SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Mitigation of attacks via improved network security in IoT network environment using RNN

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

NAME:Priyanka B
(Reg. No:125003239, B.Tech Computer Science & Engineering)
NAME :Akshayaa S
(Reg. No:125156009, B.Tech Computer Science & Engineering Artificial Intelligence and Data Science)
NAME :Harini R
(Reg. No.: 125158017, B.Tech Computer Science & Engineering IoT and Automation)

May 2024

Table of Contents

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

BONAFIDE CERTIFICATE

This is to certify that the report titled Mitigation of attacks via improved network security in IoT network environment using RNN submitted as a requirement for the course, CSE300: MINI PROJECT for B.Tech. is a bonafide record of the work done by **Ms. Priyanka B (125003239, B.Tech Computer Science & Engineering), Ms Akshayaa S (125156009, B.Tech Computer Science & Engineering Artificial Intelligence and Data Science) Ms. Harini R (125158017, B.Tech Computer Science & Engineering IoT and Automation)** during the academic year 2023-24, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation :

Date :

Mini Project Viva voce held on ________________

Examiner 1 Examiner 2

ACKNOWLEDGEMENT

We would like to thank our **Honorable Chancellor Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our **Honorable Vice-Chancellor Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan, Dean, Planning & Development,** for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli, Registrar**, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram, Dean, School of Computing, Dr. R. Muthaiah, Associate Dean, Research, Dr. K.Ramkumar, Associate Dean, Academics, Dr. D. Manivannan, Associate Dean, Infrastructure, Dr. R. Algeswaran, Associate Dean, Students Welfare.**

Our guide **Dr. Meenalochani M, Asst. Professor-III,** School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work.

We also thank the **project review panel** members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through project.

List of Figures

List of tables

ABBREVIATIONS

DDoS	Distributed Denial of Service
IoT	Internet of Things
RNN	Recurrent Neural Network
XBoost	Extreme Gradient Boosting
KDD	Knowledge Discovery and Data Mining
FLS	Fuzzy Logic Systems
CEPIDS	Clustering Enhanced Pre-processed Intrusion Detection Systems
DeepDefense	Deep Learning-based DDoS Attack Detection Approach
BLSTM	Bidirectional Long Short-Term Memory
CICIDS	Canadian Institute for Cybersecurity Detection Standard
IDS	Intrusion Detection System
SDN	Software-Defined Networking
CIDD	Cyber Range Intrusion Detection Dataset
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
ML	Machine Learning
DL	Deep Learning

NOTATIONS

accuracy_score	Sklearn library, a function for calculating the accuracy of a classifier
f1_score	Sklearn library, a function for calculating the F1-score of a classifier
LabelEncoder	Sklearn library, a class for encoding categorical labels
min_max_scaler	Sklearn library, a class for scaling features to a given range
np	NumPy library, a Python library for numerical computing
pd	Pandas library, a Python library for data manipulation and analysis
precision_score	Sklearn library, a function for calculating the precision of a classifier
recall_score	Sklearn library, a function for calculating the recall of a classifier
train_test_split	Sklearn library, a function for splitting data into training and testing sets
XGBoostClassifier	XGBoost library, a class for training gradient boosting trees

ABSTRACT

There is a growing threat to vulnerable IoT networks through Brute-force attacks, demanding security defenses against the DoS and DDoS attacks. The growing threat requires security by exploring the potential of machine learning techniques to address these challenges in IoT environments.These attacks can violate the CIA(confidentiality, integrity and availability) of the data. To eliminate the threats, techniques such as Recurrent Neural Networks(RNNs), Fuzzy Logic Systems (FLS) and DeepDefence , are proposed to handle time series data, to identify anomalous patterns and mitigate the attack potentially . The RNN technique gives a confident solution framework involving Machine Learning, Deep Learning, and data pre-processing techniques to strengthen the cybersecurity in IoT networks. .The data collected is divided into two as normal and attacks to actively recognize and respond to the threats.These methods allow enhanced monitoring of network traffic and identification of anomalous patterns indicative of potential attacks. By doing this we This approach has major significance in mitigating malicious activities that include DoS, DDoS attacks safeguarding the integrity, confidentiality, accessibility and reliability of the interconnected devices in the ever-evolving IoT landscape**.**

keywords: IoT, DoS, DDoS, Recurrent Neural Networks (RNNs), Fuzzy Logic Systems (FLS), DeepDefence, machine learning, deep learning, cybersecurity, time series data, network traffic, anomalous patterns, data pre-processing techniques, confidentiality, integrity, accessibility, reliability.

