B.Tech Major Project Report

CSPE - 40

On

INTRUSION DETECTION SYSTEM USING FEDERATED MACHINE LEARNING

By

KESHAV BHALLA (11812046)

PUNIT JAIN (11812048)

KIRAN KHARB (11812094)

Group No. 3

Under the supervision of

Dr. Mayank Dave, Professor

DEPARTMENT OF COMPUTER ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

KURUKSHETRA – 136119, HARYANA (INDIA)

March, 2022

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech Project (CSPE-40) report entitled "**Intrusion Detection Using Federated Machine Learning**", in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** is an authentic record of our own work carried out during a period from January, 2022 to May, 2022 under the supervision of Dr. Mayank Dave, Professor, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidate

KESHAV BHALLA (11812046)

PUNIT JAIN (11812048)

KIRAN KHARB (11812094)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signature of Supervisor

Dr. Mayank Dave

Professor

TABLE OF CONTENTS

Section No.	TITLE	Page no.
	ABSTRACT	5
1	INTRODUCTION	6
2	MOTIVATION	7
3	LITERATURE SURVEY	8
	3.1 Existing Approaches	8
	3.2 Related works	8
4	FEDERATED MACHINE LEARNING SIMULATION	9-10
5	FEATURE SELECTION	11-13
6	NETWORK ATTACKS	14-15
7	EXPERIMENTAL SETUP	16-17
8	DATA FLOW DIAGRAM	18-19
	8.1 Level 0 DFD	18
	8.2 Level 1 DFD	18
	8.3 Level 2 DFD	19
9	RESULTS/ OBSERVATIONS	20-22
10	CONCLUSION AND FUTURE PLAN	23

	REFERENCES (in IEEE format)	24
APPENDIX:		
A	COMPLETE **CONTRIBUTARY** SOURCE CODE	25-36

ABSTRACT

An Intrusion Detection System is a software which monitors network packet related traffic and detects aberrations. Unexpected/Un-authorised network requests might signal illegal activity at any level, from the start of an assault to a full-fledged breach.

The traditional intrusion detection systems are based upon centralised learning, where all of the data needs to be present at a central server. But storing the user's data at a central server is prone to risk of data leakage and increases the concerns of user data privacy. This problem can be tackled by using Federated Machine Learning approach. This enables users to build a consensus model collaboratively without moving data beyond the device in which they reside.

1. INTRODUCTION

The fast growth of the Internet and smart devices has resulted in an increase in network traffic, making the infrastructure increasingly complicated and varied. Mobile phones, wearable gadgets, and self-driving cars are all instances of dispersed networks that create massive amounts of data on a daily basis. Intrusion detection systems are critical for guaranteeing the security and privacy of these devices. Due to their excellent classification accuracy, Machine Learning and Deep Learning with Intrusion Detection Systems have acquired a lot of traction. However, because of the requirement to store and communicate data to a centralised server, the privacy and security elements may be jeopardised.

Federated Learning (FL), on the other hand, is a privacy-preserving decentralised learning approach that does not transport data but instead trains models locally before sending the parameters to a centralised server. This data is then aggregated on the central server using certain algorithms to update the global model. This process continues up-till a certain number of rounds till we reach the desired accuracy.

While training the model, there are a large number of input parameters/features and some of them are nothing but noise. Therefore, it is essential for us to choose certain relevant features on which the model will be trained. This not only improves the accuracy of the model, but also helps to save a lot of training time. This can be done using certain feature selection algorithms.

The goal of our project is to give an in-depth analysis of the application of Federated Machine Learning in Intrusion Detection Systems using feature selection.

2. MOTIVATION

There is rising worry regarding the security of network-connected embedded devices these days, as they may be vulnerable to a huge variety of vulnerabilities that an attacker may exploit. Intrusion Detection System (IDS) is a standard tool for detecting aberrant network behaviour.

Various machine learning and deep learning algorithms have been used to identify intrusions in recent years. They either need a lot of computational resources to train a prediction model or they outsource the training to another machine, such as a server. On edge devices, data transfer to central server units may cause further delay, increased cost of communication, and possible hazards.

We employed Federated Machine Learning to solve this problem. Federated learning is a machine learning technique that allows machine learning models to gain experience from many data sets located at separate locations (e.g., local data centres, a central server) without exchanging training data. This allows personal data to stay on local servers, lowering the risk of data breaches. We employed feature selection techniques to weed out irrelevant and noisy input data pieces to boost efficiency even more.

3. LITERATURE SURVEY

3.1 Existing Approaches

Previously, machine learning algorithms were used to identify intrusions. IDS implements the model's training process by learning from both normal and anomalous traffic using machine learning techniques [1]. It is used to find patterns relevant to malicious traffic in a dataset or in the environment. Federated machine learning was offered as a way to overcome these hurdles. FL allows devices to learn collaboratively without having to share data with a centralised server [4]. To put it another way, machine learning models may be trained on numerous devices and servers using decentralised data across multiple iterations. In a federated architecture, data is only accessible through client edge devices. Unlike global datasets, federated data cannot be tracked or identified, and its position and availability are constantly changing as new data is generated in an edge device.

3.2 Related Works

The author of paper [1] gives a detailed assessment of Federated learning-based Intrusion Detection System systems that prioritise security, privacy, and dependability. The authors describe how Federated Machine Learning can assist a centralised Machine Learning strategy in providing an effective intrusion detection solution, as well as the advantages of employing Federated Machine Learning over machine learning.

Paper [3], proposed a federated deep-learning scheme, for detecting and reducing cyber threats against industrial cyber–physical systems. They have created a federated learning framework that allows numerous industrial CPSs to work together to create a complete intrusion detection model while maintaining privacy.

The publishers of article [2] consider an anomaly-based intrusion detection system that leverages network traffic statistics to separate assaults from legitimate requests. The authors have also utilised a greedy feature selection technique to increase the model's accuracy. However, because the feature selection approach is greedy, the dataset utilised in this research is NSL-KDD, which only covers four types of attacks.

The authors of study [4] demonstrated that federated machine learning may be derived from a single model for numerous devices while local data is safeguarded, allowing devices to collaborate on data security.

4. FEDERATED MACHINE LEARNING SIMULATION

Federated learning is a machine learning strategy that allows machine learning models to gain experience from different data sets located in separate locations (for example, a local data centre) without having to share training data.

Their major goal is to create machine learning models using data sets dispersed across several devices while avoiding data loss.

The following are the many steps involved in federated machine learning:

1. On local heterogeneous datasets, local machine learning (ML) models are trained. Users of a machine learning application, for example, can identify and rectify errors in the application's predictions as they use it. Local training datasets are created in each user's device as a result of this.

