Enhancing IoT Security through Anomaly-based Intrusion Detection Systems

MSc Research Project
MSc in Data Analytics

Muhammed Musthafa Keloth Poyil
Student ID: x23162112

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Muhammed Musthafa Keloth Poyil
Student ID:	x23162112
Programme:	MSc Data Analytics
Year:	2024-2025
Module:	Research Project
Supervisor:	Furqan Rustam
Submission Due Date:	12/12/2024
Project Title:	Enhancing IoT Security through Anomaly-based Intrusion Detection Systems
Word Count:	XXX
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:	Muhammed Musthafa Keloth Poyil
Date:	10th December 2024

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

Muhammed Musthafa Keloth Poyil
x23162112

1 Introduction

This configuration manual outlines the prerequisites, setup, and execution details required to replicate the results of this project. The project utilises various machine learning and deep learning models for IoT intrusion detection. Key stages include data preprocessing, feature engineering, model training, evaluation, and results visualisation. This document covers the software and hardware setup, libraries used, code configuration, and steps to execute the project.

2 System Requirements

2.1 Hardware Requirements

- Processor: Ryzen 7 or equivalent
- RAM: 8 GB
- Storage: 256 GB SSD
- GPU: AMD Radeon RX Vega 10 Graphics (or equivalent)

2.2 Software Requirements

- Operating System: Windows 10 or above
- Programming Language: Python 3.7 or above
- Integrated Development Environment: Jupyter Notebook (bundled with Anaconda 3)
- Deep Learning Frameworks: PyTorch (1.8), TensorFlow (2.x)

2.3 Libraries Used

The following Python libraries are necessary to execute the project:

- Data Manipulation: `pandas`, `numpy`
- Visualisation: `seaborn`, `matplotlib`
- Machine Learning: `scikit-learn`

- Deep Learning: `torch`, `torch_geometric`, `tensorflow.keras`
- Miscellaneous: `warnings`, `scipy`

```python
#Import all needed libaries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch_geometric
import torch
from sklearn.svm import SVC
from sklearn.model_selection import KFold
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, SimpleRNN, Dense, Dropout
from torch_geometric.nn import GCNConv, GINConv
from torch_geometric.nn import global_mean_pool
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from torch_geometric.utils import from_scipy_sparse_matrix
from scipy.sparse import csr_matrix
from torch_geometric.data import Data
from sklearn.model_selection import cross_val_predict
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

Figure 1: List of Libraries used and Imported into Python Notebook

3 Data Collection

The dataset utilised is sourced from UCI ML Repository and includes samples for various IoT attacks such as Mirai and Gafgyt. These datasets are preprocessed and stored as CSV files:

- Files:
  - `5.gafgyt.combo.csv`
  - `5.gafgyt.junk.csv`
  - `5.gafgyt.scan.csv`
  - `5.gafgyt.tcp.csv`
  - `5.gafgyt.udp.csv`
  - `5.mirai.ack.csv`
  - `5.mirai.scan.csv`
  - `5.mirai.syn.csv`
  - `5.mirai.udp.csv`
  - `5.mirai.udpplain.csv`

Steps:

1. Load the datasets into pandas DataFrames.

```
# Load the CSV files from the specified location
benign = pd.read_csv(r'5.benign.csv')
gafgyt_combo = pd.read_csv(r'5.gafgyt.combo.csv')
gafgyt_junk = pd.read_csv(r'5.gafgyt.junk.csv')
gafgyt_scan = pd.read_csv(r'5.gafgyt.scan.csv')
gafgyt_tcp = pd.read_csv(r'5.gafgyt.tcp.csv')
gafgyt_udp = pd.read_csv(r'5.gafgyt.udp.csv')
mirai_ack = pd.read_csv(r'5.mirai.ack.csv')
mirai_scan = pd.read_csv(r'5.mirai.scan.csv')
mirai_syn = pd.read_csv(r'5.mirai.syn.csv')
mirai_udp = pd.read_csv(r'5.mirai.udp.csv')
mirai_udp_plain = pd.read_csv(r'5.mirai.udpplain.csv')
```

Figure 2: Reading the Dataset Files

2. Downsample datasets to address class imbalance.

```
# Downsample each dataset based on given fractions
benign_sampled = benign.sample(frac=0.25, replace=False)
gafgyt_combo_sampled = gafgyt_combo.sample(frac=0.25, replace=False)
gafgyt_junk_sampled = gafgyt_junk.sample(frac=0.5, replace=False)
gafgyt_scan_sampled = gafgyt_scan.sample(frac=0.5, replace=False)
gafgyt_tcp_sampled = gafgyt_tcp.sample(frac=0.15, replace=False)
gafgyt_udp_sampled = gafgyt_udp.sample(frac=0.15, replace=False)
mirai_ack_sampled = mirai_ack.sample(frac=0.25, replace=False)
mirai_scan_sampled = mirai_scan.sample(frac=0.15, replace=False)
mirai_syn_sampled = mirai_syn.sample(frac=0.25, replace=False)
mirai_udp_sampled = mirai_udp.sample(frac=0.1, replace=False)
mirai_udp_plain_sampled = mirai_udp_plain.sample(frac=0.27, replace=False)
```

Figure 3: Labelling the Data with respect to Attack Type

3. Merge and encode labels into a single DataFrame.

```
# Assign attack types to each dataset
benign_sampled['type'] = 'benign'
mirai_udp_sampled['type'] = 'mirai_udp'
gafgyt_combo_sampled['type'] = 'gafgyt_combo'
gafgyt_junk_sampled['type'] = 'gafgyt_junk'
gafgyt_scan_sampled['type'] = 'gafgyt_scan'
gafgyt_tcp_sampled['type'] = 'gafgyt_tcp'
gafgyt_udp_sampled['type'] = 'gafgyt_udp'
mirai_ack_sampled['type'] = 'mirai_ack'
mirai_scan_sampled['type'] = 'mirai_scan'
mirai_syn_sampled['type'] = 'mirai_syn'
mirai_udp_plain_sampled['type'] = 'mirai_udpplain'
```

Figure 4: Labelling the Data with respect to Attack Type

```python
# Combine the sampled data into a single DataFrame
df_N_BaIoT = pd.concat([benign_sampled, mirai_udp_sampled, gafgyt_combo_sampled,
                        gafgyt_junk_sampled, gafgyt_scan_sampled, gafgyt_tcp_sampled,
                        gafgyt_udp_sampled, mirai_ack_sampled, mirai_scan_sampled,
                        mirai_syn_sampled, mirai_udp_plain_sampled],
                       axis=0, sort=False, ignore_index=True)
```

Figure 5: Concatenating the Dataset

4 Data Visualisation

Distribution plots, bar plots, pair plots, and heatmaps are generated to analyse the dataset.

