33	0	0	0
Video Injection	576	0	0	0	0	0	0	1474	0	0
Fuzzing	872	0	0	9	0	0	0	0	7775	0
ARP MitM	7520	0	0	24	0	0	0	0	0	15361

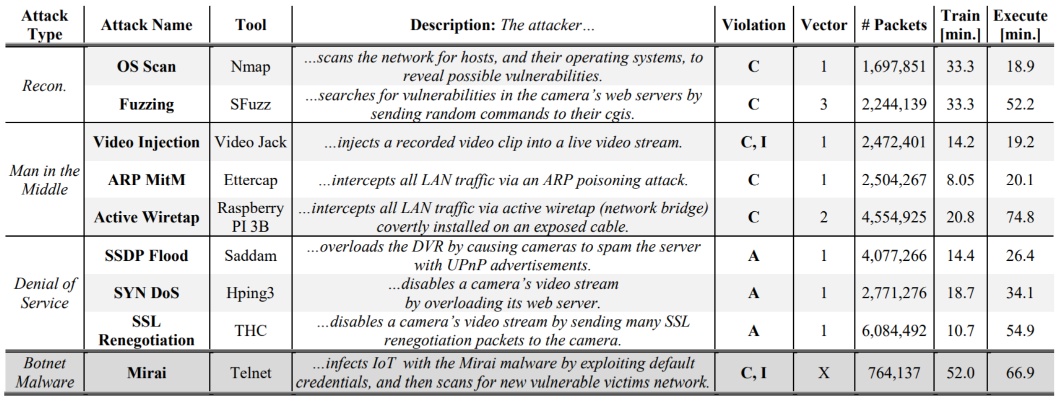
Figure 16: Confusion Matrix of Centralised Learning on the Feature-selected Dataset



	Benign	Mirai	OS Scan	Active Wiretap	SSL Renegotiation	SSDP Flood	SYN DoS	Video Injection	Fuzzing	ARP MitM
Benign	43268	1742	47	1347	632	263	4	276	1758	1591
Mirai	277	12573	0	0	0	0	0	0	0	0
OS Scan	1266	0	47	0	0	0	0	0	0	0
Active Wiretap	358	12	0	18077	0	0	0	0	0	17
SSL Renegotiation	614	0	0	0	1239	0	0	0	0	0
SSDP Flood	8	0	0	0	0	28784	0	0	0	0
SYN DoS	109	0	0	0	0	0	32	0	0	0
Video Injection	475	9	0	1	0	0	0	1565	0	0
Fuzzing	875	5	0	9	0	0	0	0	7767	0
ARP MitM	6297	12	0	29	0	3	0	0	0	16564

Figure 17: Confusion Matrix of Centralised Learning on the Non Feature-selected Dataset

E List of Cyber Attack Datasets Used



Attack Type	Attack Name	Tool	Description: *The attacker…*	Violation	Vector	# Packets	Train [min.]	Execute [min.]
Recon.	**OS Scan**	Nmap	*…scans the network for hosts, and their operating systems, to reveal possible vulnerabilities.*	**C**	1	1,697,851	33.3	18.9
	Fuzzing	SFuzz	*…searches for vulnerabilities in the camera's web servers by sending random commands to their cgis.*	**C**	3	2,244,139	33.3	52.2
Man in the Middle	**Video Injection**	Video Jack	*…injects a recorded video clip into a live video stream.*	**C, I**	1	2,472,401	14.2	19.2
	ARP MitM	Ettercap	*…intercepts all LAN traffic via an ARP poisoning attack.*	**C**	1	2,504,267	8.05	20.1
	Active Wiretap	Raspberry PI 3B	*…intercepts all LAN traffic via active wiretap (network bridge) covertly installed on an exposed cable.*	**C**	2	4,554,925	20.8	74.8
Denial of Service	**SSDP Flood**	Saddam	*…overloads the DVR by causing cameras to spam the server with UPnP advertisements.*	**A**	1	4,077,266	14.4	26.4
	SYN DoS	Hping3	*…disables a camera's video stream by overloading its web server.*	**A**	1	2,771,276	18.7	34.1
	SSL Renegotiation	THC	*…disables a camera's video stream by sending many SSL renegotiation packets to the camera.*	**A**	1	6,084,492	10.7	54.9
Botnet Malware	**Mirai**	Telnet	*…infects IoT with the Mirai malware by exploiting default credentials, and then scans for new vulnerable victims network.*	**C, I**	X	764,137	52.0	66.9

Figure 18: List of Cyber Attack Datasets Used