2. The model parameters are shared between these local data centres on a regular basis. Many models encrypt these parameters before sending them. Data samples from the local area are not shared. This increases data security and protection.

3. A shared global model is built.

4. The global model's attributes are provided with local data centres so that they can include the global model into their ML local models.

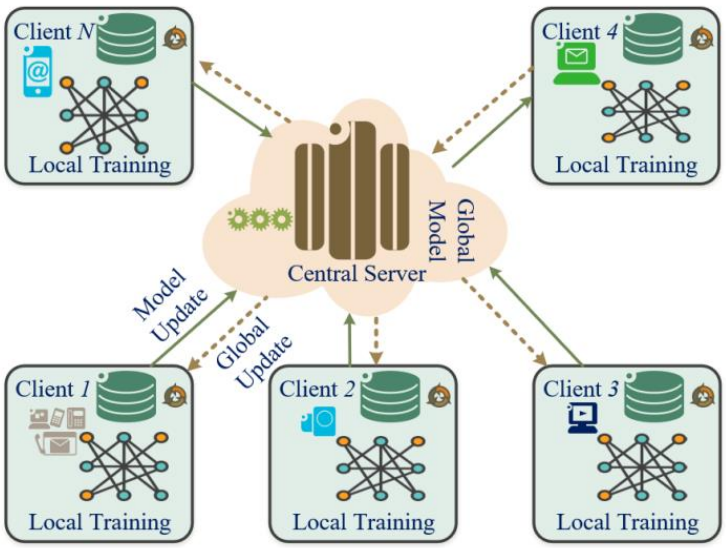

Fig 4.1 – Federated Machine Learning Architecture

The main concepts in our work include:

- **On-device neural network**: To execute network intrusion detection, we use the most up-to-date on-device sequential learning neural network. Embedded devices handle both training and inference calculations.
- **Federated learning-based intrusion detection system**: To create detection models with model aggregation, we use a horizontal federated learning architecture. The aggregation process is dependent on the feature set that each user selects.

Advantages

The advantages of using Federated Machine Learning are:

- Data Security: Keeping the training dataset on the devices eliminates the need for a data pool for the model.
- Real Time Learning: There is no need to aggregate data for continuous learning because models are continually upgraded using client data.
- Hardware Efficient: Because federated learning models do not require a single complex central server to interpret data, this technique requires less complex hardware.

5. FEATURE SELECTION

In machine learning, features are independent variables that act as an input to the system, models use such features to make predictions. Each column in our dataset constitute a feature. In order to optimally train a machine learning model, we need to use only relevant features. With too many features model can capture unimportant patterns, learn from noise and give undesirable results. Feature selection in machine learning is the technique to identify and select the most appropriate from the data and remove the irrelevant, redundant or less important features**.**

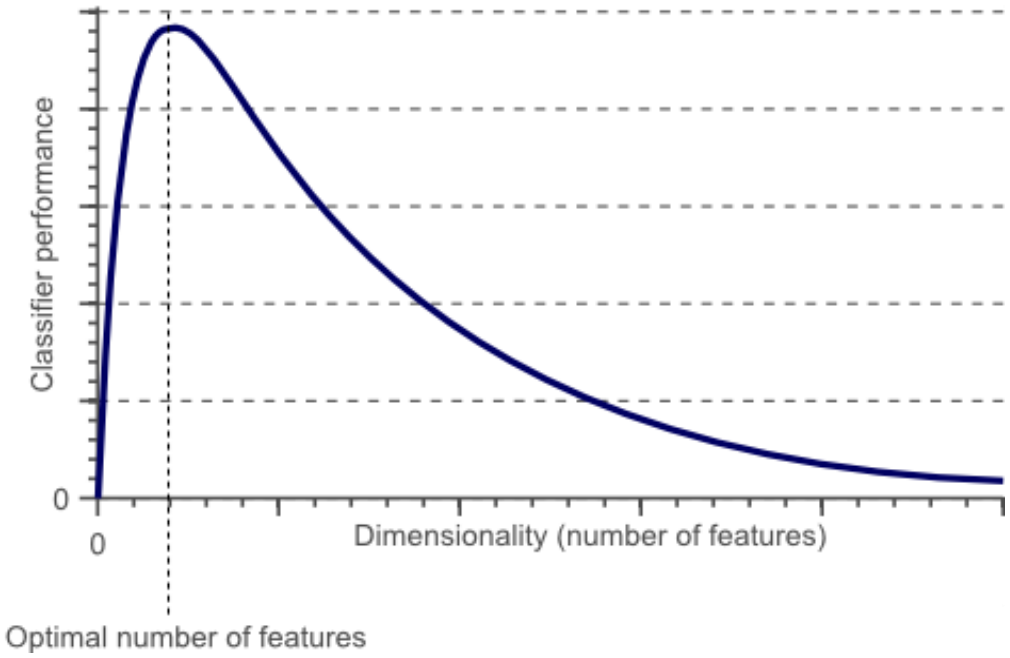

Fig 5.1 – Dimensionality/Performance Curve

Machine learning models are trained on a large quantity of data, which enables them to learn more effectively. In most cases, a significant amount of the data contains noise that is unrelated to the output desired. Furthermore, a large volume of data affects the model's efficiency, and because of the noise and irrelevant data, the model may be unable to forecast and perform effectively, decreasing its accuracy. As a result, it is critical to eliminate such noisy and unnecessary data from the dataset, which is accomplished using Feature Selection Techniques.

There are 2 types of Feature Selection Models - Supervised and Unsupervised. The Supervised Feature selection Models are further classified into three types:

- Filter Method: In this method, features are dropped on the basis of their correlation to the output. If the features are negatively correlated to the output class, we drop them.

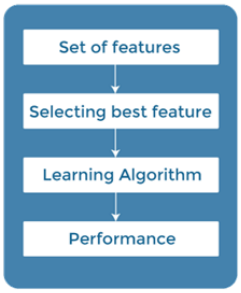

Fig 5.2 – Filter Method

- Wrapper Method: In Wrapper method, model is trained on the subsets of the dataset iteratively. Based on the correlation with the output class, we add and subtract features and the model is again trained on this dataset.

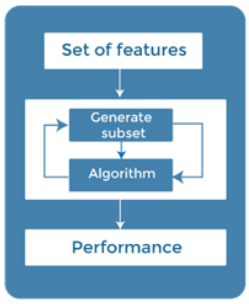

Fig 5.3 –Wrapper Method

- Embedded Methods: Embedded techniques, as the name implies, incorporate the benefits of both Filter and Wrapper methods by taking into account the interplay of features while maintaining a low computing cost. These approaches are likewise iterative, in which each iteration is evaluated and the most significant characteristics that contribute the most to training in that iteration are optimally found.