CHAPTER 1

SUMMARY OF THE BASE PAPER

Paper details:

Title: Mitigation of attacks via improved network security in IOT network environment using RNN

Journal: Measurement: Sensors

Publisher : Elsevier Ltd.

Year : 2024

Indexed in : CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Content and Novelty

The base paper proposes an IoT network-based threat mitigation strategy using RNN algorithm. The RNN classifies the preprocessed and feature extracted data into attack related attributes. After min-max scaling, the XGBoost model is used for feature selection. The metrics like accuracy, precision, recall, and f-score are assessed after the simulations are executed over the datasets. High accuracy in both the accuracy and the precision is achieved using the proposed RNN based schema. The proposed approach is novel in its use of machine learning and deep learning techniques to analyze time series data in IoT networks, which is particularly relevant in IoT networks where data is continuously generated and transmitted.

Research Addressed and Solution Proposed

The research paper addresses the problem of DoS and DDoS attacks in the IoT networks. Due to the increasing amount of devices connected over the internet and poor security measures, the DoS and DDoS attacks have become a common thing. The machine learning and the deep learning techniques like RNN, FLS and DeepDefence techniques are used by the authors to address the time series data and strengthen the cyber security in the IoT devices.The RNN technique gives a confident solution for strengthening cyber security in IoT devices. To analyze the IoT networks the machine learning and deep learning techniques have been very novel. The proposed method allows for enhanced monitoring of network traffic and identification of anomalous patterns for the mitigation of the attacks. To meet the growing attacks of DoS and DDoS, this paper gives a novel solution to identify and mitigate them.

Architecture and Algorithm Proposed and Its Correctness

The architecture proposed in the paper is an RNN-based threat mitigation strategy for IoT networks. The correctness of the proposed algorithm is evaluated using the KDD99 dataset,and CICIDS17 dataset, which is well known for intuition detection the results make sure that the proposed algorithms achieve the high accuracy, precision, recall, and F1-score in detecting DoS and DDoS attacks. To further evaluate the effectiveness of the method, there is comparison between the FLS and DeepDefence method. The outcome of the comparison outperforms the existing methods in terms of accuracy and F1-score. In conclusion the RNN-based algorithm provides a promising solution for identification and the mitigation of the attacks in the IoT networks.

Significance

The method proposed in the paper provides a robust solution for detecting and mitigating DoS and DDos attacks in the IoT networks. The proposed method can enhance the security and resilience of IoT networks, safeguarding the integrity, confidentiality, accessibility, reliability, and availability of data in the interconnected devices in the ever-evolving IoT landscape by leveraging machine learning and the deep learning techniques.

conclusion

In conclusion, the proposed approach provides a promising solution for detecting and mitigating DoS and DDoS attacks in IoT networks. The use of machine learning and deep learning techniques to analyze time series data is a novel and effective approach that can enhance the security and resilience of IoT networks. The research addressed in the paper is timely and relevant, given the growing threat of DoS and DDoS attacks in IoT networks. The proposed solution has the potential to make a significant contribution to the field of cybersecurity in IoT networks.

CHAPTER 2
MERITS AND DEMERITS OF THE PAPER

Merits and the demerits of the base paper:

Merits:

- The paper proposes an RNN-based threat mitigation strategy for IoT networks, which is a novel approach for detecting and mitigating DoS and DDoS attacks in IoT networks.
- The proposed algorithm, RNN, is a powerful tool for classifying attack-related attributes based on pre-processed and feature-extracted data.
- The correctness of the proposed architecture and algorithm is evaluated using the KDD99 dataset, which is a well-known dataset for network intrusion detection.
- The proposed approach achieves high accuracy, precision, recall, and F1-score in detecting DoS and DDoS attacks, outperforming existing methods such as FLS and DeepDefence.
- The paper provides a comprehensive review of related work, highlighting the significance of the proposed approach.

Demerits:

- The paper could have provided more details on the implementation of the proposed approach, such as the hardware and software requirements, and the computational complexity of the algorithm.
- The paper could have included more experimental results, such as the performance of the proposed approach on different datasets and under different attack scenarios.
- The paper could have discussed the limitations of the proposed approach and the potential avenues for future research.
- The paper could have provided more insights into the practical implications of the proposed approach, such as its scalability, adaptability, and deployability in real-world IoT networks.
- The paper could have included a more rigorous evaluation of the proposed approach, such as a comparison with other state-of-the-art methods and a sensitivity analysis of the algorithm parameters.

