Configuration Manual

MSc Research Project
Cybersecurity

Jawad Altaf
Student ID: 23203803

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jawad Altaf
Student ID:	X23203803
Programme:	: Masters in Cybersecurity **Year:** 2023-2024
Module:	Research in Computing
Lecturer:	Vikas Sahni
Submission Due Date:	12th August 2024
Project Title:	Enhancing Intrusion Detection using Federated Learning
Word Count:	1219 **Page Count: 19**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: : Jawad Altaf

Date: : 11th August

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jawad Altaf
X23203803

1 Introduction

The configuration manual explains the requirement to create the environment, implementation of the model, necessary hardware, software and code snippets used for completion of research work. The purpose of this manual is to demonstrate step by step coding procedure taken to perform the project. It will help to replicate and verify the results in future. The project is based on enhancing network intrusion detection using federated learning model using flower (open source federated frame work) GBHM, F.L. (2024) integrated with GRU (Gated recurrent model).

2 Implementation

2.1 Hardware

This section describes the hardware which is supported to conduct research utilising machine learning algorithms. Analysing large datasets with machine learning algorithms requires significant resource utilization. The following information provides idea of resource utilization:

- CPU: Intel 13th Gen core i5-1335U 1.30 ghz
- Installed RAM: 24GB
- Windows Edition: 11 Home
- Storage: 512 SSD

2.2 Software and tools

- Integrated Development Environment: Google Collaboratory (Google Collab) with High CPU and GPU Usage L4
- Coding Language Platform: Python 3.7
- Data Storage: PC/Google Drive

Dataset: The dataset used for this project was obtained from Kaggle (Herzalla, 2023)which was made network traffic pattern to facilitate the research and development of intrusion detection systems (IDS) in 2023. It has two parts: one in tabular representation of extracted in features csv format and other part provides raw network traffic pattern in PCAP files.

- The dataset was downloaded from the public repository.
- The complete TII-SSRC 23 dataset is 27.5 GB and has two major categories: benign and malicious, from eight diverse types of traffic.
- The traffic has been divided into 32 subtypes: six benign and 26 malicious. The dataset was comprised into raw network traffic data in the form of Packet Capture (PCAP)

files, as well as the extracted features in the form of Comma-Separated Values (CSV) files.

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Traffic Type	Traffic Subtype
0	192.168.1.90-192.168.1.3-53930-64738-6	192.168.1.90	53930.0	192.168.1.3	64738	6.0	01/01/1970 07:41:46 AM	52601173.0	1701.0	1793.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
1	192.168.1.3-192.168.1.90-64738-37700-6	192.168.1.3	64738.0	192.168.1.90	37700	6.0	01/01/1970 07:41:46 AM	119106942.0	36.0	57.0	...	3.416174e+06	19996926.0	14078617.0	5001511.0	1737.400069	5003516.0	5000449.0	Benign	Audio	Audio
2	192.168.1.3-192.168.1.90-22-40854-6	192.168.1.3	22.0	192.168.1.90	40854	6.0	01/01/1970 07:41:46 AM	5589.0	1.0	1.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
3	192.168.1.70-192.168.1.3-55422-64738-6	192.168.1.70	55422.0	192.168.1.3	64738	6.0	01/01/1970 07:41:47 AM	118166562.0	3932.0	4196.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
4	192.168.1.90-192.168.1.3-59658-64738-17	192.168.1.90	59658.0	192.168.1.3	64738	17.0	01/01/1970 07:41:50 AM	119988385.0	25.0	6795.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio

5 rows × 86 columns

Figure 1: Snapshot of the Dataset used for this project

2.3 Datafiles used for the Analysis

The file used for this project are listed below:

- Project Centralized model.ipynb: coding files which was loaded into Google Collaboratory.
- Federated Learning Model.ipynb: coding file loaded into Google Collaboratory for federated learning.
- Dataset: network_intrusion_dataset.csv

2.4 Python Libraries

The research project employed Python as the coding language to configure and run the model. Multiple Python libraries were imported in Google Collaboratory- IDE for various functions used in the study.
Pandas

- **pandas:** For data analysis.
- **matplotlib.pyplot**: For the graph plotting and visualizations.
- **google.colab.drive:** For accessing Google Drive.
- **sklearn.preprocessing:** For data preprocessing, including label encoding and scaling.
- **sklearn.model_selection:** For splitting data into training and test sets.
- **sklearn.feature_selection:** For feature selection techniques.
- **imblearn.under_sampling**: For undersampling techniques to handle imbalanced datasets.
- **sklearn.decomposition:** For Principal Component Analysis (PCA).
- **torch:** For constructing and training neural networks.
- **torch.nn:** For neural network layers and operations.
- **torch.optim:** For optimization algorithms.
- **sklearn.metrics:** For evaluating model performance.
- **seaborn:** For statistical visualization.
- **flwr (Flower):** For federated learning framework.
- **numpy:** For numerical operations and handling arrays.