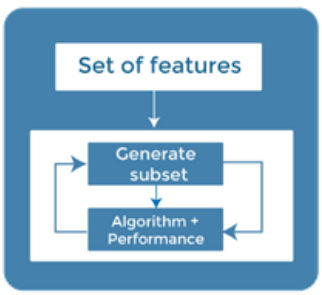

Fig 5.3 – Embedded Method

The basic algorithm of the feature selection algorithm that we are using is given below:

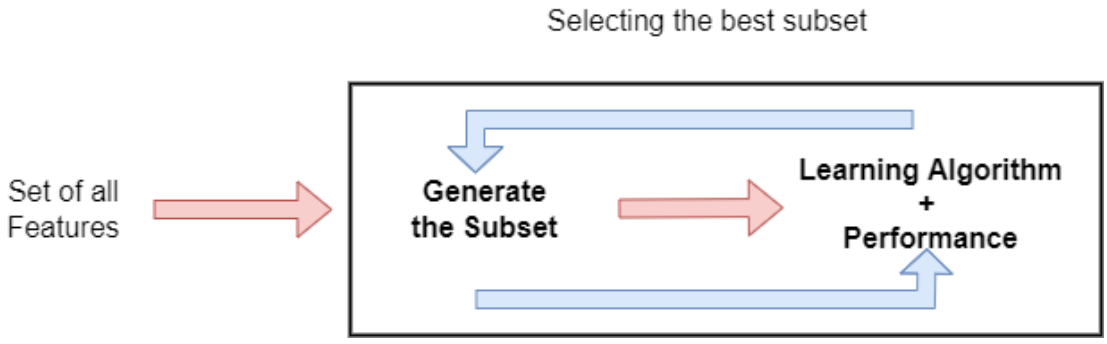

Fig 5.4 – Feature Selection

Feature selection has various advantages:

- It reduces the training time. Since the model gets trained on a more precise subset of features, the amount of time needed to train a model gets shortened.
- It aids in avoiding the dimensionality curse. The dimensionally cursed phenomenon argues that as dimensionality and the number of features rise, the volume of space expands at such a rapid rate that the amount of data available decreases. To get around this, we can utilise PCA feature selection to minimise dimensionality.
- Feature selection also reduces the problem of overfitting.

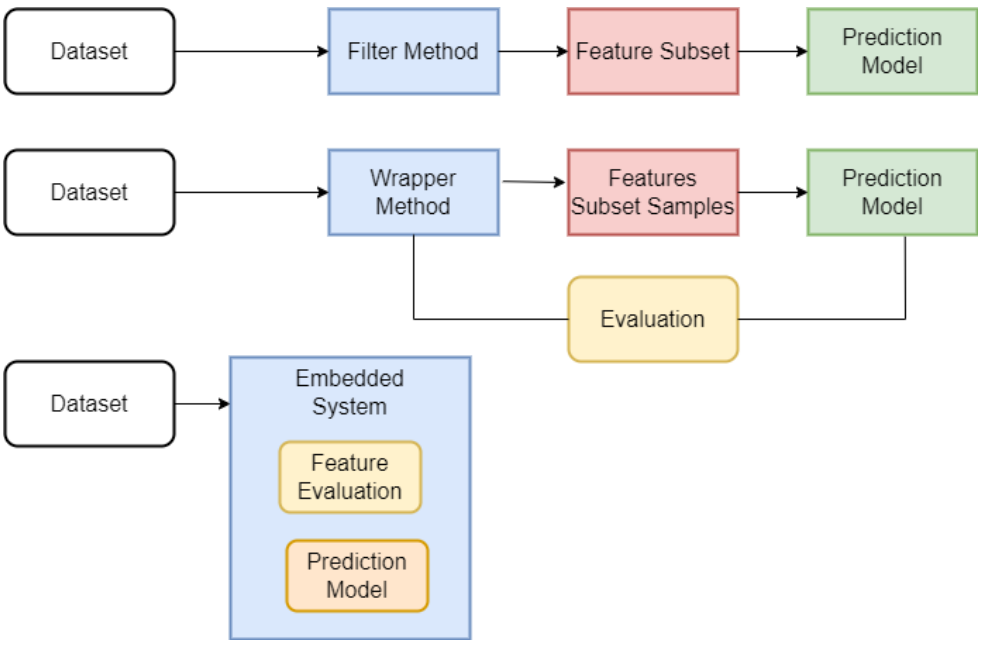

Fig 5.5 – Feature Selection Methods

6. NETWORK ATTACKS

CIC-IDS dataset covers seven types of attacks the details of which are discussed below:

Brute-Force Attack: This is a hacking attack that cracks passwords, login credentials, and encryption keys by trial and error. Hackers use brute-force attacks to obtain unauthorised access to a company's network and systems, as well as individual accounts. The hackers have a large number of potential password combinations that they attempt repeatedly until they locate the proper information. This is an ancient cyberattack approach that continues to be used by hackers.

HeartBleed Attack: HeartBleed attack is exploited using tool HeartLeech. In this attack the tool scans the systems that are vulnerable to a specific bug and then use that bug to infiltrate systems and exfiltrate data . Here are some important features of HeartBleed:

- It gives conclusive answers as to whether or not the target is susceptible.
- It allows for quicker HeartBleed data download into big files for offline processing utilising many threads.
- This approach allows for the retrieval of private keys to be done automatically.
- It provides STARTTLS support.
- It also has IPv6 support.

Botnet: In CICIDS dataset, Zeus has been used . Zeus is a Trojan horse based virus/malware package that runs on various versions of Windows. This attack is often used to rob banking information keystroke logging and form grabbing. The systems are infiltrated with Zeus mainly by drive-by downloads and phishing schemes. In another scenario of Botnet attack, the system is infiltrated with two types of bots , namely , Zeus & Ares and the attacker request screenshots every 400 seconds.

Denial-of-Service (DoS): A denial-of-service attack involves flooding a system or network with unwanted traffic or delivering it information that will cause it to crash. A Sloworis Perl-based utility is used in some instances to take down web servers. It accomplishes this by letting a single system to bring down another machine's web services with minimum bandwidth and other harmful effects on other services and ports.

Distributed Denial-of-Service (DDoS): DDoS (distributed denial of service) attacks are sub-type of DoS (distributed denial of service) attack. DDoS attacks employ numerous linked internet devices to fill a target website with bogus traffic (botnet). These assaults are not aimed at breaching the target's security parameters, but rather at making the website or servers unavailable to legitimate users. If successful, this attack will have a large influence on the online community, making it a highly visible event.

Web Attacks: This assault is directed towards a web application. This attack may be carried out by injecting data into a web application in order to alter it and retrieve the information needed. DNS spoofing, in which data is injected into a DNS resolver's cache, leading the name server to provide an inaccurate IP address, routing traffic to the attacker's computer or any other computer, is another method.