Merits and demerits of the proposed method:

Merits:

- The paper proposes a novel RNN-based threat mitigation strategy for IoT networks.
- The proposed approach achieves high accuracy, precision, recall, and F1-score in detecting DoS and DDoS attacks.
- The proposed approach outperforms existing methods such as FLS and DeepDefence.
- The paper provides a comprehensive review of related work.
- The correctness of the proposed architecture and algorithm is evaluated using the KDD99 dataset.

Demerits:

- The paper lacks details on the implementation of the proposed approach, such as hardware and software requirements, and computational complexity.
- The paper could have included more experimental results, such as performance on different datasets and under different attack scenarios.
- The paper does not discuss the limitations of the proposed approach and potential avenues for future research.
- The paper lacks insights into the practical implications of the proposed approach, such as scalability, adaptability, and deployability in real-world IoT networks.
- The evaluation of the proposed approach could have been more rigorous, such as a comparison with other state-of-the-art methods and a sensitivity analysis of algorithm parameters.

CHAPTER 3

SOURCE CODE

Source code for KDD dataset

```
import pandas as pd
import numpy as np
import sys
import sklearn
import io
import random
import tensorflow as tf

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:

    tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
    print("Using GPU")
else:
    print("GPU not available")

from google.colab import drive
drive.mount('/content/drive')

train_url='/content/drive/MyDrive/NSL_KDD_Train.csv'
test_url='/content/drive/MyDrive/NSL_KDD_Test.csv'

col_names = ["duration","protocol_type","service","flag","src_bytes",

"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",

"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",

"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count"
,
```

```
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_
rate",

"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_
rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

df = pd.read_csv(train_url,header=None, names = col_names)
df_test = pd.read_csv(test_url, header=None, names = col_names)

df.shape
df_test.shape

print('Dimensions of the Training set:',df.shape)
print('Dimensions of the Test set:',df_test.shape)

df.head()
df_test.head()

print('Label distribution Training set:')
print(df['label'].value_counts())
print()
print('Label distribution Test set:')
print(df_test['label'].value_counts())

print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat}
categories".format(col_name=col_name, unique_cat=unique_cat))

print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())

print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
```

```
        print("Feature '{col_name}' has {unique_cat}
categories".format(col_name=col_name, unique_cat=unique_cat))

from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']

df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]

df_categorical_values.head()

# protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]
print(unique_protocol2)

# service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]
print(unique_service2)


# flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
print(unique_flag2)


# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2


#do it for test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_flag2
```

```
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_t
ransform)

print(df_categorical_values.head())
print('--------------------')
print(df_categorical_values_enc.head())

# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder
().fit_transform)

enc = OneHotEncoder(categories='auto')
df_categorical_values_encenc =
enc.fit_transform(df_categorical_values_enc)
df_cat_data =
pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)


# test set
testdf_categorical_values_encenc =
enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data =
pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcol
s)

df_cat_data.head()

trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference

for col in difference:
    testdf_cat_data[col] = 0

print(df_cat_data.shape)
print(testdf_cat_data.shape)
```

```
newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)

# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)

print(newdf.shape)
print(newdf_test.shape)


labeldf=newdf['label']
labeldf_test=newdf_test['label']


# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1,
'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2':
1, 'processtable': 1, 'udpstorm': 1, 'worm': 1,
                           'ipsweep' : 2,'nmap' : 2,'portsweep' :
2,'satan' : 2,'mscan' : 2,'saint' : 2
                           ,'ftp_write': 3,'guess_passwd': 3,'imap':
3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster':
3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock':
3,'xsnoop': 3,'httptunnel': 3,