```python
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Plot distributions for a selection of features
features_to_plot = ['MI_dir_L5_weight', 'H_L5_mean', 'HH_L5_std', 'HpHp_L5_mean']
plt.figure(figsize=(12, 8))

for i, feature in enumerate(features_to_plot):
    plt.subplot(2, 2, i + 1)
    sns.histplot(df_N_BaIoT[feature], bins=30, kde=True)
    plt.title(f'Distribution of {feature}')

plt.tight_layout()
plt.show()
```

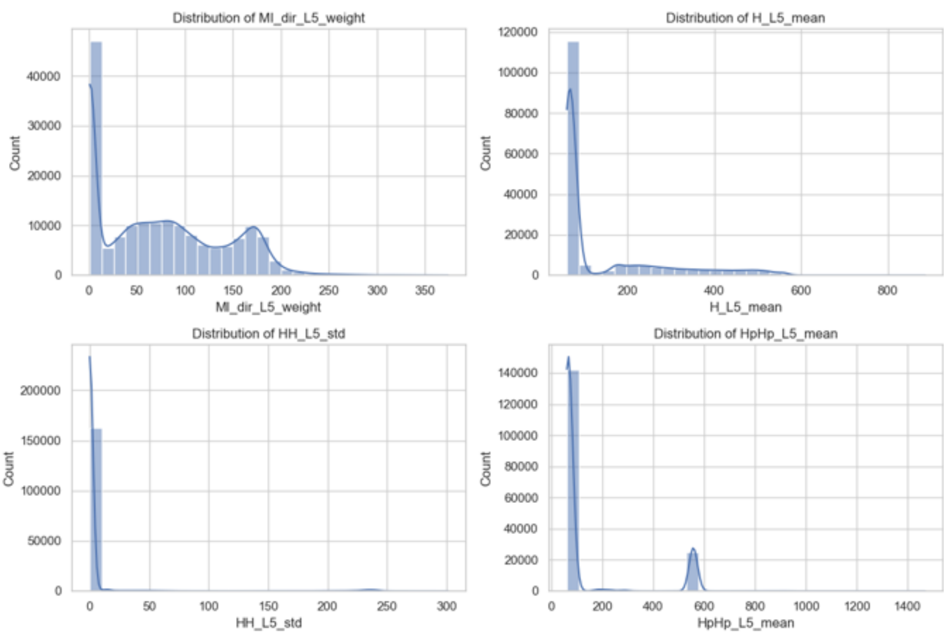

Figure 6: Plotting Histogram Plots for Four Different Features for Exploration

```
#Barplot for type and MI_dir_L5_weight
sns.barplot(x='type', y='MI_dir_L5_weight', data=df_N_BaIoT)
plt.title('Box Plot of MI_dir_L5_weight by Type')
plt.xticks(rotation=90)
plt.show()
```

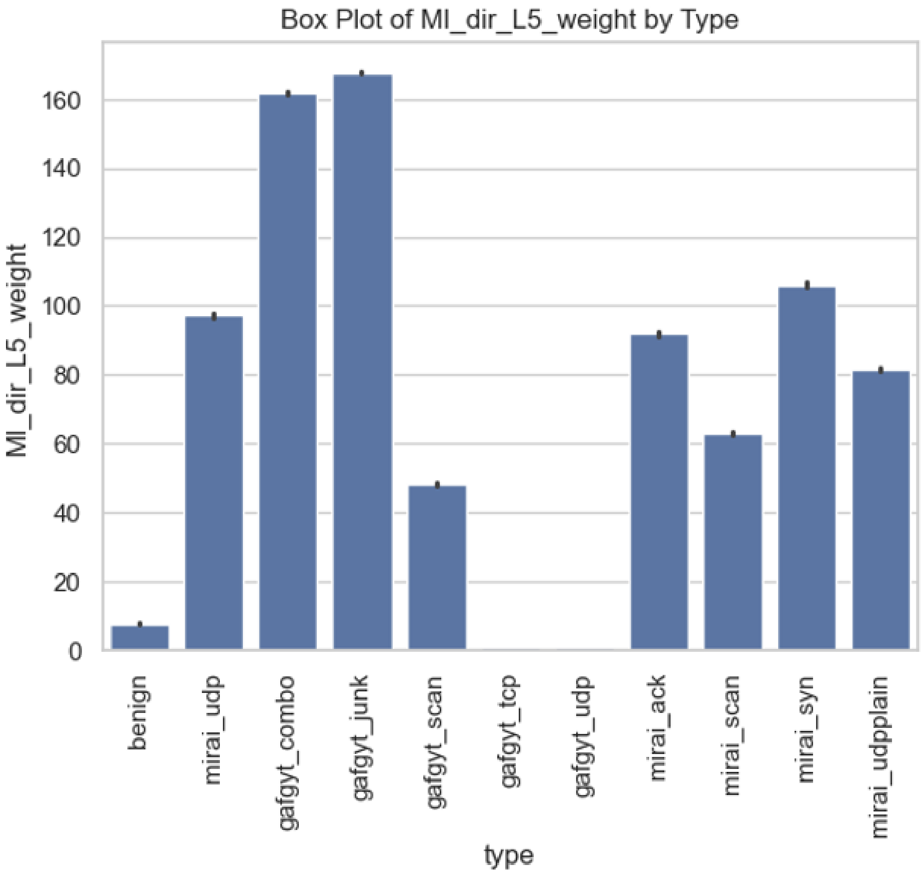

Figure 7: Boxplot for `MI_Dir_L5_weight`