Infiltration of the network from inside: The purpose of this attack is to find a vulnerable application. The target is sent a malicious document, and following successful exploitation, a backdoor is installed on the victim's machine. Finally, various assaults on the victim's network are carried out, including IP sweeps, service enumerations, and complete port scans.

7. EXPERIMENTAL SETUP

1. **Dataset**: The Canadian Institute of Cybersecurity provided CICIDS2017, a state-of-the-art dataset containing the most recent threats and characteristics, to evaluate the efficacy of the IDS. This dataset contains risks that were not covered by previous datasets. Heartbleed, Brute-force, Botnet, DoS, DDoS, Web assaults, and network penetration from within are among the seven attack scenarios included in the dataset.

Sl No	*New Labels*	*Old Labels*	*Number of instances*	*% of prevalence w.r.t. the majority class*	*% of prevalence w.r.t. the total instances*
1	Normal	Benign	2359087	100	83.34
2	Botnet ARES	Bot	1966	0.083	0.06
3	Brute Force	FTP-Patator, SSH-Patator	13835	0.59	0.48
4	Dos/DDos	DDoS, DoS GoldenEye, DoS Hulk, DoS Slow-httptest, DoS slowloris, Heartbleed	294506	12.49	10.4
5	Infiltration	Infiltration	36	0.001	0.001
6	PortScan	PortScan	158930	6.74	5.61
7	Web Attack	Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS	2180	0.092	0.07

Fig 7.1 Attacks in CIC-IDS2017

2. **Keras**: Keras runs on top of open-source machine libraries like TensorFlow and provides an easy way to create deep learning models based on TensorFlow. The activation function applied to the hidden layer and output layer is tanh and softmax, respectively.

   First layer in sequential model should receive information about its input shape as the following layers can do automatic shape inference. One of the ways to do this is to pass *input_shape* argument to the first layer. Refer to the figure given below:

```
model=keras.models.Sequential([
        keras.layers.Flatten(input_shape=[77,]),
        keras.layers.Dense(200,activation='tanh'),
        keras.layers.Dense(100,activation='tanh'),
        keras.layers.Dense(12,activation='softmax')
    ])
```

Fig 7.2 – Keras Model

Keras models are trained on Numpy arrays of input data and labels. For training a model, we used the fit function.

```
history=model.fit(self.dataset_x, self.dataset_y, epochs=self.epoch_number,batch_size=self.batch)
```