3 Data Preprocessing

Following are the steps of data preprocessing as shown below:

Step 1: Uploading dataset in google Collaboratory

```python
import pandas as pd
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Now try reading the file again
file_path = '/content/drive/My Drive/network_intrusion_dataset.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the dataframe to ensure it's loaded correctly
df.head()
```

Figure 2: Code Snippet for Uploading of Dataset

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Traffic Type	Traffic Subtype
0	192.168.1.90-192.168.1.3-53930-64738-6	192.168.1.90	53930.0	192.168.1.3	64738	6.0	01/01/1970 07:41:46 AM	52601173.0	1701.0	1793.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
1	192.168.1.3-192.168.1.90-64738-37700-6	192.168.1.3	64738.0	192.168.1.90	37700	6.0	01/01/1970 07:41:46 AM	119106942.0	36.0	57.0	...	3.416174e+06	19996926.0	14078617.0	5001511.0	1737.400069	5003516.0	5000449.0	Benign	Audio	Audio
2	192.168.1.3-192.168.1.90-22-40854-6	192.168.1.3	22.0	192.168.1.90	40854	6.0	01/01/1970 07:41:46 AM	5589.0	1.0	1.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
3	192.168.1.70-192.168.1.3-55422-64738-6	192.168.1.70	55422.0	192.168.1.3	64738	6.0	01/01/1970 07:41:47 AM	118166562.0	3932.0	4196.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio
4	192.168.1.90-192.168.1.3-59658-64738-17	192.168.1.90	59658.0	192.168.1.3	64738	17.0	01/01/1970 07:41:50 AM	119988385.0	25.0	6795.0	...	0.000000e+00	0.0	0.0	0.0	0.000000	0.0	0.0	Benign	Audio	Audio

5 rows × 86 columns

Figure 3: Snapshot of the Dataset used in the Project

Step 2: Checking of Null Values in Dataset

```python
# Check for missing values in the dataset
missing_values = df.isnull().sum()
print(missing_values)
```

```
Flow ID             0
Src IP              0
Src Port            0
Dst IP              0
Dst Port            0
                   ..
Idle Max            0
Idle Min            0
Label               0
Traffic Type        0
Traffic Subtype     0
Length: 86, dtype: int64
```

Figure 4: Code Snippet and Null Values Status in Dataset

Step 3: Checking of Unique Values of Label Column, Traffic type, Traffic Subtypes, Identifying of attack and non-attack counts.

```
# Check unique values in the 'Label' column
label_counts = df['Label'].value_counts()
print("Label counts:")
print(label_counts)

# Check unique values in the 'Traffic Type' column
traffic_type_counts = df['Traffic Type'].value_counts()
print("\nTraffic Type counts:")
print(traffic_type_counts)

# Check unique values in the 'Traffic Subtype' column
traffic_subtype_counts = df['Traffic Subtype'].value_counts()
print("\nTraffic Subtype counts:")
print(traffic_subtype_counts)

# Identify attack and non-attack counts
attack_count = df[df['Label'] != 'Benign'].shape[0]
non_attack_count = df[df['Label'] == 'Benign'].shape[0]

print(f"\nNumber of attack instances: {attack_count}")
print(f"Number of non-attack (benign) instances: {non_attack_count}")
```

Figure 5: Code Snippet for Unique Values in Dataset

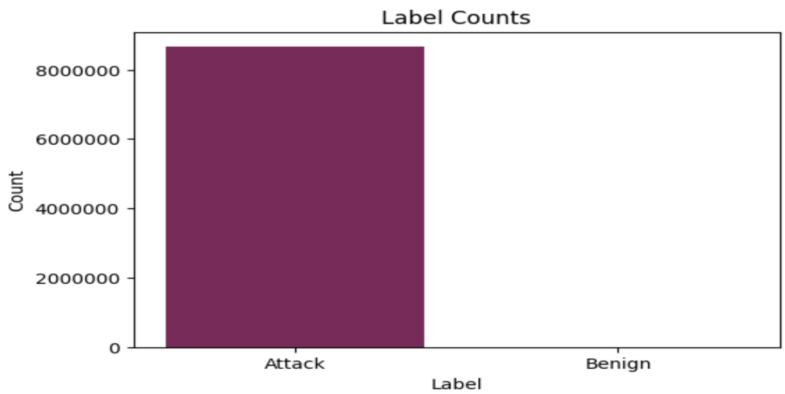


Figure 6: Label Count for Attack and Benign Traffic in Dataset

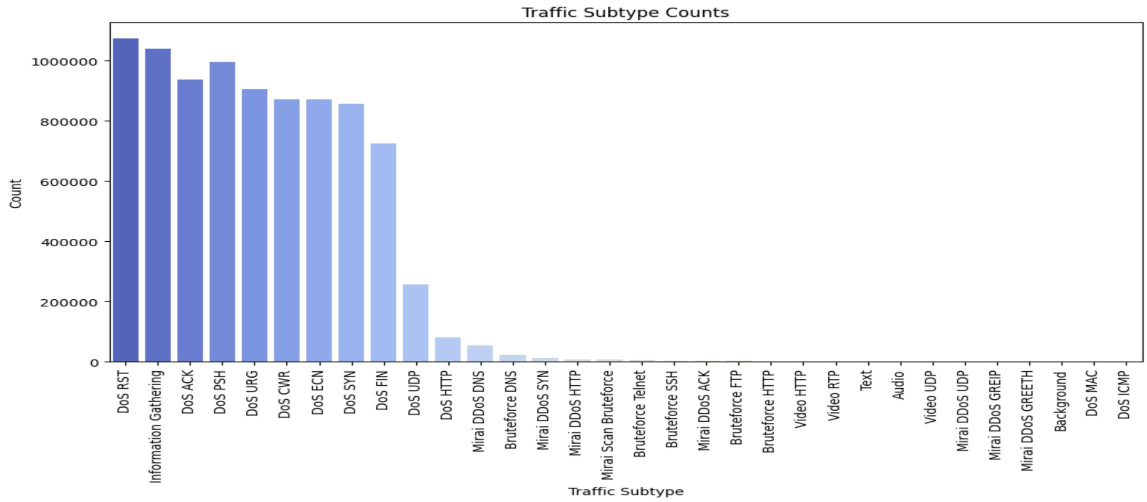


Figure 7: Traffic Subtype Classification in the Dataset

```
Label counts:
Label
Malicious    8655466
Benign          1301
Name: count, dtype: int64

Traffic Type counts:
Traffic Type
DoS                      7490929
Information Gathering    1038363
Mirai                      91002
Bruteforce                 35172
Video                        870
Text                         209
Audio                        190
Background                    32
Name: count, dtype: int64
```