```
#Pairplot
sns.pairplot(df_N_BaIoT, hue='type', vars=['MI_dir_L5_mean', 'H_L5_mean', 'HpHp_L5_mean'])
plt.show()
```

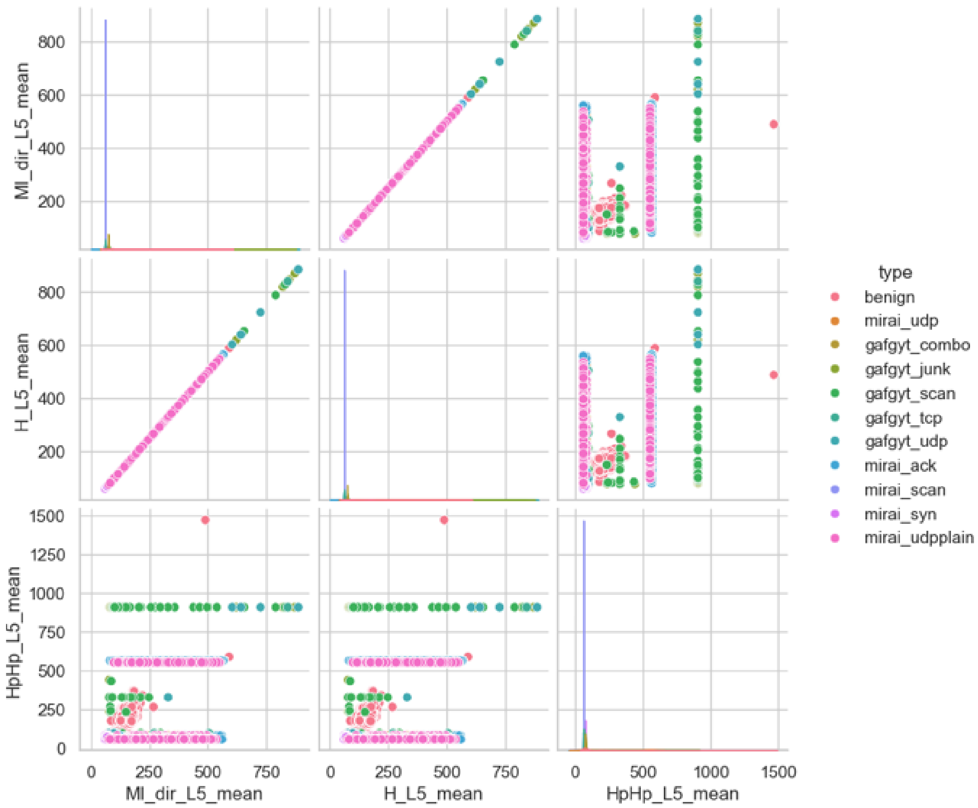

Figure 8: Pairplot for Some of the Features

```python
#Barplot
mean_values = df_N_BaIoT.groupby('type').mean().reset_index()
mean_values.plot(x='type', y=['MI_dir_L5_weight', 'H_L5_mean'], kind='bar', figsize=(12, 6))
plt.title('Mean Values of Selected Features by Type')
plt.ylabel('Mean Value')
plt.show()
```

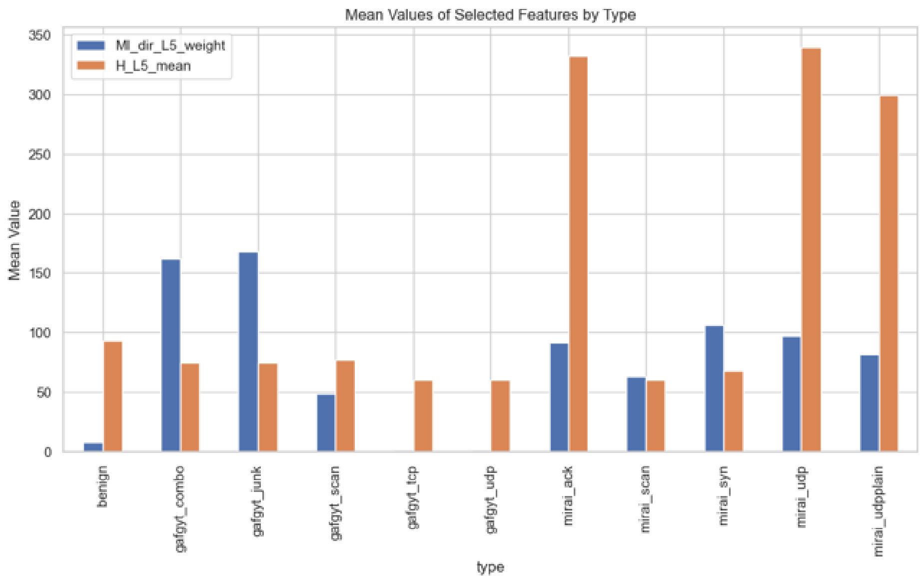

Figure 9: Barplot for `MI_dir_L5_weight` and `H_L5_mean` w.r.t. Attack Type

```python
# Count the occurrences of each type
type_counts = df_N_BaIoT['type'].value_counts()

# Create a pie chart
plt.pie(type_counts, labels=type_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Each Type')
plt.axis('equal')  # Equal aspect ratio ensures the pie chart is circular.
plt.show()
```

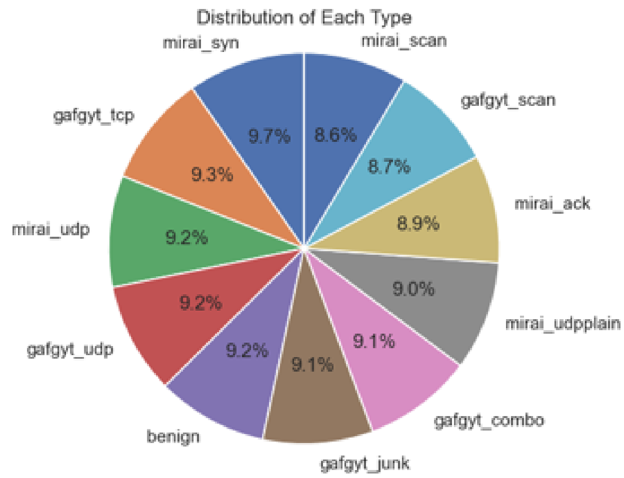

Figure 10: Pie Chart Showing the Distribution of Labels in the Dataset

5 Feature Engineering

Correlation matrix for all the features in the dataset are plotted for multicollinearity check. This is done by choosing subsets of 25 features at a time. Correlation matrix for the first 25 features is shown in Figure 11 below.