Fig 7.3 – Model Training

8. Data Flow Diagram

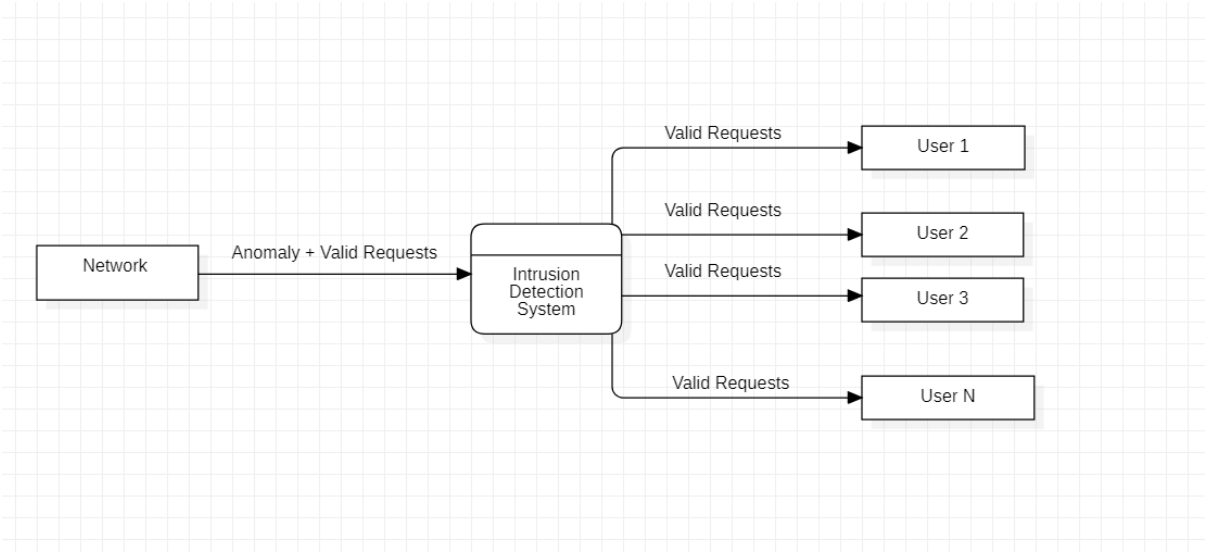

Fig 8.1 -Level 0-DFD

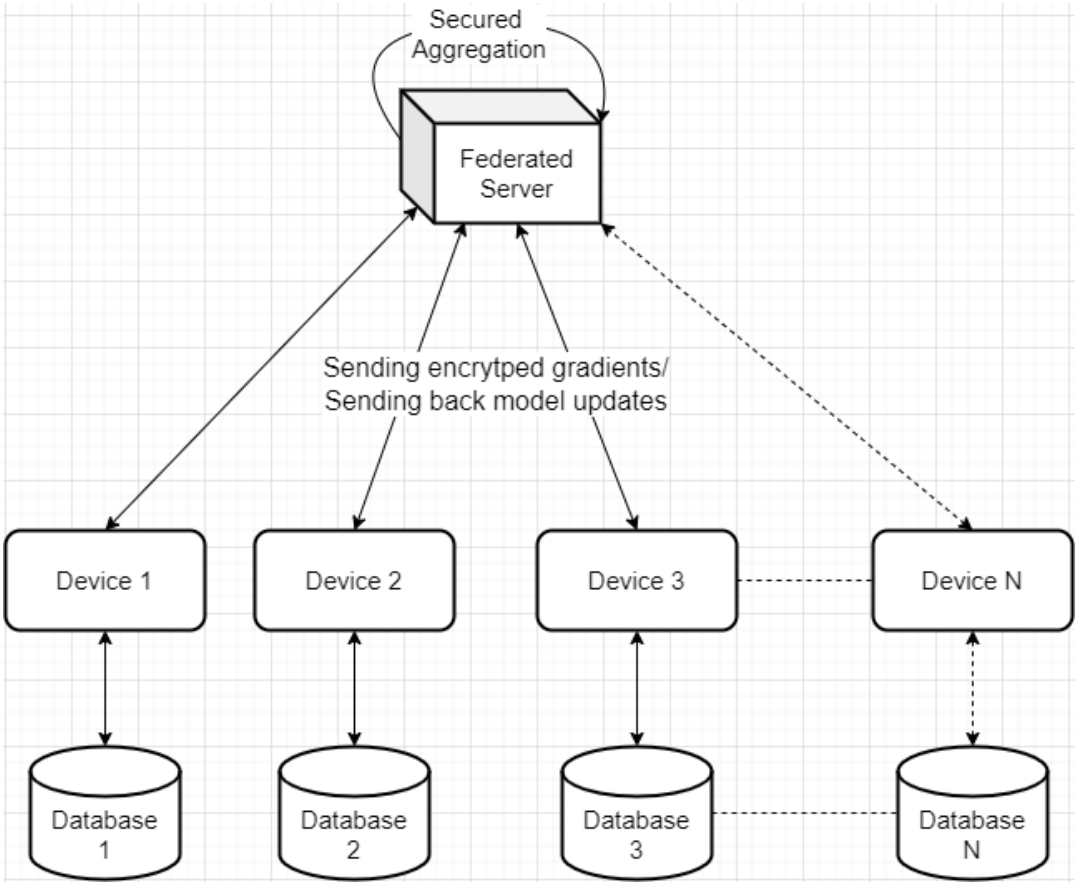

Fig 8.2- Level 1-DFD

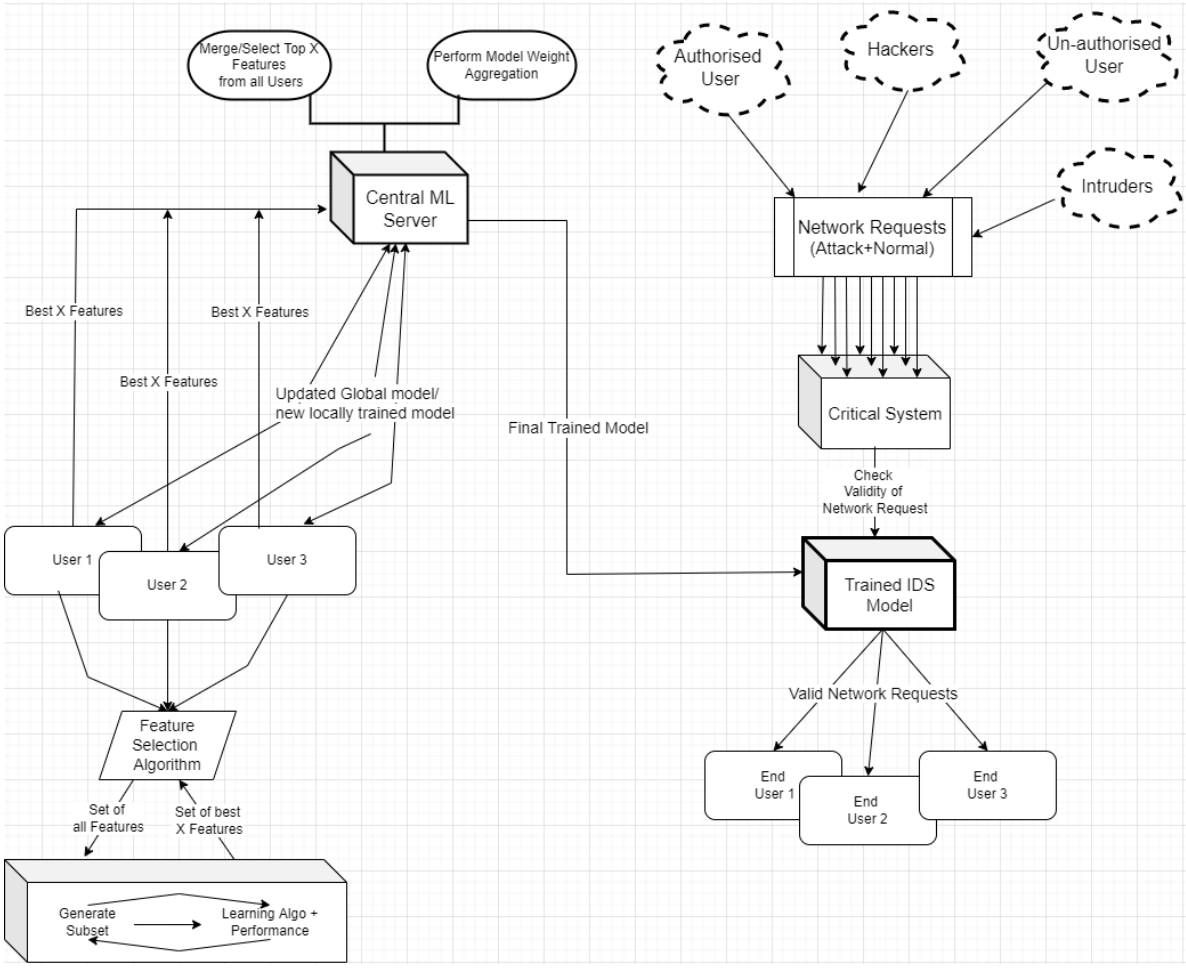

Fig 8.3-Level 2-DFD

9. RESULTS/OBSERVATIONS

In this section we will look into the results we have derived from testing our model against the CIC-IDS Dataset.

- **IDS Results without Feature Selection**

Firstly, we divided our dataset into 11 parts – one for each of the virtual clients. Then, we trained our model using Federated Machine Learning and tested it without using Feature Selection. In the dataset, we had 77 features and we simulated FML using 11 clients. The results are shown in the figure below:

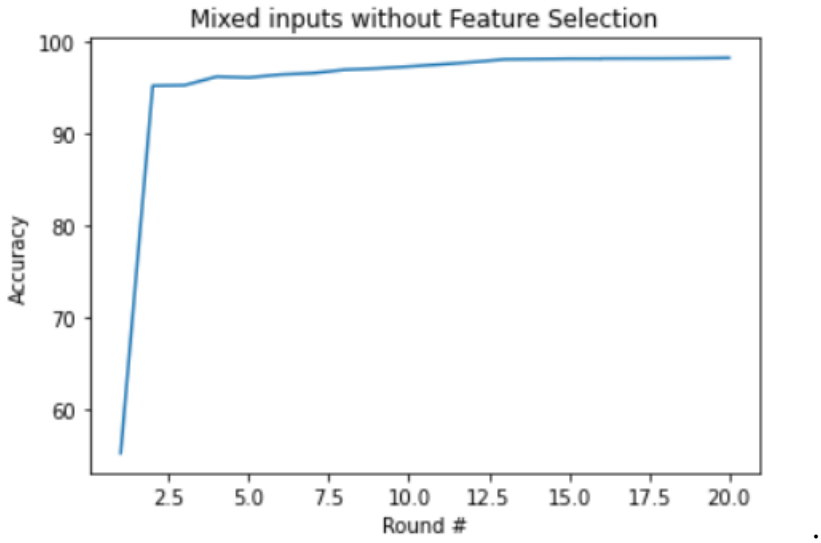

Fig 9.1 - IDS accuracy without Feature Selection

The accuracy of the model increased as the model trained through subsequent rounds, and it tend to flatten at the later rounds. Here, since the number of input features were large, the training time taken in each round was relatively high.