Step 4: Checking of Datatypes

```
# Display the data types of each column
print("Data types of each column:")
print(df.dtypes)
```

Figure 8: Code Snippets for the Datatype Checking

```
Data types of each column:
Flow ID               object
Src IP                object
Src Port             float64
Dst IP                object
Dst Port               int64
                       ...
Idle Max             float64
Idle Min             float64
Label                 object
Traffic Type          object
Traffic Subtype       object
Length: 86, dtype: object
```

Figure 9: Result of Columns datatypes

Step 5: Identification of non-numeric columns

```python
# Identify non-numeric columns
non_numeric_columns = df.select_dtypes(include=['object']).columns
# Print non-numeric columns
print("Non-numeric columns:", non_numeric_columns)
```

```
Non-numeric columns: Index(['Flow ID', 'Src IP', 'Dst IP', 'Timestamp', 'Label', 'Traffic Type',
       'Traffic Subtype'],
      dtype='object')
```

Figure 10: Code Snippet and non-numeric Columns result

Step 6: Applying Label Encoding for non-numeric columns.

```python
from sklearn.preprocessing import LabelEncoder
# Identify non-numeric columns
non_numeric_columns = df.select_dtypes(include=['object']).columns

# Apply label encoding to non-numeric columns
label_encoders = {}
for column in non_numeric_columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Display the first few rows of the updated DataFrame
print(df.head())

# Display the data types of the updated DataFrame
print(df.dtypes)
```

Figure 11: Code Snippet for Label Encoding of non-numeric Columns

Step 7: Checking and Removing of Duplicate Entries

```
# Checking for duplicate rows in the DataFrame
duplicate_rows = df[df.duplicated()]

# Displaying the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)
```

```
[1142 rows x 86 columns]
```

```
# Dropping duplicate rows from the DataFrame
df = df.drop_duplicates()

# Displaying the DataFrame after dropping duplicates
print("DataFrame after dropping duplicates:")
print(df)
```

Step 8: Splitting of Dataset in testing and training and applying feature selection method using SelectKbest method. Features are displayed according to score.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif

# Assuming 'df' is already defined and contains your features and labels
# Separating features and the target label
X = df.drop('Label', axis=1)  # adjust this if your label column is named differently
y = df['Label']

# Splitting the dataset into training and testing sets with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)

# Standard scaling to normalize the feature set
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Feature selection using SelectKBest
selector = SelectKBest(score_func=f_classif, k=20)  # Change 'k' to the number of features you want to keep
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Identifying which features were selected by SelectKBest
features_selected = selector.get_support(indices=True)
selected_feature_names = X.columns[features_selected]

# Printing selected feature names
print("Selected features:", selected_feature_names.tolist())

# Plotting the selected features and their scores
feature_scores = selector.scores_[features_selected]
features_df = pd.DataFrame({'Feature': selected_feature_names, 'Score': feature_scores})

# Sorting the features by score
features_df = features_df.sort_values(by='Score', ascending=False)

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.barh(features_df['Feature'], features_df['Score'], color='skyblue')
plt.xlabel('Score')
plt.title('Feature Scores by SelectKBest')
plt.gca().invert_yaxis()
plt.show()
```

Figure 12: Code Snippet for Features Selection

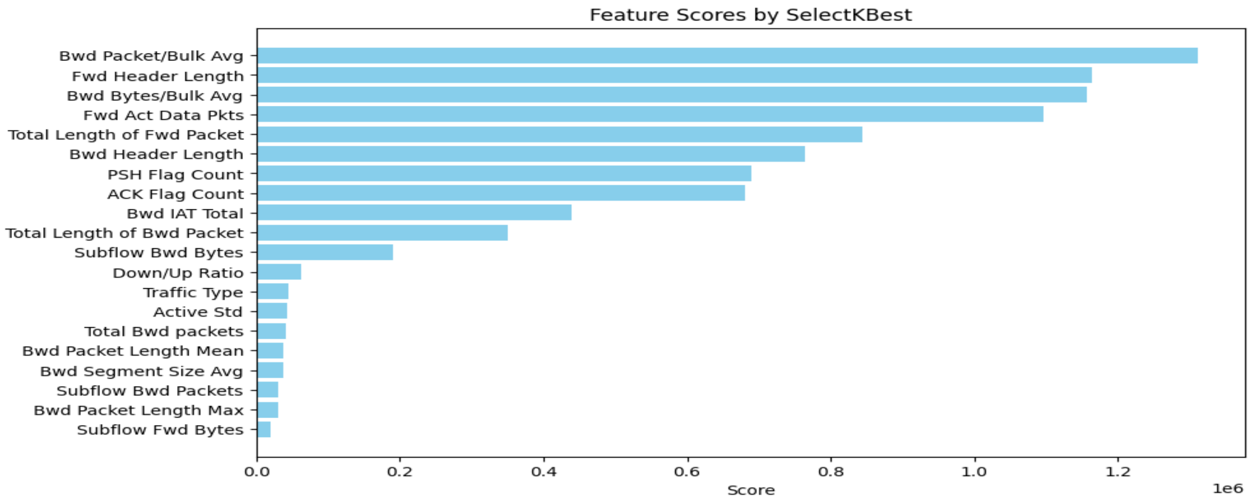


Figure 13: Selected Features Columns from the Dataset

Step 8: Distribution of Traffic: (Malicious and Benign)