```python
#Correlation Heatmap for first 25 rows
plt.figure(figsize=(18, 15))
correlation_matrix_first_25 = df_N_BaIoT.iloc[:, :25].corr()
sns.heatmap(correlation_matrix_first_25, annot=True, fmt=".2f", cmap='Accent', square=True)
plt.title('Correlation Heatmap of First 25 Features')
plt.show()
```

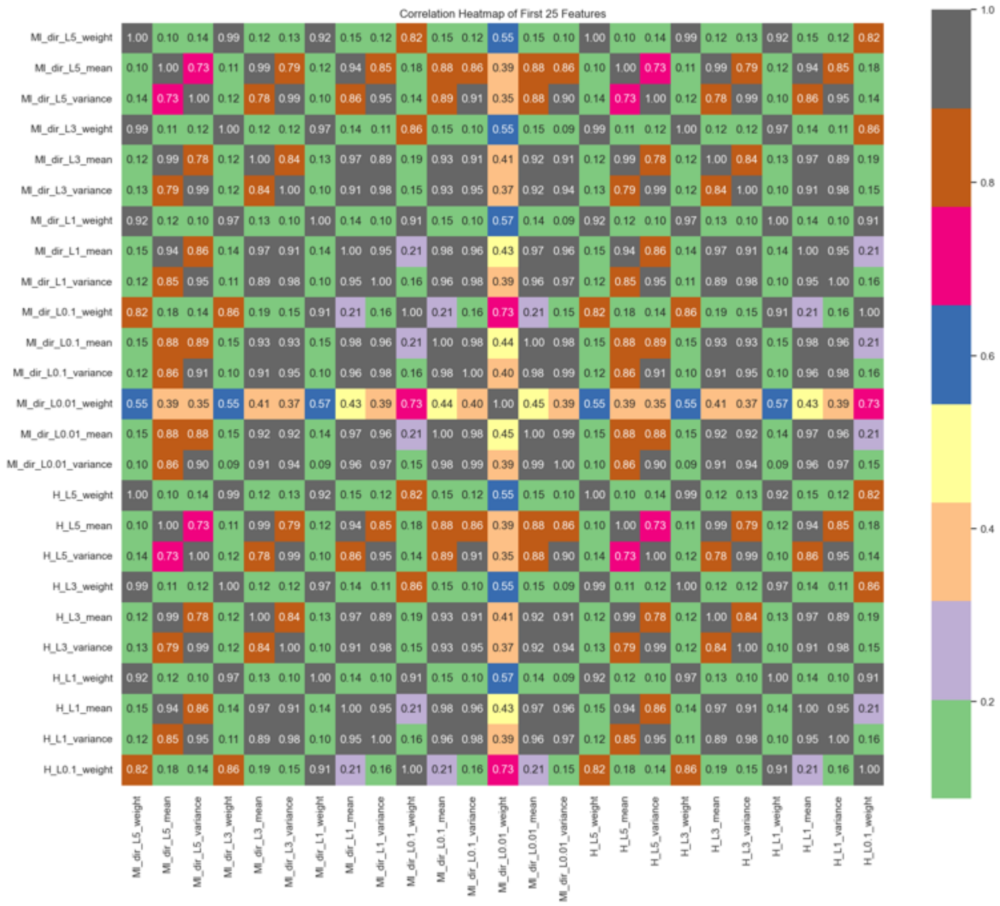

Figure 11: Correlation Matrix for First 25 Features

Next, the multicollinear features are dropped from the dataset. This is done by thresholding the Correlation Matrix by 0.8 value. Features with correlation coefficient above 0.8 are dropped. The code to implement this is shown in Figure 12.

```python
# numerical columns from the dataset
numerical_columns = df_N_BaIoT.select_dtypes(include=[np.number])

# Compute the correlation matrix for numerical columns
correlation_matrix = numerical_columns.corr()

# Identify highly correlated features
least_correlated_features = correlation_matrix[correlation_matrix < 0.8]

#  Print all highly correlated numerical columns
print("Least correlated numerical features (absolute correlation < 0.8):\n", least_correlated_features.columns.tolist())
```

Figure 12: Thresholding the Correlation Coefficients

As the `type` column representing the labels of the data is in categorical format, it needs to be converted to numerical form for modelling using Neural Network Architectures. Figure 13 below shows the implementation of `LabelEncoder()` from sklearn on the `type` column.

```python
# Label Encoder
label_encoder = LabelEncoder()
df_N_BaIoT['type'] = label_encoder.fit_transform(df_N_BaIoT['type'])
```

Figure 13: Label Encoding of the `Type` Column

Next, the dependent and independent variables are separated for modelling. The features are separated into the `X` dataframe whereas the dependent variable is stored in `y`. Its implementation is shown in Figure 14.

```python
#Sepearte dendent and independent variables
X = df_N_BaIoT[least_correlated_features.columns].values
y = df_N_BaIoT['type'].values
```

Figure 14: Separating the Dependent and Independent Columns

The dataset is then split into training and testing sets using the `train_test_split` function from sklearn's `model_selection` module.

```python
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figure 15: Dividing Data into Training and Testing Sets

The data is then normalized using the `StandardScaler()` from sklearn. It is first fit on the training data and then used to transform it.

```
# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 16: Normalizing the Data Using Standardization

PCA is then applied to get 10 components for feature reduction.

```
#PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

Figure 17: Application of PCA for Feature Reduction

```
subset_size = int(len(X_train_pca) * 0.7)
X_train_subset = X_train_pca[:subset_size]
y_train_subset = y_train[:subset_size]
```

Figure 18: Subsets of PCA Applied Data are Created for Validation Purposes

6 Modelling

A total of 10 models have been applied and tested on the dataset. These models, along with the libraries used, are listed below:

1. Gradient Boosting Machine (GBM) - Sklearn
2. KNN Model (KNN) - Sklearn
3. Gaussian Naïve Bayes (GNB) - Sklearn
4. Random Forest Model (RF) - Sklearn
5. Support Vector Machine Model (SVM) - Sklearn
6. LSTM Model - TensorFlow
7. RNN Model - TensorFlow
8. GRU Model - TensorFlow
9. Graph Convolution Network (GCN) - Torch Geometric
10. Graph Isomorphism Network (GIN) - Torch Geometric

The implementations of the different models with cross-validation are given below.

```python
#GBM
gbm_model = GradientBoostingClassifier()

# Perform 10-fold Cross-Validation on the subset
y_pred_gbm_cv = cross_val_predict(gbm_model, X_train_subset, y_train_subset, cv=10)
y_pred_proba_gbm_cv = cross_val_predict(gbm_model, X_train_subset, y_train_subset, cv=10, method='predict_proba')

# Calculate Cross-Validation Metrics
cv_accuracy_gbm = accuracy_score(y_train_subset, y_pred_gbm_cv)
cv_precision_gbm = precision_score(y_train_subset, y_pred_gbm_cv, average='weighted')
cv_recall_gbm = recall_score(y_train_subset, y_pred_gbm_cv, average='weighted')
cv_f1_gbm = f1_score(y_train_subset, y_pred_gbm_cv, average='weighted')
cv_roc_auc_gbm = roc_auc_score(y_train_subset, y_pred_proba_gbm_cv, multi_class='ovr')

# Print Cross-Validation Metrics
print("GBM Cross-Validation Performance on Subset:")
print("Cross-Validated Accuracy:", cv_accuracy_gbm)
print("Cross-Validated Precision:", cv_precision_gbm)
print("Cross-Validated Recall:", cv_recall_gbm)
print("Cross-Validated F1 Score:", cv_f1_gbm)
print("Cross-Validated ROC AUC:", cv_roc_auc_gbm)

# Classification Report with class names
print("\nClassification Report for Cross-Validated Predictions:\n",
      classification_report(y_train_subset, y_pred_gbm_cv, target_names=class_names))

# Confusion Matrix
conf_matrix_cv_gbm = confusion_matrix(y_train_subset, y_pred_gbm_cv)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", conf_matrix_cv_gbm)
```