- **IDS Results with Feature Selection**

After that, we applied Feature Selection. Using feature selection we extracted top features from each client. Then on the server we selected the global top features on which the model will be trained and tested. In this simulation, we used 11 clients and top 30 global features.

The results are shown in the figure below:

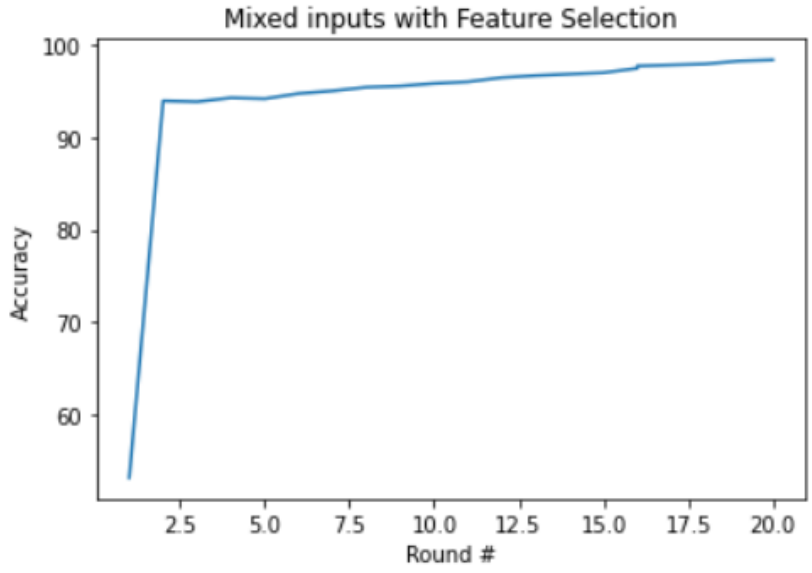

Fig 9.2 - IDS accuracy with Feature Selection

The graph shows the accuracy of the model for all types of attacks (mixed attacks). We also calculated the accuracy of the model for individual attacks, and the results are given below:

```
4462/4462 [==============================] - 8s 2ms/step - loss: 0.0458 - accuracy: 0.9820
#######-----Acccuracy for round  17 is  0.9819554686546326  ------########
2232/2232 [==============================] - 4s 2ms/step - loss: 0.0527 - accuracy: 0.9758
#######-----Acccuracy for attack  0 is  0.9758477807044983  ------########
8/8 [==============================] - 0s 3ms/step - loss: 1.3463 - accuracy: 0.5407
#######-----Acccuracy for attack  1 is  0.5406504273414612  ------########
510/510 [==============================] - 1s 2ms/step - loss: 0.0030 - accuracy: 0.9995
#######-----Acccuracy for attack  2 is  0.9995095729827881  ------########
1013/1013 [==============================] - 2s 2ms/step - loss: 0.0372 - accuracy: 0.9984
#######-----Acccuracy for attack  3 is  0.9983644485473633  ------########
32/32 [==============================] - 0s 2ms/step - loss: 0.0930 - accuracy: 0.9873
#######-----Acccuracy for attack  4 is  0.9872549176216125  ------########
1/1 [==============================] - 0s 19ms/step - loss: 0.3035 - accuracy: 1.0000
#######-----Acccuracy for attack  5 is  1.0  ------########
1/1 [==============================] - 0s 21ms/step - loss: 12.5334 - accuracy: 0.1111
#######-----Acccuracy for attack  6 is  0.1111111119389534  ------########
635/635 [==============================] - 1s 2ms/step - loss: 0.0033 - accuracy: 0.9995
#######-----Acccuracy for attack  7 is  0.9995071887969971  ------########
24/24 [==============================] - 0s 2ms/step - loss: 0.5648 - accuracy: 0.5059
#######-----Acccuracy for attack  8 is  0.5058977603912354  ------########
6/6 [==============================] - 0s 2ms/step - loss: 1.3371 - accuracy: 0.0904
#######-----Acccuracy for attack  9 is  0.0904255285859108  ------########
1/1 [==============================] - 0s 21ms/step - loss: 7.1499 - accuracy: 0.0000e+00
#######-----Acccuracy for attack  10 is  0.0  ------########
```

Fig 9.3 –Individual Attack Type Model Accuracy

Sr. No.	Attack Type	Accuracy	Test Data(Count)
1	DDoS	97.58%	2232
2	PortScan	54.07%	8
3	DoS	99.95%	510
4	Bot	99.84%	1013
5	SSH-Patator	98.73%	32
6	FTP-Patator	100%	1
7	Web Attack(XSS)	11%	1
8	Web Attack(BruteForce)	99.95%	635
9	Infiltration	50.59%	24
10	Heartbleed	9.04%	6
11	Web Attack(SQL Injection)	0.0%	1

Fig 9.3 – Individual Attack Accuracy and test data count

As we can see, for the attacks which had significant amount of data, the model was able to perform well, but for those which had negligible data-the accuracy is relatively low.

10. CONCLUSION AND FUTURE PLANS

10.1 Conclusion

Three important aspects have aided the huge success of machine learning techniques and associated applications. To begin, there is the availability of data accumulated through time. Second, technological advancements have enabled increased processing power, resulting in the creation of flexible and integrated devices that train data models quicker and at lower prices. Deep learning models have enabled self-learning, resulting in improved accuracy. However, there are a number of issues with data privacy and security.

The privacy of such data and linked equipment must be protected. Machine learning-based intrusion detection systems (IDS) are designed to meet this need. However, because to the necessity to store and send data to a centralised server, these frameworks continue to struggle to fulfil privacy and security standards. The adoption of federated learning-based IDS systems allows for the training of a high-quality centralised model while maintaining user privacy.

10.2 Future Plans

For future work, we would want to look into the methods of data compression - to reduce the packet size of the data packet that is transmitted to and fro between the central server and the edge devices, improve the feature selection and the ML server weight aggregation approaches that are being used currently.

11. REFERENCES

[1] Federated Learning for Intrusion Detection System: Concepts, Challenges and Future Directions Shaashwat Agrawal , Sagnik Sarkar , Ons Aouedi, Gokul Yenduri, Kandaraj Piamrat, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu.