```
Label
1    8655466
0       1301
Name: count, dtype: int64
```

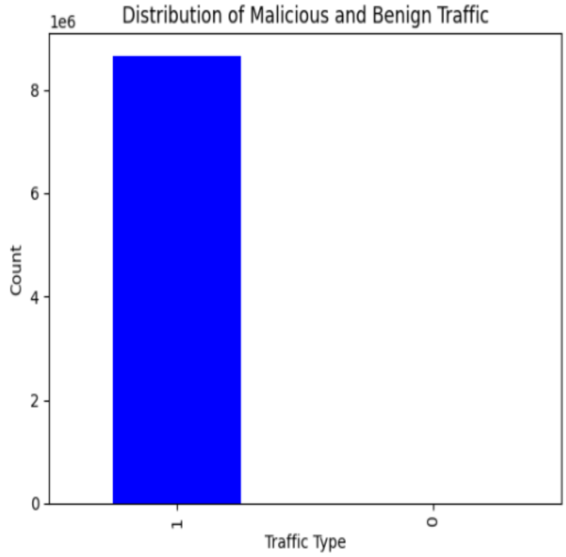
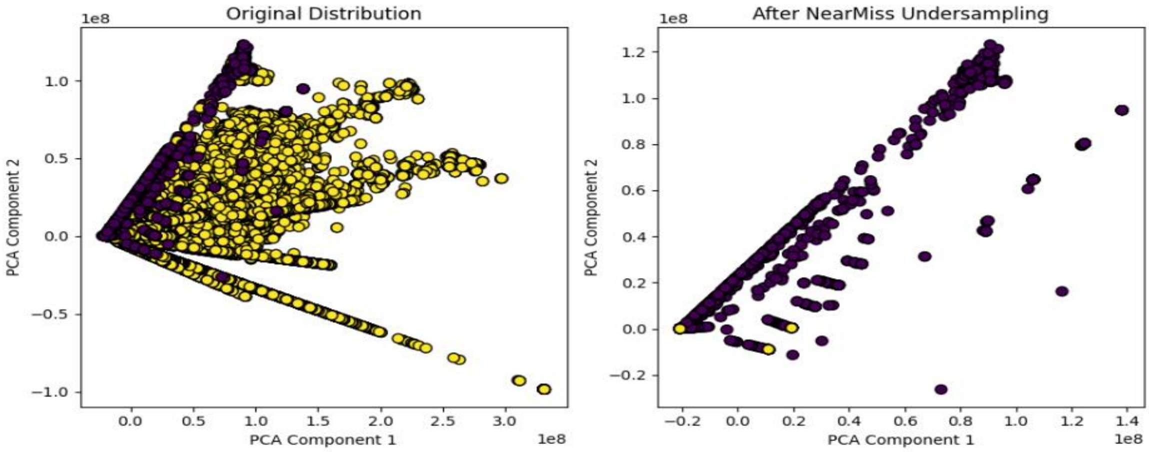


Figure 14: Plot of Malicious and Benign Traffic

Step 9: Applying Sampling Technique using Near miss and PCA Method.




```
Traffic Counts after resampling:
Label
0    1301
1    1301
Name: count, dtype: int64

Traffic Percentages after resampling:
Label
0    50.0
1    50.0
Name: count, dtype: float64
```

Figure 15: Results of undersampling Technique

4 Classification Models

The research work consists of 2 classification models using machine learning Algorithm

- Centralized learning integrated with GRU model analysed with 50, 100, 150 Epochs
- Federated learning model with flower framework integrated with GRU along with multiple clients with 50, 100, 150 Epochs.

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Assuming df_resampled is already defined
# Split data into features and labels
X = df_resampled.drop('Label', axis=1).values
y = df_resampled['Label'].values

# Initial train-test split
def initial_split(X, y, test_size=0.2):
    from sklearn.model_selection import train_test_split
    classes = np.unique(y)
    if len(classes) < 2:
        raise ValueError(f"Dataset has only one class: {classes}. Please use a dataset with multiple classes.")
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, stratify=y, random_state=42)
    return X_train, X_test, y_train, y_test

try:
    X_train, X_test, y_train, y_test = initial_split(X, y)
except ValueError as e:
    print(e)
    exit()

# Convert to PyTorch tensors if not already done
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)

# Step 1: Define the GRU Model
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GRUModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.gru = nn.GRU(input_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_dim).to(x.device)
        out, _ = self.gru(x, h0)
        out = self.fc(out[:, -1, :])
        return out
```

Figure 16: Code Snippet for Centralized Learning I

```python
input_dim = X_train.shape[1]
hidden_dim = 128  # Reduced for faster training
output_dim = len(np.unique(y_train))
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = GRUModel(input_dim, hidden_dim, output_dim).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Step 2: Train the Centralized Model
def train_model(model, X_train, y_train, criterion, optimizer, epochs=50, batch_size=32):
    model.train()
    for epoch in range(epochs):
        permutation = torch.randperm(X_train.size(0))
        for i in range(0, X_train.size(0), batch_size):
            indices = permutation[i:i+batch_size]
            batch_x, batch_y = X_train[indices], y_train[indices]
            optimizer.zero_grad()
            outputs = model(batch_x.unsqueeze(1).to(device))
            loss = criterion(outputs, batch_y.to(device))
            loss.backward()
            optimizer.step()
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item()}")

train_model(model, X_train, y_train, criterion, optimizer)
```