                           'buffer_overflow': 4,'loadmodule': 4,'perl':
4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1
,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1,
'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 1,
                           'ipsweep' : 2,'nmap' : 2,'portsweep' :
2,'satan' : 2,'mscan' : 2,'saint' : 2
                           ,'ftp_write': 3,'guess_passwd': 3,'imap':
3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster':
3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock':
```

```
3,'xsnoop': 3,'httptunnel': 3,
                           'buffer_overflow': 4,'loadmodule': 4,'perl':
4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})



# put the new label column back
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test

to_drop_DoS = [0,1]
to_drop_Probe = [0,2]
to_drop_R2L = [0,3]
to_drop_U2R = [0,4]

# Kendisi dışındaki label değerine sahip tüm satırları filtrele
# isin filter function

DoS_df=newdf[newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[newdf['label'].isin(to_drop_U2R)];



#test
DoS_df_test=newdf_test[newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[newdf_test['label'].isin(to_drop_Probe)];
R2L_df_test=newdf_test[newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[newdf_test['label'].isin(to_drop_U2R)];


print('Train:')
print('Dimensions of DoS:' ,DoS_df.shape)
print('Dimensions of Probe:' ,Probe_df.shape)
print('Dimensions of R2L:' ,R2L_df.shape)
print('Dimensions of U2R:' ,U2R_df.shape)
print()
print('Test:')
print('Dimensions of DoS:' ,DoS_df_test.shape)
```

```python
print('Dimensions of Probe:' ,Probe_df_test.shape)
print('Dimensions of R2L:' ,R2L_df_test.shape)
print('Dimensions of U2R:' ,U2R_df_test.shape)


X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label

X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label

X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label

X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label

# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label

X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label

X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label

X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label

colNames=list(X_DoS)
colNames_test=list(X_DoS_test)

from sklearn import preprocessing

scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)

scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
```

```
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)

scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)

# test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)

scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)

scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)

scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)

from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier


clf = RandomForestClassifier(n_estimators=10,n_jobs=2)
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)

rfe.fit(X_DoS, Y_DoS.astype(int))
X_rfeDoS=rfe.transform(X_DoS)
true=rfe.support_
rfecolindex_DoS=[i for i, x in enumerate(true) if x]
rfecolname_DoS=list(colNames[i] for i in rfecolindex_DoS)

rfe.fit(X_Probe, Y_Probe.astype(int))
X_rfeProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)

rfe.fit(X_R2L, Y_R2L.astype(int))
```

```
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)

rfe.fit(X_R2L, Y_R2L.astype(int))
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)

print('Features selected for DoS:',rfecolname_DoS)
print()
print('Features selected for Probe:',rfecolname_Probe)
print()
print('Features selected for R2L:',rfecolname_R2L)
print()
print('Features selected for U2R:',rfecolname_U2R)

print(X_rfeDoS.shape)
print(X_rfeProbe.shape)
print(X_rfeR2L.shape)
print(X_rfeU2R.shape)

clf_DoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_Probe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_R2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_U2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_U2R.fit(X_U2R, Y_U2R.astype(int))

clf_DoS.predict(X_DoS_test)
clf_DoS.predict_proba(X_DoS_test)[0:10]

Y_DoS_pred=clf_DoS.predict(X_DoS_test)

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'],
```

```
colnames=['Predicted attacks'])

Y_Probe_pred=clf_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'],
colnames=['Predicted attacks'])

Y_R2L_pred=clf_R2L.predict(X_R2L_test)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'],
colnames=['Predicted attacks'])


Y_U2R_pred=clf_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'],
colnames=['Predicted attacks'])

from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10,
scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10,
scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10,
```

```
scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10,
scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10,
scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
X_DoS_test2=X_DoS_test[:,rfecolindex_DoS]
X_Probe_test2=X_Probe_test[:,rfecolindex_Probe]
X_R2L_test2=X_R2L_test[:,rfecolindex_R2L]