[2] Federated Learning-Based Network Intrusion Detection with a Feature Selection Approach by Yang Qin & Masaaki Kondo The University of Tokyo Tokyo, Japan.

[3] DeepFed: Federated Deep Learning for Intrusion Detection in Industrial Cyber–Physical Systems Beibei Li, Member, IEEE, Yuhao Wu, Jiarui Song, Rongxing Lu, Senior Member, IEEE, Tao Li , Member, IEEE, and Liang Zhao , Member, IEEE.

[4] Federated Machine Learning: Concept and Applications QIANG YANG, Hong Kong University of Science and Technology, Hong Kong YANG LIU and TIANJIAN CHEN, Webank, China YONGXIN TONG, Beihang University, China.

[5] Intrusion Detection Using Machine Learning: A Comparison Study Saroj Kr. Biswas1 1CSE dept., NIT Silchar, Assam, India.

[6] Kang, S. H., and Kim, K. J., “A feature selection approach to find optimal feature subsets for the network intrusion detection system,” Cluster Computing, 19(1), pp. 325-333, 20162.

[7] Tsukada, M., Kondo, M., and Matsutani, H. , “A neural network based on-device learning anomaly detector for edge devices,” IEEE Transactions on Computers, 69(7), pp. 1027-1044, 2020.

[8] A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection Preeti Mishra; Vijay Varadharajan; Uday Tupakula; Emmanuel S. Pilli

[9] CICIDS2017 data. https://www.unb.ca/cic/datasets/ids-2017.html

[10] Network intrusion detection system: A systematic study of machine learning and deep learning approaches Zeeshan Ahmad, Adnan Shahid , Khan,Cheah Wai, Shiang,Johari Abdullah,Farhan AhmadZeeshan Ahmad,Adnan Shahid Khan,Cheah Wai Shiang,Johari Abdullah,Farhan Ahmad

APPENDIX

Simulating Federated Machine Learning with Feature Selection

```
import pickle
import time

import matplotlib as plt
import numpy as np
import pandas as pd
from google.colab import drive
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, f_classif
from tensorflow import keras

drive.mount('/content/drive')



class Client:
    def __init__(self, dataset_x, dataset_y, epoch_number, learning_rate,
                 weights, batch, features):
        self.dataset_x = dataset_x
        self.dataset_y = dataset_y
        self.epoch_number = epoch_number
        self.learning_rate = learning_rate
        self.weights = weights
        self.batch = batch
        self.features = features

    def train(self):
        model = keras.models.Sequential([
            keras.layers.Flatten(input_shape=[
                self.features,
```

```
        ]),
        keras.layers.Dense(200, activation='tanh'),
        keras.layers.Dense(100, activation='tanh'),
        keras.layers.Dense(12, activation='softmax')
    ])

    #setting weight of the model
    model.set_weights(self.weights)

    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=keras.optimizers.Adam(),
                  metrics=['accuracy'])
    history = model.fit(self.dataset_x,
                        self.dataset_y,
                        epochs=self.epoch_number,
                        batch_size=self.batch)

    #getting the final_weight
    output_weight = model.get_weights()
    return output_weight


file = open("/content/drive/MyDrive/Dataset/x_data", 'rb')
x_data = pickle.load(file)

file = open("/content/drive/MyDrive/Dataset/y_data", 'rb')
y_data = pickle.load(file)

file = open("/content/drive/MyDrive/Dataset/x_valid", 'rb')
x_valid = pickle.load(file)

file = open("/content/drive/MyDrive/Dataset/y_valid", 'rb')
y_valid = pickle.load(file)
```

```
def feature_selection(x_data, y_data, x_valid):
    arr = [0] * 77
    fvalue_Best = SelectKBest(f_classif, k=35)
    x = np.array(x_data)
    y = np.array(y_data)

    for idx in range(len(x)):
        X_kbest = fvalue_Best.fit_transform(x[idx], y[idx])
        i, j = 0, 0
        while (j < 78 and i < 35):
            if (eval(str(X_kbest[0][i])) == eval(str(x[idx][0][j]))):
                arr[j] += 1
                i += 1
            j += 1

    temp_arr = []
    for idx in range(len(arr)):
        temp_arr.append((arr[idx], idx))
    temp_arr.sort()
    temp_arr.reverse()

    feature_count = 20
    final_ft = []
    for idx in range(len(temp_arr)):
        if idx < feature_count:
            final_ft.append(temp_arr[idx][1])
        else:
            if (temp_arr[idx][0] == temp_arr[idx - 1][0]):
                final_ft.append(temp_arr[idx][1])
            else:
                break
```

```
    final_ft.sort()

    # Selecting the features from x_data.
    x_data = np.array(x_data)
    x_data = x_data[:, :, final_ft]

    # Selecting the features from x_valid.
    x_valid = np.array(x_valid)
    x_valid = x_valid[:, final_ft]
    # print(final_ft)
    return x_data, x_valid, len(final_ft)


def get_model(features):
    model = keras.models.Sequential([
        keras.layers.Flatten(input_shape=[
            features,
        ]),
        keras.layers.Dense(200, activation='tanh'),
        keras.layers.Dense(100, activation='tanh'),
        keras.layers.Dense(12, activation='softmax')
    ])

    return model


# Function for averaging.
def model_average(client_weights):
    average_weight_list = []
    for index1 in range(len(client_weights[0])):
        layer_weights = []
        for index2 in range(len(client_weights)):
            weights = client_weights[index2][index1]
```

```
            layer_weights.append(weights)
        average_weight = np.mean(np.array([x for x in layer_weights]), axis=0)
        average_weight_list.append(average_weight)
    return average_weight_list


def create_model(features):
    model = get_model(features)
    weight = model.get_weights()
    return weight


# Initializing the client automatically.

accuracy_list_global = []


def train_server(training_rounds, epoch, batch, learning_rate, x_data, x_valid,
                 features):
    # temp_variable
    # training_rounds=2
    # epoch=5
    # batch=128

    accuracy_list = []
    client_weight_for_sending = []

    for index1 in range(1, training_rounds):
        print('Training for round ', index1, 'started')
        client_weights_tobe_averaged = []
        for index in range(len(y_data)):
            print('-------Client-------', index)
            if index1 == 1:
```

```
            print(
                'Sharing Initial Global Model with Random Weight Initialization'
            )
            initial_weight = create_model(features)
            client = Client(x_data[index], y_data[index], epoch,
                            learning_rate, initial_weight, batch, features)
            weight = client.train()
            client_weights_tobe_averaged.append(weight)
        else:
            client = Client(x_data[index], y_data[index], epoch,
                            learning_rate,
                            client_weight_for_sending[index1 - 2], batch,
                            features)
            weight = client.train()
            client_weights_tobe_averaged.append(weight)

    # Calculating the avearge weight from all the clients.
    client_average_weight = model_average(client_weights_tobe_averaged)
    client_weight_for_sending.append(client_average_weight)

    # Validating the model with avearge weight.
    model = get_model(features)

    model.set_weights(client_average_weight)
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=keras.optimizers.SGD(lr=learning_rate),
                  metrics=['accuracy'])
    # print(x_valid)
    result = model.evaluate(x_valid, y_valid)
    accuracy = result[1]
    print('#######-----Acccuracy for round ', index1, 'is ', accuracy,
          ' ------########')
    accuracy_list.append(accuracy)
```

```
        accuracy_list_global.append(accuracy)


            for i in range(12):
            temp_y_valid = []
            temp_x_valid = []
            for ind, x in enumerate(y_valid):
                if x == i:
                    temp_y_valid.append(x)
                    temp_x_valid.append(list(x_valid[ind]))
            temp_y_valid = np.array(temp_y_valid)
            temp_x_valid = np.array(temp_x_valid)
            result = model.evaluate(temp_x_valid, temp_y_valid)
            accuracy = result[1]
            print('#######-----Acccuracy for attack ', i, 'is ', accuracy,
                  ' ------########')

    return accuracy_list, client_weight_for_sending


def train_main(x_data, x_valid, features):
    start = time.time()
    training_accuracy, weights = train_server(100, 5, 64, 0.01, x_data,
                                              x_valid, features)
    end = time.time()
    print('TOTAL TIME = ', end - start)


if __name__ == "__main__":
    x_data, x_valid, features = feature_selection(x_data, y_data, x_valid)
    # features = 77
    train_main(x_data, x_valid, features)
    pass
```

```
print(accuracy_list_global)
```