Figure 19: Implementation of the GBM Model

```python
# KNN
knn_model = KNeighborsClassifier()

#Perform 10-fold Cross-Validation on the subset
y_pred_knn_cv = cross_val_predict(knn_model, X_train_subset, y_train_subset, cv=10)
y_pred_proba_knn_cv = cross_val_predict(knn_model, X_train_subset, y_train_subset, cv=10, method='predict_proba')

# Calculate Cross-Validation Metrics
cv_accuracy_knn = accuracy_score(y_train_subset, y_pred_knn_cv)
cv_precision_knn = precision_score(y_train_subset, y_pred_knn_cv, average='weighted')
cv_recall_knn = recall_score(y_train_subset, y_pred_knn_cv, average='weighted')
cv_f1_knn = f1_score(y_train_subset, y_pred_knn_cv, average='weighted')
cv_roc_auc_knn = roc_auc_score(y_train_subset, y_pred_proba_knn_cv, multi_class='ovr')

# Print Cross-Validation Metrics
print("KNN Cross-Validation Performance on Subset:")
print("Cross-Validated Accuracy:", cv_accuracy_knn)
print("Cross-Validated Precision:", cv_precision_knn)
print("Cross-Validated Recall:", cv_recall_knn)
print("Cross-Validated F1 Score:", cv_f1_knn)
print("Cross-Validated ROC AUC:", cv_roc_auc_knn)

#  Classification Report
print("\nClassification Report for Cross-Validated Predictions:\n", classification_report(y_train_subset, y_pred_knn_cv))

# Confusion Matrix
conf_matrix_cv_knn = confusion_matrix(y_train_subset, y_pred_knn_cv)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", conf_matrix_cv_knn)
```

Figure 20: Implementation of the KNN Model

```python
# Naive Bayes
gnb_model = GaussianNB()

# Perform 10-fold Cross-Validation on the subset
y_pred_gnb_cv = cross_val_predict(gnb_model, X_train_subset, y_train_subset, cv=10)
y_pred_proba_gnb_cv = cross_val_predict(gnb_model, X_train_subset, y_train_subset, cv=10, method='predict_proba')

#  Calculate Cross-Validation Metrics
cv_accuracy_gnb = accuracy_score(y_train_subset, y_pred_gnb_cv)
cv_precision_gnb = precision_score(y_train_subset, y_pred_gnb_cv, average='weighted')
cv_recall_gnb = recall_score(y_train_subset, y_pred_gnb_cv, average='weighted')
cv_f1_gnb = f1_score(y_train_subset, y_pred_gnb_cv, average='weighted')
cv_roc_auc_gnb = roc_auc_score(y_train_subset, y_pred_proba_gnb_cv, multi_class='ovr')

# Print Cross-Validation Metrics
print("GNB Cross-Validation Performance on Subset:")
print("Cross-Validated Accuracy:", cv_accuracy_gnb)
print("Cross-Validated Precision:", cv_precision_gnb)
print("Cross-Validated Recall:", cv_recall_gnb)
print("Cross-Validated F1 Score:", cv_f1_gnb)
print("Cross-Validated ROC AUC:", cv_roc_auc_gnb)

#  Classification Report
print("\nClassification Report for Cross-Validated Predictions:\n", classification_report(y_train_subset, y_pred_gnb_cv,target_names=class_names))

# Confusion Matrix
conf_matrix_cv_gnb = confusion_matrix(y_train_subset, y_pred_gnb_cv)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", conf_matrix_cv_gnb)
```

Figure 21: Implementation of the GNB Model

```python
# Random Forest
rf_model = RandomForestClassifier()

#  Perform 10-fold Cross-Validation on the subset
y_pred_rf_cv = cross_val_predict(rf_model, X_train_subset, y_train_subset, cv=10)
y_pred_proba_rf_cv = cross_val_predict(rf_model, X_train_subset, y_train_subset, cv=10, method='predict_proba')

# Calculate Cross-Validation Metrics
cv_accuracy_rf = accuracy_score(y_train_subset, y_pred_rf_cv)
cv_precision_rf = precision_score(y_train_subset, y_pred_rf_cv, average='weighted')
cv_recall_rf = recall_score(y_train_subset, y_pred_rf_cv, average='weighted')
cv_f1_rf = f1_score(y_train_subset, y_pred_rf_cv, average='weighted')
cv_roc_auc_rf = roc_auc_score(y_train_subset, y_pred_proba_rf_cv, multi_class='ovr')

# Print Cross-Validation Metrics
print("Random Forest Cross-Validation Performance on Subset:")
print("Cross-Validated Accuracy:", cv_accuracy_rf)
print("Cross-Validated Precision:", cv_precision_rf)
print("Cross-Validated Recall:", cv_recall_rf)
print("Cross-Validated F1 Score:", cv_f1_rf)
print("Cross-Validated ROC AUC:", cv_roc_auc_rf)

#  Classification Report
print("\nClassification Report for Cross-Validated Predictions:\n", classification_report(y_train_subset, y_pred_rf_cv,target_names=class_names))

#  Confusion Matrix
conf_matrix_cv_rf = confusion_matrix(y_train_subset, y_pred_rf_cv)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", conf_matrix_cv_rf)
```

Figure 22: Implementation of the RF Model

```python
# SVM
svm_model = SVC(probability=True)

# Perform 10-fold Cross-Validation on the subset
y_pred_svm_cv = cross_val_predict(svm_model, X_train_subset, y_train_subset, cv=10)
y_pred_proba_svm_cv = cross_val_predict(svm_model, X_train_subset, y_train_subset, cv=10, method='predict_proba')

#  Calculate Cross-Validation Metrics
cv_accuracy_svm = accuracy_score(y_train_subset, y_pred_svm_cv)
cv_precision_svm = precision_score(y_train_subset, y_pred_svm_cv, average='weighted')
cv_recall_svm = recall_score(y_train_subset, y_pred_svm_cv, average='weighted')
cv_f1_svm = f1_score(y_train_subset, y_pred_svm_cv, average='weighted')
cv_roc_auc_svm = roc_auc_score(y_train_subset, y_pred_proba_svm_cv, multi_class='ovr')

# Print Cross-Validation Metrics
print("SVM :")
print("Cross-Validated Accuracy:", cv_accuracy_svm)
print("Cross-Validated Precision:", cv_precision_svm)
print("Cross-Validated Recall:", cv_recall_svm)
print("Cross-Validated F1 Score:", cv_f1_svm)
print("Cross-Validated ROC AUC:", cv_roc_auc_svm)

#  Classification Report
print("\nClassification Report:\n", classification_report(y_train_subset, y_pred_svm_cv,target_names=class_names))

#  Confusion Matrix
conf_matrix_cv_svm = confusion_matrix(y_train_subset, y_pred_svm_cv)
print("\nConfusion Matrix :\n", conf_matrix_cv_svm)
```