Figure 17: Code Snippet for Centralized Learning II

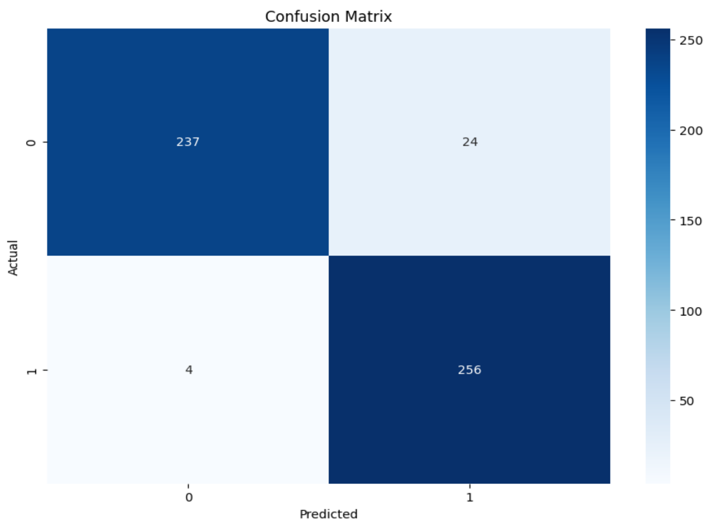
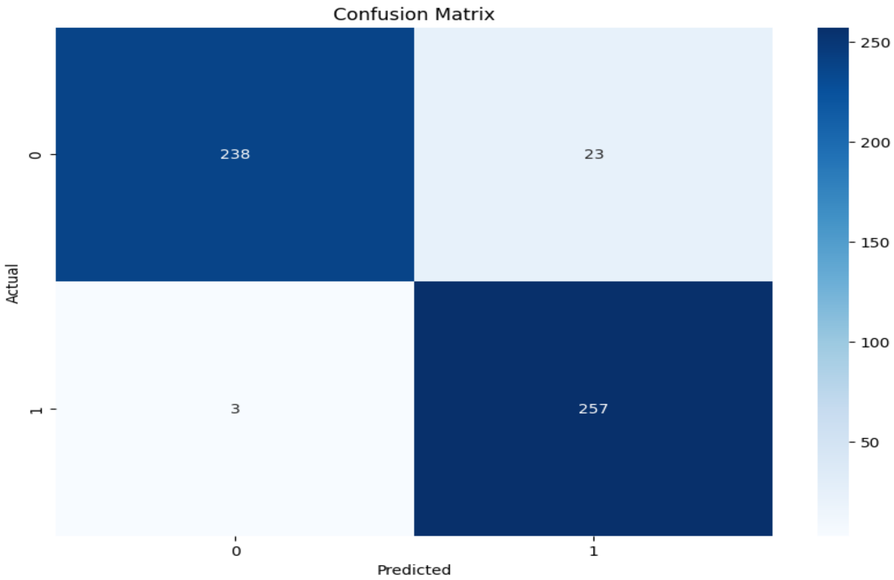


Figure 18: Confusion matrix for Centralized Model with 50 Epochs



;