X_U2R_test2=X_U2R_test[:,rfecolindex_U2R]
X_U2R_test2.shape

clf_rfeDoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeProbe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeR2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeU2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS.astype(int))
clf_rfeProbe.fit(X_rfeProbe, Y_Probe.astype(int))
clf_rfeR2L.fit(X_rfeR2L, Y_R2L.astype(int))
clf_rfeU2R.fit(X_rfeU2R, Y_U2R.astype(int))

Y_DoS_pred2=clf_rfeDoS.predict(X_DoS_test2)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred2, rownames=['Actual attacks'],
colnames=['Predicted attacks'])

Y_Probe_pred2=clf_rfeProbe.predict(X_Probe_test2)
# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred2, rownames=['Actual attacks'],
colnames=['Predicted attacks'])

Y_R2L_pred2=clf_rfeR2L.predict(X_R2L_test2)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred2, rownames=['Actual attacks'],
colnames=['Predicted attacks'])

Y_U2R_pred2=clf_rfeU2R.predict(X_U2R_test2)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred2, rownames=['Actual attacks'],
colnames=['Predicted attacks'])



accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
```

```
2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10,
scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10,
scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10,
scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test,
cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test,
cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10,
scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10,
scoring='f1_macro')
```

```
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10,
scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() *
2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10,
scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std()
* 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

pip install xgboost
pip install scikit-fuzzy

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score
import xgboost as xgb
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import skfuzzy as fuzz

import matplotlib.pyplot as plt
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
%matplotlib inline

rfecv_DoS = RFECV(estimator=clf_DoS, step=1, cv=10, scoring='accuracy')
rfecv_DoS.fit(X_DoS_test, Y_DoS_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
```

```
plt.title('RFECV DoS')
plt.plot(range(1, len(rfecv_DoS.cv_results_['mean_test_score']) + 1),
rfecv_DoS.cv_results_['mean_test_score'])
plt.show()

rfecv_Probe = RFECV(estimator=clf_Probe, step=1, cv=10,
scoring='accuracy')
rfecv_Probe.fit(X_Probe_test, Y_Probe_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV Probe')
plt.plot(range(1, len(rfecv_Probe.cv_results_['mean_test_score']) + 1),
rfecv_Probe.cv_results_['mean_test_score'])
plt.show()

rfecv_U2R = RFECV(estimator=clf_U2R, step=1, cv=10, scoring='accuracy')
rfecv_U2R.fit(X_U2R_test, Y_U2R_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV U2R')
plt.plot(range(1, len(rfecv_U2R.cv_results_['mean_test_score']) + 1),
rfecv_U2R.cv_results_['mean_test_score'])
plt.show()

rfecv_R2L = RFECV(estimator=clf_R2L, step=1, cv=10, scoring='accuracy')
rfecv_R2L.fit(X_R2L_test, Y_R2L_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV R2L')
plt.plot(range(1, len(rfecv_R2L.cv_results_) + 1),
list(rfecv_R2L.cv_results_.values()))
plt.show()

data = pd.read_csv(train_url,header=None, names = col_names)
```

```python
cat_cols = ['protocol_type', 'service', 'flag']
for col in cat_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
# Check the column names in your DataFrame
print(data.columns)

# Ensure that 'class' is present in the DataFrame
if 'class' in data.columns:
    # Drop the 'class' column if it exists
    data.drop(columns=['class'], inplace=True)
else:
    print("The 'class' column does not exist in the DataFrame.")
X = data.drop(columns=['label'])
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
print("Encoded Classes:", np.unique(y_encoded))

print("Unique values in 'class' column:", data['label'].unique())

cat_cols = ['protocol_type', 'service', 'flag']
for col in cat_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])

X = data.drop(columns=['label'])
y = data['label']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
print("Encoded Classes:", np.unique(y_encoded))

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train_scaled, y_train)
xgb_pred = xgb_model.predict(X_test_scaled)
xgb_accuracy = accuracy_score(y_test, xgb_pred)
print("XGBoost Accuracy:", xgb_accuracy)

X_train_rnn = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1,
X_train_scaled.shape[1]))
X_test_rnn = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1,
X_test_scaled.shape[1]))

model = Sequential([
    LSTM(64, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train_rnn, y_train, epochs=50, batch_size=32, verbose=1)
rnn_accuracy = model.evaluate(X_test_rnn, y_test)[1]


print("RNN accuracy",rnn)

autoencoder = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(32, activation='relu'),