Figure 23: Implementation of the SVM Model

```python
from sklearn.model_selection import KFold
X_train_subset = X_train_subset.reshape((X_train_subset.shape[0], 1, X_train_subset.shape[1]))

#  LSTM model function
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(64, return_sequences=False, input_shape=input_shape))
    model.add(Dense(11, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

#  10-fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Lists to store metrics for each fold
all_accuracy = []
all_precision = []
all_recall = []
all_f1 = []
all_confusion_matrices = np.zeros((11, 11))

# Accumulate true and predicted labels
true_labels = []
pred_labels = []
```

```python
for fold, (train_index, val_index) in enumerate(kf.split(X_train_subset, y_train_subset), 1):
    print(f"\n--- Fold {fold} ---")

    #  Split the data into training and validation sets for the current fold
    X_train_fold, X_val_fold = X_train_subset[train_index], X_train_subset[val_index]
    y_train_fold, y_val_fold = y_train_subset[train_index], y_train_subset[val_index]

    #  Define and compile the LSTM model
    lstm_model = create_lstm_model(input_shape=(X_train_fold.shape[1], X_train_fold.shape[2]))

    #  Train the model for 100 epochs
    lstm_model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=32, verbose=0)

    # Predict on validation data
    y_pred_val = lstm_model.predict(X_val_fold)
    y_pred_classes = np.argmax(y_pred_val, axis=1)

    # Evaluate the model for this fold
    accuracy = accuracy_score(y_val_fold, y_pred_classes)
    precision = precision_score(y_val_fold, y_pred_classes, average='weighted')
    recall = recall_score(y_val_fold, y_pred_classes, average='weighted')
    f1 = f1_score(y_val_fold, y_pred_classes, average='weighted')

    #  confusion matrix
    conf_matrix = confusion_matrix(y_val_fold, y_pred_classes, labels=np.arange(11))
    all_confusion_matrices += conf_matrix  # Accumulate confusion matrices across folds

    # Collect true and predicted labels for the overall classification report
    true_labels.extend(y_val_fold)
    pred_labels.extend(y_pred_classes)

    # Store metrics
    all_accuracy.append(accuracy)
    all_precision.append(precision)
    all_recall.append(recall)
    all_f1.append(f1)

#  average metrics across all folds
average_accuracy = np.mean(all_accuracy)
average_precision = np.mean(all_precision)
average_recall = np.mean(all_recall)
average_f1 = np.mean(all_f1)
```

```python
#  Generate overall classification report and confusion matrix
overall_classification_report = classification_report(true_labels, pred_labels,target_names=class_names)
average_confusion_matrix = all_confusion_matrices / 10  # Average confusion matrix across folds

# Print the average metrics, classification report, and confusion matrix
print("\n--- Average Metrics across all folds ---")
print(f"Average Accuracy: {average_accuracy:.4f}")
print(f"Average Precision: {average_precision:.4f}")
print(f"Average Recall: {average_recall:.4f}")
print(f"Average F1 Score: {average_f1:.4f}")

# Print overall classification report
print("\n--- Overall Classification Report ---")
print(overall_classification_report)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", average_confusion_matrix)
```

Figure 24: Implementation and Evaluation of the LSTM Model

```python
# RNN model function
def create_rnn_model(input_shape):
    model = Sequential()
    model.add(SimpleRNN(64, return_sequences=False, input_shape=input_shape))
    model.add(Dense(11, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

#  10-fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Lists to store metrics for each fold
all_accuracy = []
all_precision = []
all_recall = []
all_f1 = []
all_confusion_matrices = np.zeros((11, 11))

# Accumulate true and predicted labels
true_labels = []
pred_labels = []
```

```python
for fold, (train_index, val_index) in enumerate(kf.split(X_train_subset, y_train_subset), 1):
    print(f"\n--- Fold {fold} ---")

    # Split the data into training and validation sets
    X_train_fold, X_val_fold = X_train_subset[train_index], X_train_subset[val_index]
    y_train_fold, y_val_fold = y_train_subset[train_index], y_train_subset[val_index]

    # Define and compile the RNN model
    rnn_model = create_rnn_model(input_shape=(X_train_fold.shape[1], X_train_fold.shape[2]))

    # Train the model for 100 epochs
    rnn_model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=32, verbose=0)

    # Predict on validation data
    y_pred_val = rnn_model.predict(X_val_fold)
    y_pred_classes = np.argmax(y_pred_val, axis=1)

    # Evaluate the model for this fold
    accuracy = accuracy_score(y_val_fold, y_pred_classes)
    precision = precision_score(y_val_fold, y_pred_classes, average='weighted')
    recall = recall_score(y_val_fold, y_pred_classes, average='weighted')
    f1 = f1_score(y_val_fold, y_pred_classes, average='weighted')

    # Generate confusion matrix for this fold and accumulate
    conf_matrix = confusion_matrix(y_val_fold, y_pred_classes, labels=np.arange(11))
    all_confusion_matrices += conf_matrix

    # Collect true and predicted labels for the overall classification report
    true_labels.extend(y_val_fold)
    pred_labels.extend(y_pred_classes)

    # Store metrics
    all_accuracy.append(accuracy)
    all_precision.append(precision)
    all_recall.append(recall)
    all_f1.append(f1)

#  Calculate average metrics across all folds
average_accuracy_rnn = np.mean(all_accuracy)
average_precision_rnn = np.mean(all_precision)
average_recall_rnn = np.mean(all_recall)
average_f1_rnn = np.mean(all_f1)
```

```python
#  Generate overall classification report and confusion matrix
overall_classification_report_rnn = classification_report(true_labels, pred_labels,target_names=class_names)
average_confusion_matrix_rnn = all_confusion_matrices / 10

# Print the average metrics, classification report, and confusion matrix
print("\n--- Average Metrics across all folds ---")
print(f"Average Accuracy: {average_accuracy_rnn:.4f}")
print(f"Average Precision: {average_precision_rnn:.4f}")
print(f"Average Recall: {average_recall_rnn:.4f}")
print(f"Average F1 Score: {average_f1_rnn:.4f}")

# Print overall classification report
print("\n--- Overall Classification Report ---")
print(overall_classification_report_rnn)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", average_confusion_matrix_rnn)
```