Code for cleaning and dividing the CIC-ISD Dataset

```
import pickle
from collections import Counter

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.utils import shuffle

path_list = [
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Tuesday-WorkingHours.pcap_ISCX.csv',
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Wednesday-workingHours.pcap_ISCX.csv',
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv',
    '/homme/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv',
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Friday-WorkingHours-Morning.pcap_ISCX.csv',
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv',
    '/home/keshav/Desktop/CIC AWS 2018/cicids 2017/cicids data/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv'
]


def reduce_data(data):
    label = list(data[' Label'].unique())
    data_count = data[' Label'].value_counts()
    attack_count = 0
```

```
    for count in data_count[1:]:
        attack_count = attack_count + count

    data1 = pd.DataFrame()
    for name in label:
        if name is label[0]:
            temp_data = data[data[' Label'] == name]
            temp_data = temp_data[:attack_count * 2]
            data1 = pd.concat([data1, temp_data], ignore_index=True)

        else:
            temp_data = data[data[' Label'] == name]
            data1 = pd.concat([data1, temp_data], ignore_index=True)

    return data1


def merge_data():
    data = pd.DataFrame()
    for path in path_list:
        # print(path)
        if path == path_list[1]:
            file = open('ddos_data.txt', 'rb')
            temp_data = pickle.load(file)
            reduced_data = reduce_data(temp_data)
            data = pd.concat([data, reduced_data], ignore_index=True)

        else:
            temp_data = pd.read_csv(path)
            reduced_data = reduce_data(temp_data)
            data = pd.concat([data, reduced_data], ignore_index=True)

        data = shuffle(data)
```

```
    return data


def get_data(data):
    column_list = data.columns
    info = data[column_list[-1]].value_counts()
    data_info = list(data[column_list[-1]].unique())
    label_length = len(data_info)
    attack_labels = data_info[1:label_length]

    # Appending malicious data to the dataframe.
    bruteforce_data = pd.DataFrame()
    for label in attack_labels:
        temp_data = data[data[column_list[-1]] == label]
        bruteforce_data = pd.concat([bruteforce_data, temp_data],
                                    ignore_index=True)

    # Appending benign data to the dataframe.
    benign_data_all = data[data[column_list[-1]] == data_info[0]]
    benign_data = benign_data_all[:len(bruteforce_data)]
    bruteforce_data = pd.concat([bruteforce_data, benign_data],
                                ignore_index=True)

    # Shuffling the dataset.
    bruteforce_data = shuffle(bruteforce_data)
    bruteforce_data.index = np.arange(0, len(bruteforce_data))

    # Dropping the destination host column.
    bruteforce_data = bruteforce_data.drop([column_list[0]], axis=1)

    # Splitting the dataset into X and Y.
    y_bruteforce = bruteforce_data[' Label']
```

```
    x_bruteforce = bruteforce_data.drop([' Label'], axis=1)

    # Handling infinite and nan values.
    x_bruteforce.replace([np.inf, -np.inf], np.nan, inplace=True)
    x_bruteforce.fillna(x_bruteforce.mean(), inplace=True)

    # Doing feature scaling.
    scaler = StandardScaler()
    x_bruteforce = scaler.fit_transform(x_bruteforce)
    x_bruteforce = pd.DataFrame(x_bruteforce)
    column_list = column_list.drop([column_list[0], column_list[-1]])
    x_bruteforce.columns = column_list

    # Label encoding.
    le = LabelEncoder()
    y_bruteforce = le.fit_transform(y_bruteforce)

    # Train dataset.
    x_train_bruteforce = x_bruteforce[:int(0.8 * (len(x_bruteforce)))]
    y_train_bruteforce = y_bruteforce[:int(0.8 * (len(y_bruteforce)))]

    # Test dataset.
    x_test_bruteforce = x_bruteforce[int(0.8 * (len(x_bruteforce))):]
    y_test_bruteforce = y_bruteforce[int(0.8 * (len(x_bruteforce))):]

    return x_train_bruteforce, y_train_bruteforce, x_test_bruteforce, y_test_bruteforce


def get_cic_data(data):
    file = open('final_data.txt', 'rb')
    data = pickle.load(file)
    x_train, y_train, x_test, y_test = get_data(data)
    return x_train, y_train, x_test, y_test
```

```
def divide_without_label(parts, X_train_full, y_train_full):
    each_part_number = int(len(X_train_full) / parts)
    list_x_train = []
    list_y_train = []

    number_list = []
    number_list.append(0)
    for x in range(1, parts + 1):
        number_list.append(each_part_number * x)

    for x in range(0, parts):
        data_x = X_train_full[number_list[x]:number_list[x + 1]]
        data_y = y_train_full[number_list[x]:number_list[x + 1]]
        list_x_train.append(data_x)
        list_y_train.append(data_y)

    return list_x_train, list_y_train



def main():
    data = merge_data()
    x_train, y_train, x_valid, y_valid = get_cic_data(data)
    x_data, y_data = divide_without_label(11, x_train, y_train)



if __name__ == "__main__":
    main()
```