Figure 19 : Confusion matrix for Centralized Model with 100 Epochs

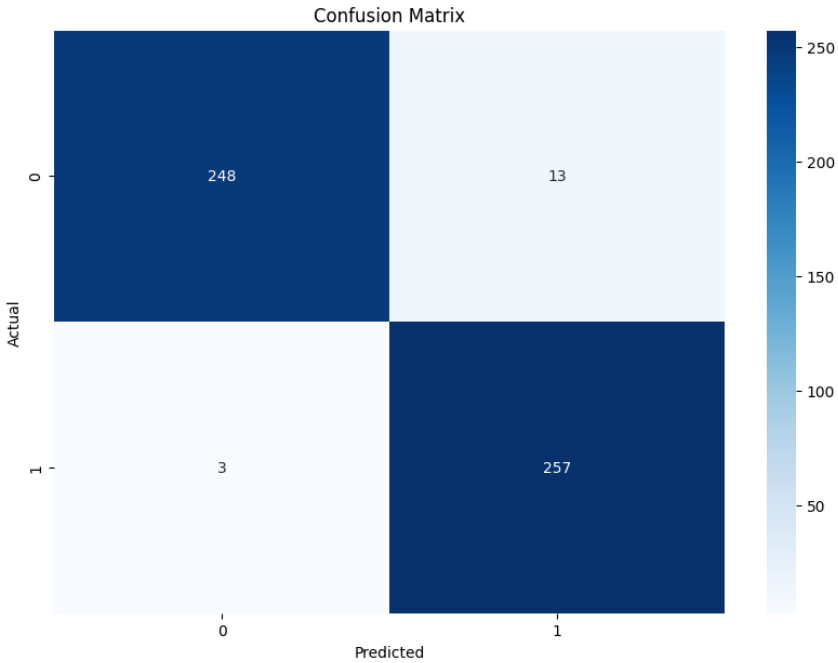


Figure 20 : Centralized Model Confusion Matrix with 150 Epochs

4.1 Installation of Flower framework in Python

```
pip install flwr
```

```
Collecting flwr
  Downloading flwr-1.10.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: cryptography<43.0.0,>=42.0.4 in /usr/local/lib/python3.10/dist-packages (from flwr) (42.0.8)
Requirement already satisfied: grpcio!=1.64.2,!=1.65.1,<2.0.0,>=1.60.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (1.64.1)
Collecting iterators<0.0.3,>=0.0.2 (from flwr)
  Downloading iterators-0.0.2-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: numpy<2.0.0,>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (1.26.4)
Collecting pathspec<0.13.0,>=0.12.1 (from flwr)
  Downloading pathspec-0.12.1-py3-none-any.whl.metadata (21 kB)
Collecting protobuf<5.0.0,>=4.25.2 (from flwr)
  Downloading protobuf-4.25.4-cp37-abi3-manylinux2014_x86_64.whl.metadata (541 bytes)
Collecting pycryptodome<4.0.0,>=3.18.0 (from flwr)
  Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Requirement already satisfied: tomli<3.0.0,>=2.0.1 in /usr/local/lib/python3.10/dist-packages (from flwr) (2.0.1)
Collecting tomli-w<2.0.0,>=1.0.0 (from flwr)
  Downloading tomli_w-1.0.0-py3-none-any.whl.metadata (4.9 kB)
Collecting typer<0.10.0,>=0.9.0 (from typer[all]<0.10.0,>=0.9.0->flwr)
  Downloading typer-0.9.4-py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography<43.0.0,>=42.0.4->flwr) (1.16.0)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.9.0->typer[all]<0.10.0,>=0.9.0->flwr) (8.1.7)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.9.0->typer[all]<0.10.0,>=0.9.0->flwr) (4.12.2)
Collecting colorama<0.5.0,>=0.4.3 (from typer[all]<0.10.0,>=0.9.0->flwr)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: shellingham<2.0.0,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (1.5.4)
Requirement already satisfied: rich<14.0.0,>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (13.7.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography<43.0.0,>=42.0.4->flwr) (2.22)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (0.1.2)
Downloading flwr-1.10.0-py3-none-any.whl (421 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 421.5/421.5 kB 29.1 MB/s eta 0:00:00
Downloading iterators-0.0.2-py3-none-any.whl (3.9 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Downloading protobuf-4.25.4-cp37-abi3-manylinux2014_x86_64.whl (294 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.6/294.6 kB 27.2 MB/s eta 0:00:00
Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 84.8 MB/s eta 0:00:00
Downloading tomli_w-1.0.0-py3-none-any.whl (6.0 kB)
Downloading typer-0.9.4-py3-none-any.whl (45 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 4.8 MB/s eta 0:00:00
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: typer, tomli-w, pycryptodome, protobuf, pathspec, iterators, colorama, flwr
  Attempting uninstall: typer
    Found existing installation: typer 0.12.3
    Uninstalling typer-0.12.3:
      Successfully uninstalled typer-0.12.3
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.3
    Uninstalling protobuf-3.20.3:
      Successfully uninstalled protobuf-3.20.3
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tensorflow-metadata 1.15.0 requires protobuf<4.21,>=3.20.3; python_version < "3.11", but you have protobuf 4.25.4 which is incompatible.
Successfully installed colorama-0.4.6 flwr-1.10.0 iterators-0.0.2 pathspec-0.12.1 protobuf-4.25.4 pycryptodome-3.20.0 tomli-w-1.0.0 typer-0.9.4
```

Figure 21 : Flower Framework Installation in Python

4.2 Upgrading of Flower

```
pip install --upgrade flwr
```

```
Requirement already satisfied: flwr in /usr/local/lib/python3.10/dist-packages (1.10.0)
Requirement already satisfied: cryptography<43.0.0,>=42.0.4 in /usr/local/lib/python3.10/dist-packages (from flwr) (42.0.8)
Requirement already satisfied: grpcio!=1.64.2,!=1.65.1,<2.0.0,>=1.60.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (1.64.1)
Requirement already satisfied: iterators<0.0.3,>=0.0.2 in /usr/local/lib/python3.10/dist-packages (from flwr) (0.0.2)
Requirement already satisfied: numpy<2.0.0,>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (1.26.4)
Requirement already satisfied: pathspec<0.13.0,>=0.12.1 in /usr/local/lib/python3.10/dist-packages (from flwr) (0.12.1)
Requirement already satisfied: protobuf<5.0.0,>=4.25.2 in /usr/local/lib/python3.10/dist-packages (from flwr) (4.25.4)
Requirement already satisfied: pycryptodome<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (3.20.0)
Requirement already satisfied: tomli<3.0.0,>=2.0.1 in /usr/local/lib/python3.10/dist-packages (from flwr) (2.0.1)
Requirement already satisfied: tomli-w<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from flwr) (1.0.0)
Requirement already satisfied: typer<0.10.0,>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (0.9.4)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography<43.0.0,>=42.0.4->flwr) (1.16.0)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.9.0->typer[all]<0.10.0,>=0.9.0->flwr) (8.1.7)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.9.0->typer[all]<0.10.0,>=0.9.0->flwr) (4.12.2)
Requirement already satisfied: colorama<0.5.0,>=0.4.3 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (0.4.6)
Requirement already satisfied: shellingham<2.0.0,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (1.5.4)
Requirement already satisfied: rich<14.0.0,>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]<0.10.0,>=0.9.0->flwr) (13.7.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography<43.0.0,>=42.0.4->flwr) (2.22)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14.0.0,>=10.11.0->typer[all]<0.10.0,>=0.9.0->flwr) (0.1.2)
```

Figure 22 : Upgrading of Flower Part 1

```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import flwr as fl

# Split the resampled data into training and testing sets
X_train_res, X_test_res, y_train_res, y_test_res = train_test_split(X_res, y_res, test_size=0.20, random_state=42, stratify=y_res)

# Convert training and testing data to tensors
X_train_tensor = torch.tensor(X_train_res.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_res.values, dtype=torch.long)
X_test_tensor = torch.tensor(X_test_res.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test_res.values, dtype=torch.long)

# Define the GRU Model with Dropout
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_prob=0.5):
        super(GRUModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.gru = nn.GRU(input_dim, hidden_dim, batch_first=True)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_dim).to(x.device)
        out, _ = self.gru(x, h0)
        out = self.dropout(out[:, -1, :])
        out = self.fc(out)
        return out

# Function to initialize the model, criterion, and optimizer
def initialize_model(input_dim, hidden_dim, output_dim, dropout_prob, l2_lambda, lr=0.01):
    model = GRUModel(input_dim, hidden_dim, output_dim, dropout_prob).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=l2_lambda)
    return model, criterion, optimizer
```