Figure 25: Implementation and Evaluation of the RNN Model

```python
#  GRU model
def create_gru_model(input_shape):
    model = Sequential()
    model.add(GRU(64, return_sequences=False, input_shape=input_shape))
    model.add(Dense(11, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# 10-fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Lists to store metrics for each fold
all_accuracy = []
all_precision = []
all_recall = []
all_f1 = []
all_confusion_matrices = np.zeros((11, 11))

# Accumulate true and predicted labels
true_labels = []
pred_labels = []
```

```python
for fold, (train_index, val_index) in enumerate(kf.split(X_train_subset, y_train_subset), 1):
    print(f"\n--- Fold {fold} ---")

    # Split the data into training and validation sets for the current fold
    X_train_fold, X_val_fold = X_train_subset[train_index], X_train_subset[val_index]
    y_train_fold, y_val_fold = y_train_subset[train_index], y_train_subset[val_index]

    # Define and compile the GRU model
    gru_model = create_gru_model(input_shape=(X_train_fold.shape[1], X_train_fold.shape[2]))

    #  Train the model for 100 epochs
    gru_model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=32, verbose=0)

    #  Predict on validation data
    y_pred_val = gru_model.predict(X_val_fold)
    y_pred_classes = np.argmax(y_pred_val, axis=1)
```

```python
    # Evaluate the model for this fold
    accuracy = accuracy_score(y_val_fold, y_pred_classes)
    precision = precision_score(y_val_fold, y_pred_classes, average='weighted')
    recall = recall_score(y_val_fold, y_pred_classes, average='weighted')
    f1 = f1_score(y_val_fold, y_pred_classes, average='weighted')

    # Generate confusion matrix for this fold and accumulate
    conf_matrix = confusion_matrix(y_val_fold, y_pred_classes, labels=np.arange(11))
    all_confusion_matrices += conf_matrix

    # Collect true and predicted labels
    true_labels.extend(y_val_fold)
    pred_labels.extend(y_pred_classes)

    # Store metrics
    all_accuracy.append(accuracy)
    all_precision.append(precision)
    all_recall.append(recall)
    all_f1.append(f1)
```

```python
# Calculate average metrics across all folds
average_accuracy_gru = np.mean(all_accuracy)
average_precision_gru = np.mean(all_precision)
average_recall_gru = np.mean(all_recall)
average_f1_gru = np.mean(all_f1)

# Generate overall classification report and confusion matrix
overall_classification_report_gru = classification_report(true_labels, pred_labels,target_names=class_names)
average_confusion_matrix_gru = all_confusion_matrices / 10

# Print the average metrics
print("\n--- Average Metrics across all folds ---")
print(f"Average Accuracy: {average_accuracy_gru:.4f}")
print(f"Average Precision: {average_precision_gru:.4f}")
print(f"Average Recall: {average_recall_gru:.4f}")
print(f"Average F1 Score: {average_f1_gru:.4f}")

# Print overall classification report
print("\n--- Overall Classification Report ---")
print(overall_classification_report_gru)
print("\nConfusion Matrix for Cross-Validated Predictions:\n", average_confusion_matrix_gru)
```

Figure 26: Implementation and Evaluation of the GRU Model

```python
# Define a threshold
threshold = 0.5
```

```python
edges = np.argwhere(np.abs(correlation_matrix) > threshold)
edges = edges[edges[:, 0] != edges[:, 1]]
edge_index = torch.tensor(edges.T, dtype=torch.long)
```

Figure 27: Generating Graph Data from the PCA Components

```python
# Initialize the model
class GCN(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_features, 16)
        self.conv2 = GCNConv(16, num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

```python
# Prepare your data tensors
x_tensor = torch.tensor(X_train_pca, dtype=torch.float)
y_tensor = torch.tensor(y_train, dtype=torch.long)

# Cross-validation setup
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 5-fold cross-validation
accuracies1, precisions1, recalls1, f1_scores1, roc_aucs1 = [], [], [], [], []
all_preds = []  # To store all predictions for confusion matrix
all_true_labels = []  # To store all true labels for confusion matrix
```

```python
# Cross-validation loop
for train_idx, val_idx in kf.split(x_tensor):
    # Split the data into training and validation
    x_train_cv, x_val_cv = x_tensor[train_idx], x_tensor[val_idx]
    y_train_cv, y_val_cv = y_tensor[train_idx], y_tensor[val_idx]

    # Create data objects for training and validation
    train_data = Data(x=x_train_cv, edge_index=edge_index)
    val_data = Data(x=x_val_cv, edge_index=edge_index)

    # Initialize the model and optimizer
    model = GCN(num_features=X_train_pca.shape[1], num_classes=len(set(y_train)))
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    criterion = torch.nn.NLLLoss()

    # Training the model
    model.train()
    for epoch in range(200):
        optimizer.zero_grad()
        output = model(train_data)
        loss = criterion(output, y_train_cv)
        loss.backward()
        optimizer.step()

    # Validation the model
    model.eval()
    with torch.no_grad():
        val_output = model(val_data)

    # Get probabilities for ROC AUC
    val_probs = F.softmax(val_output, dim=1)
    val_preds = val_output.argmax(dim=1).numpy()

    # Evaluate the model
    accuracy = accuracy_score(y_val_cv, val_preds)
    precision = precision_score(y_val_cv, val_preds, average='weighted')
    recall = recall_score(y_val_cv, val_preds, average='weighted')
    f1 = f1_score(y_val_cv, val_preds, average='weighted')
```

```python
    # Calculate ROC AUC
    roc_auc = roc_auc_score(y_val_cv, val_probs.numpy(), multi_class='ovr', average='weighted')

    # Append results for cross-validation
    accuracies1.append(accuracy)
    precisions1.append(precision)
    recalls1.append(recall)
    f1_scores1.append(f1)
    roc_aucs1.append(roc_auc)

    # Store predictions and true labels for confusion matrix
    all_preds.extend(val_preds)
    all_true_labels.extend(y_val_cv.numpy())

# Calculate the average performance across all folds
print(f"Average Accuracy: {np.mean(accuracies1):.4f}")
print(f"Average Precision: {np.mean(precisions1):.4f}")
print(f"Average Recall: {np.mean(recalls1):.4f}")
print(f"Average F1 Score: {np.mean(f1_scores1):.4f}")
print(f"Average ROC-AUC: {np.mean(roc_aucs1):.4f}")

# Classification report and confusion matrix for all folds
print("\nClassification Report:\n", classification_report(all_true_labels, all_preds))

# Confusion Matrix
cm = confusion_matrix(all_true_labels, all_preds)
print(cm)
```

Figure 28: Implementation and Evaluation of the GCN Model

```python
# GIN model class
class GIN(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super(GIN, self).__init__()
        self.conv1 = GINConv(torch.nn.Sequential(
            torch.nn.Linear(num_features, 16),
            torch.nn.ReLU(),
            torch.nn.Linear(16, 16)
        ))
        self.conv2 = GINConv(torch.nn.Sequential(
            torch.nn.Linear(16, 16),
            torch.nn.ReLU(),
            torch.nn.Linear(16, num_classes)
        ))

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

# Prepare your data tensors
x_tensor = torch.tensor(X_train_pca, dtype=torch.float)
y_tensor = torch.tensor(y_train, dtype=torch.long)

# Cross-validation setup
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 5-fold cross-validation
accuracies, precisions, recalls, f1_scores, roc_aucs = [], [], [], [], []
all_preds = []
all_true_labels = []
```