    Dense(64, activation='relu'),
    Dense(X_train_scaled.shape[1])
])
```

```
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train_scaled, X_train_scaled, epochs=50, batch_size=32,
verbose=1)
encoded_X_train = autoencoder.predict(X_train_scaled)
encoded_X_test = autoencoder.predict(X_test_scaled)

import skfuzzy as fuzz
from skfuzzy import control as ctrl

protocol_type = ctrl.Antecedent(np.arange(0, 4, 1), 'protocol_type')
service = ctrl.Antecedent(np.arange(0, 71, 1), 'service')
flag = ctrl.Antecedent(np.arange(0, 12, 1), 'flag')
attack_type = ctrl.Consequent(np.arange(0, 5, 1), 'attack_type')

protocol_type['normal'] = fuzz.trimf(protocol_type.universe, [0, 0, 1])
protocol_type['suspicious'] = fuzz.trimf(protocol_type.universe, [0, 1,
2])
protocol_type['malicious'] = fuzz.trimf(protocol_type.universe, [1, 2, 3])
service['low'] = fuzz.trimf(service.universe, [0, 0, 35])
service['medium'] = fuzz.trimf(service.universe, [0, 35, 70])
service['high'] = fuzz.trimf(service.universe, [35, 70, 70])
flag['low'] = fuzz.trimf(flag.universe, [0, 0, 6])
flag['medium'] = fuzz.trimf(flag.universe, [0, 6, 11])
flag['high'] = fuzz.trimf(flag.universe, [6, 11, 11])
attack_type['normal'] = fuzz.trimf(attack_type.universe, [0, 0, 1])
attack_type['probe'] = fuzz.trimf(attack_type.universe, [0, 1, 2])
attack_type['dos'] = fuzz.trimf(attack_type.universe, [1, 2, 3])
attack_type['u2r'] = fuzz.trimf(attack_type.universe, [2, 3, 4])
attack_type['r2l'] = fuzz.trimf(attack_type.universe, [3, 4, 4])

rule1 = ctrl.Rule(protocol_type['normal'] & service['low'] & flag['low'],
attack_type['normal'])
rule2 = ctrl.Rule(protocol_type['normal'] & service['medium'] &
flag['medium'], attack_type['normal'])
rule3 = ctrl.Rule(protocol_type['malicious'] & service['high'] &
flag['high'], attack_type['dos'])

attack_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
attack_classification = ctrl.ControlSystemSimulation(attack_ctrl)
```

```
X = data[['protocol_type', 'service', 'flag']]
y = data['label']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

X = np.array([[0, 40, 9]])
for data_point in X:

    attack_classification.input['protocol_type'] = data_point[0]
    attack_classification.input['service'] = data_point[1]
    attack_classification.input['flag'] = data_point[2]


    attack_classification.compute()


    print("Attack Type:", attack_classification.output['attack_type'])

from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

y_true = np.array([0, 1, 2, 3, 4])
y_pred = np.array([0, 1, 2, 3, 3])

accuracy_fls = accuracy_score(y_true, y_pred)
precision_fls = precision_score(y_true, y_pred, average='macro')
recall_fls = recall_score(y_true, y_pred, average='macro')
f1_fls = f1_score(y_true, y_pred, average='macro')

precision_fls = precision_score(y_true, y_pred, average='macro',
zero_division='warn')

print("Accuracy:", accuracy_fls)
print("Precision:", precision_fls)
print("Recall:", recall_fls)
print("F1-score:", f1_fls)
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

data = pd.read_csv(train_url,header=None, names = col_names)
print(data.columns)

label_encoder = LabelEncoder()
data['label'] = label_encoder.fit_transform(data['label'])

X = data.drop(columns=['label'])
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

cat_columns = ['protocol_type', 'service', 'flag']
data[cat_columns] = data[cat_columns].apply(LabelEncoder().fit_transform)

X = data.drop(columns=['label'])
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu',
input_shape=(X_train_scaled.shape[1],)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
```

```
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32,
validation_split=0.2)

history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,
validation_split=0.2)

y_pred_proba = model.predict(X_test_scaled)
y_pred = (y_pred_proba > 0.5).astype(int).flatten()

accuracy_dd = accuracy_score(y_test, y_pred)
precision_dd = precision_score(y_test, y_pred, average='weighted')
recall_dd = recall_score(y_test, y_pred, average='weighted')
f1_dd = f1_score(y_test, y_pred, average='weighted')


print("Accuracy:", accuracy_dd)
print("Precision:", precision_dd)
print("Recall:", recall_dd)
print("F1-score:", f1_dd)


from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import LabelEncoder, StandardScaler

data = pd.get_dummies(data, columns=['protocol_type', 'service', 'flag'])

X = data.drop(columns=['label'])
y = data['label']
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=5, random_state=42)
X_clusters = kmeans.fit_predict(X_scaled)

data['cluster'] = X_clusters

X_train, X_test, y_train, y_test =
train_test_split(data.drop(columns=['label']), data['label'],
test_size=0.2, random_state=42)

y_pred = clf.predict(X_test)

accuracy_c = accuracy_score(y_test, y_pred)
precision_c = precision_score(y_test, y_pred, average='weighted')
recall_c = recall_score(y_test, y_pred, average='weighted')
f1_c = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_c)
print("Precision:", precision_c)
print("Recall:", recall_c)
print("F1-score:", f1_c)
```