Figure 23 : Federated Learning I

```python
# Flower client definition
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, model, X_train, y_train, X_test, y_test, criterion, optimizer):
        self.model = model
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.criterion = criterion
        self.optimizer = optimizer

    def get_parameters(self):
        return [val.cpu().numpy() for _, val in self.model.state_dict().items()]

    def set_parameters(self, parameters):
        params_dict = zip(self.model.state_dict().keys(), parameters)
        state_dict = {k: torch.tensor(v) for k, v in params_dict}
        self.model.load_state_dict(state_dict, strict=True)

    def fit(self, parameters, config):
        self.set_parameters(parameters)
        self.model.train()
        epochs = config.get("epochs", 50)  # Default to 50 epochs if not specified
        batch_size = config.get("batch_size", 32)  # Default to batch size of 32 if not specified
        for epoch in range(epochs):
            permutation = torch.randperm(self.X_train.size(0))
            for i in range(0, self.X_train.size(0), batch_size):
                indices = permutation[i:i+batch_size]
                batch_x, batch_y = self.X_train[indices], self.y_train[indices]
                self.optimizer.zero_grad()
                outputs = self.model(batch_x.unsqueeze(1).to(device))
                loss = self.criterion(outputs, batch_y.to(device))
                loss.backward()
                self.optimizer.step()
        return self.get_parameters(), len(self.X_train), {}
```

Figure 24 : Federated Learning Code Part II

```python
    def evaluate(self, parameters, config):
        self.set_parameters(parameters)
        self.model.eval()
        with torch.no_grad():
            outputs = self.model(self.X_test.unsqueeze(1).to(device))
            loss = self.criterion(outputs, self.y_test.to(device)).item()
            _, predicted = torch.max(outputs.data, 1)
            accuracy = accuracy_score(self.y_test.cpu(), predicted.cpu())
        return float(loss), len(self.X_test), {"accuracy": float(accuracy)}

# Define your model dimensions and device
input_dim = X_train_res.shape[1]
hidden_dim = 128
output_dim = len(np.unique(y_res))
dropout_prob = 0.5
l2_lambda = 0.01
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Initialize the model, criterion, and optimizer
model, criterion, optimizer = initialize_model(input_dim, hidden_dim, output_dim, dropout_prob, l2_lambda)

# Start multiple Flower clients
clients = []
for _ in range(2):
    client = FlowerClient(model, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, criterion, optimizer)
    clients.append(client)

# Function to start the clients
def start_client(client):
    try:
        fl.client.start_numpy_client("127.0.0.1:8080", client=client)
    except Exception as e:
        print(f"Error starting Flower client: {e}")
```

Figure 25 : Federated Learning Code III

```python
import threading

# Configuration for training
fit_config = {"epochs": 50, "batch_size": 32}

# Start all clients in separate threads
threads = []
for client in clients:
    thread = threading.Thread(target=start_client, args=(client,))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()
```

Figure 26 : Federated Learning Code IV

5 Evaluation

Python code was used to generate statistical output for the above-mentioned classifier in the form of confusion matrix.

```
# Step 3: Evaluate the Centralized Model with Performance Metrics
y_pred = model(X_test.unsqueeze(1).to(device)).argmax(dim=1).cpu().numpy()
conf_matrix = confusion_matrix(y_test.cpu().numpy(), y_pred)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calculate accuracy
accuracy = accuracy_score(y_test.cpu().numpy(), y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Print classification report
report = classification_report(y_test.cpu().numpy(), y_pred)
print(report)
```

Figure 27 : Code Snippet for Centralized Model Classification

```
# Evaluate the model with a confusion matrix
model.eval()
with torch.no_grad():
    y_pred = model(X_test_tensor.unsqueeze(1).to(device)).argmax(dim=1).cpu().numpy()
conf_matrix = confusion_matrix(y_test_tensor.cpu().numpy(), y_pred)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print(classification_report(y_test_tensor.cpu().numpy(), y_pred))
```

Figure 28 : Code Snippet for Federated Learning

Figure 29 : Results of Confusion Matrix for the Federated Model with Multiple Clients