```python
# Cross-validation loop
for train_idx, val_idx in kf.split(x_tensor):
    # Split the data into training and validation
    x_train_cv, x_val_cv = x_tensor[train_idx], x_tensor[val_idx]
    y_train_cv, y_val_cv = y_tensor[train_idx], y_tensor[val_idx]

    # Create data objects for training and validation
    train_data = Data(x=x_train_cv, edge_index=edge_index)
    val_data = Data(x=x_val_cv, edge_index=edge_index)

    # Initialize the model and optimizer
    model = GIN(num_features=X_train_pca.shape[1], num_classes=len(set(y_train)))
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    criterion = torch.nn.NLLLoss()

    # Training the model
    model.train()
    for epoch in range(200):
        optimizer.zero_grad()
        output = model(train_data)
        loss = criterion(output, y_train_cv)
        loss.backward()
        optimizer.step()

    # Validation the model
    model.eval()
    with torch.no_grad():
        val_output = model(val_data)

    # Get probabilities for ROC AUC
    val_probs = F.softmax(val_output, dim=1)
    val_preds = val_output.argmax(dim=1).numpy()

    # Evaluate the model
    accuracy = accuracy_score(y_val_cv, val_preds)
    precision = precision_score(y_val_cv, val_preds, average='weighted')
    recall = recall_score(y_val_cv, val_preds, average='weighted')
    f1 = f1_score(y_val_cv, val_preds, average='weighted')

    # Calculate ROC AUC
    roc_auc = roc_auc_score(y_val_cv, val_probs.numpy(), multi_class='ovr', average='weighted')

    # Append results for cross-validation
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)
```

```python
    # Store predictions and true labels for confusion matrix
    all_preds.extend(val_preds)
    all_true_labels.extend(y_val_cv.numpy())

# Calculate the average performance across all folds
print(f"Average Accuracy: {np.mean(accuracies):.4f}")
print(f"Average Precision: {np.mean(precisions):.4f}")
print(f"Average Recall: {np.mean(recalls):.4f}")
print(f"Average F1 Score: {np.mean(f1_scores):.4f}")
print(f"Average ROC-AUC: {np.mean(roc_aucs):.4f}")

# Classification report and confusion matrix for all folds
print("\nClassification Report:\n", classification_report(all_true_labels, all_preds))

# Confusion Matrix
cm = confusion_matrix(all_true_labels, all_preds)
print(cm)
```

Figure 29: Implementation and Evaluation of the GIN Model

```
model_results = []

# Append model metrics
model_results.append({'Model': 'GCN', 'Accuracy': np.mean(accuracies1), 'Precision': np.mean(precisions1), 'Recall': np.mean(recalls1), 'F1 Score': np.mean(f1_scores1)})
model_results.append({'Model': 'GIN', 'Accuracy': np.mean(accuracies), 'Precision': np.mean(precision), 'Recall': np.mean(recall), 'F1 Score': np.mean(f1)})
model_results.append({'Model': 'Naive Bayes', 'Accuracy': cv_accuracy_gnb, 'Precision': cv_precision_gnb, 'Recall': cv_recall_gnb, 'F1 Score': cv_f1_gnb})
model_results.append({'Model': 'k-NN', 'Accuracy': cv_accuracy_knn, 'Precision': cv_precision_knn, 'Recall': cv_recall_knn, 'F1 Score': cv_f1_knn})
model_results.append({'Model': 'GBM', 'Accuracy': cv_accuracy_gbm, 'Precision': cv_precision_gbm, 'Recall': cv_recall_gbm, 'F1 Score': cv_f1_gbm})
model_results.append({'Model': 'Random Forest', 'Accuracy': cv_accuracy_rf, 'Precision': cv_precision_rf, 'Recall': cv_recall_rf, 'F1 Score': cv_f1_rf})
model_results.append({'Model': 'Support Vector Machine', 'Accuracy': cv_accuracy_svm, 'Precision': cv_precision_svm, 'Recall': cv_recall_svm, 'F1 Score': cv_f1_svm})
model_results.append({'Model': 'LSTM', 'Accuracy': average_accuracy, 'Precision': average_precision, 'Recall': average_recall, 'F1 Score': average_f1})
model_results.append({'Model': 'RNN', 'Accuracy': average_accuracy_rnn, 'Precision': average_precision_rnn, 'Recall': average_recall_rnn, 'F1 Score': average_f1_rnn})
model_results.append({'Model': 'GRU', 'Accuracy': average_accuracy_gru, 'Precision': average_precision_gru, 'Recall': average_recall_gru, 'F1 Score': average_f1_gru})

# Convert to DataFrame
results_df = pd.DataFrame(model_results)
print(results_df)
```

Figure 30: Combining the Model Results

```
# Plotting
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
results_df.set_index('Model', inplace=True)

# Plot each metric
results_df[metrics].plot(kind='bar', figsize=(10, 6))
plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

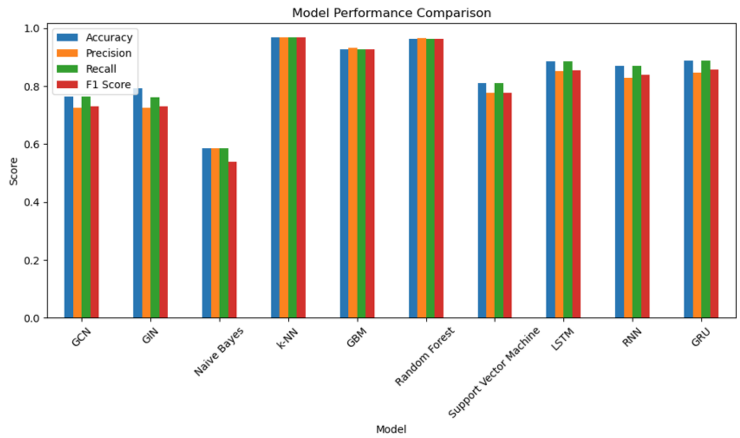

Figure 31: Plotting the Results Graphically

Similar to the above implementation, the models are used without cross-validation. The implementation of the models is exactly the same.

7 Execution Steps

1. **Setup Environment:**
   - Install required libraries using `pip install` or Anaconda.
   - Launch Jupyter Notebook.
2. **Run Code:**
   - Execute the provided `.ipynb` file in sequence.
   - Configure file paths for datasets.
3. **View Results:**
   - Outputs and visualisations will be generated within the notebook.

8 References

- Guide (no date). https://www.tensorflow.org/guide.
- PyG Documentation — pytorch_geometric documentation (no date). https://pytorch-geometric.readthedocs.io/en/latest/.
- scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation (no date). https://scikit-learn.org/.
- UCI Machine Learning Repository (no date). https://archive.ics.uci.edu/dataset/442/detection+of+iot+botnet+attacks+n+baiot.