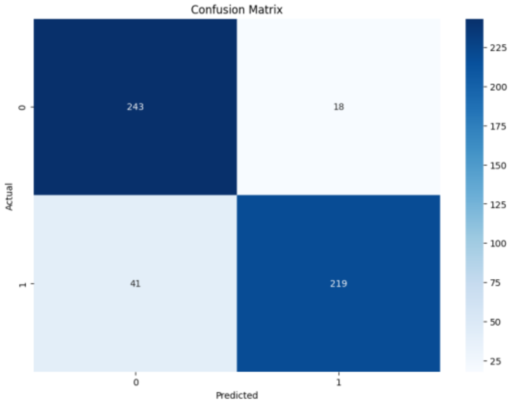


Figure 1 FL Model with 2 Clients and 50 Epochs

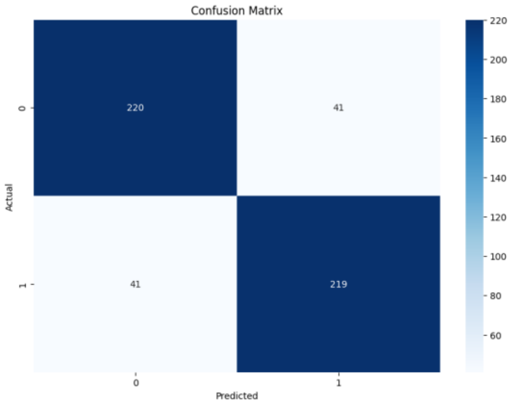


Figure 2 FL Model with 2 Clients and 100 Epochs

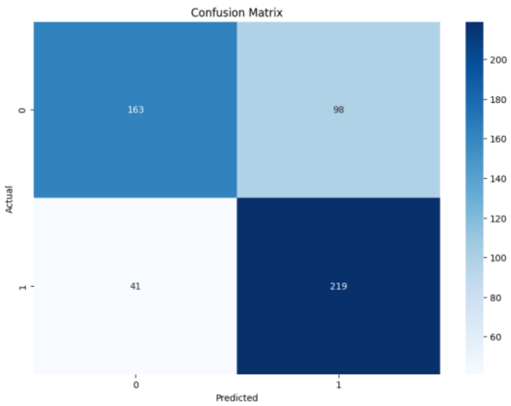


Figure 3 FL Model with 2 Clients and 150 Epochs

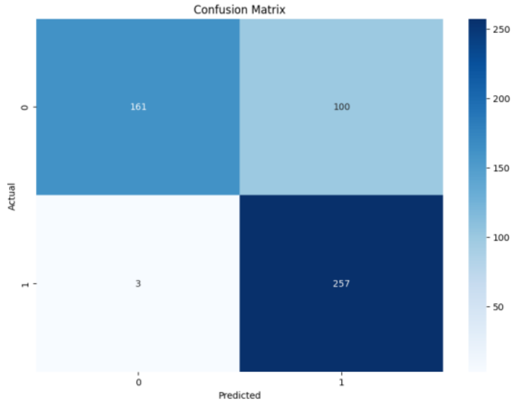


Figure 4 FL Model with 4 Clients and 50 Epochs

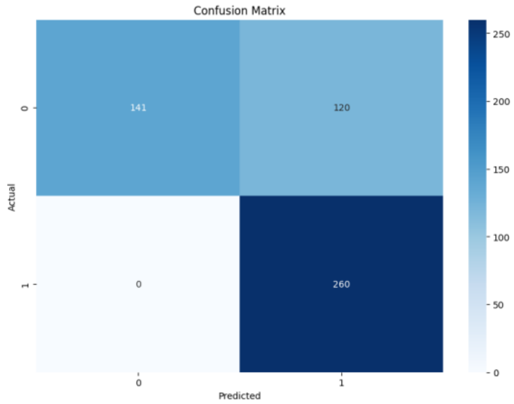


Figure 5 FL Model with 4 Clients and 100 Epochs

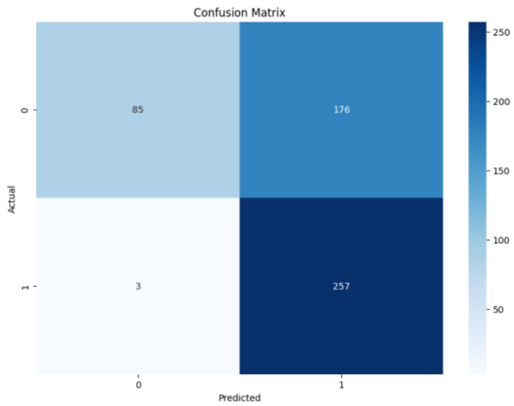


Figure 6 FL Model with 4 Clients and 150 Epochs

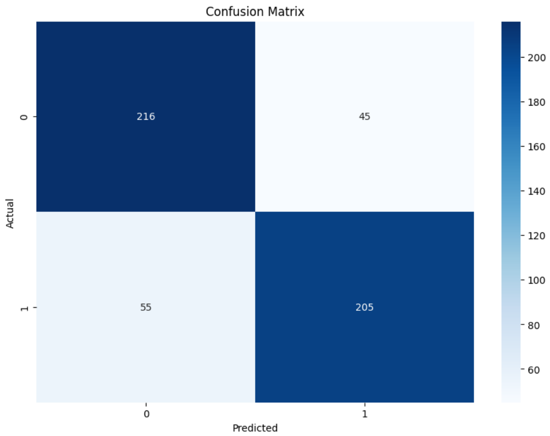


Figure 30 : FL Model with 5 Clients and 50 Epochs

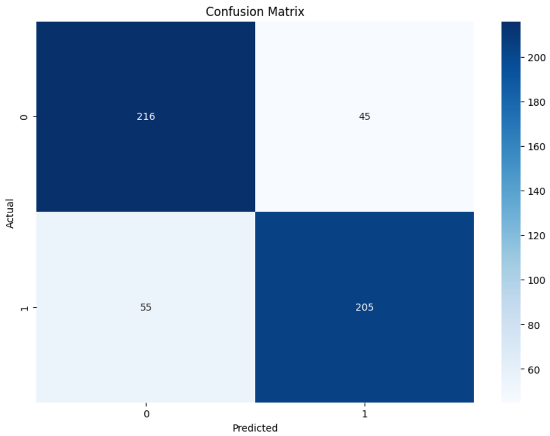


Figure 31 : FL Model with 5 Clients and 150 Epochs

References

Herzalla, D., 2023. *TII-SSRC-23 Dataset.*. [Online]
Available at: https://www.kaggle.com/datasets/daniaherzalla/tii-ssrc-23
[Accessed 2024]
GBHM, F.L. (2024). Flower Framework main. [online] flower.ai. Available at: https://flower.ai/docs/framework/tutorial-series-get-started-with-flower-pytorch.html.