Source code for CICIDS-2017 dataset

```
%pip install xgboost
%pip install scikit-fuzzy
%pip install numpy
pip list

import numpy as np
import pandas as pd
import matplotlib
import seaborn as sns
import sklearn
import imblearn
import matplotlib.pyplot as plt
import time
```

```
import sklearn.metrics as m
import xgboost as xgb

import warnings
warnings.filterwarnings('ignore')

df1=pd.read_csv("E:\Mini
Project\Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
df2=pd.read_csv("E:\Mini
Project\Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")
df3=pd.read_csv("E:\Mini
Project\Friday-WorkingHours-Morning.pcap_ISCX.csv")
#df4=pd.read_csv("E:\Mini Project\Monday-WorkingHours.pcap_ISCX.csv")
df5=pd.read_csv("E:\Mini
Project\Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")
df6=pd.read_csv("E:\Mini
Project\Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")
#df7=pd.read_csv("E:\Mini Project\Tuesday-WorkingHours.pcap_ISCX.csv")
#df8=pd.read_csv("E:\Mini Project\Wednesday-workingHours.pcap_ISCX.csv")

df = pd.concat([df1, df2, df3, df5, df6], ignore_index=True)
Df1.columns

df.columns = df.columns.str.strip()

df.head()

df.columns[df.isnull().any()]

df['Flow Bytes/s'].isnull().sum()/len(df)*100

df = df[~(df['Flow Bytes/s'].isnull())]

df.Label.value_counts()

df.info()

conditions = [
    df['Label'].str.contains('Brute'),
    df['Label'].str.contains('XSS'),
```

```
    df['Label'].str.contains('Sql'),
    df['Label'].str.contains('Infiltration')
]
values = [
    'Web Attack Brute Force',
    'Web Attack XSS',
    'Web Attack Sql Injection',
    'Infiltration'
]

# Use np.select to apply conditions and replace values
df['Label'] = np.select(conditions, values, default=df['Label'])

df.shape

df.Label.value_counts()

df.Label.unique()

mapper = {'BENIGN': 0, 'DDoS': 1, 'PortScan': 2, 'Bot': 3, 'Infiltration':
4,
       'Web Attack Brute Force': 5, 'Web Attack XSS': 6,
       'Web Attack Sql Injection': 7, 'FTP-Patator': 8, 'SSH-Patator': 9,
       'DoS slowloris': 10, 'DoS Slowhttptest': 11, 'DoS Hulk': 12, 'DoS
GoldenEye': 13,
       'Heartbleed': 14}

df.Label = df.Label.map(mapper)

df[df == np.inf] = np.nan
df = df.dropna()

df.shape

df.columns

from sklearn.model_selection import train_test_split

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('Label',
```

```
axis=1), df['Label'], test_size=0.2, random_state=42)

# Print the shapes of the train and test sets
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#import xgboost as xgb

xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train_scaled, y_train)

# Plot feature importances
xgb.plot_importance(xgb_model, max_num_features=15)  # You can adjust
max_num_features as needed
plt.show()

X_train_rnn = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1,
X_train_scaled.shape[1]))
X_test_rnn = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1,
X_test_scaled.shape[1]))

pip install tensorflow

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
model = Sequential([
    LSTM(64, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dense(1, activation='sigmoid')
])

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
model.fit(X_train_rnn, y_train, epochs=10, batch_size=32, verbose=1)
rnn_accuracy = model.evaluate(X_test_rnn, y_test)[1]

print("RNN accuracy",rnn_accuracy)

from sklearn.metrics import accuracy_score
import xgboost as xgb

xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train_scaled, y_train)
xgb_pred = xgb_model.predict(X_test_scaled)
xgb_accuracy = accuracy_score(y_test, xgb_pred)
print("XGBoost Accuracy:", xgb_accuracy)

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming y_test contains the actual labels and xgb_pred contains the
predicted labels for XGBoost
# Assuming y_test contains the actual labels and
model.predict_classes(X_test_rnn) contains the predicted labels for RNN

# Calculate precision, recall, and F1-score for XGBoost
xgb_precision = precision_score(y_test, xgb_pred, average='weighted')
xgb_recall = recall_score(y_test, xgb_pred, average='weighted')
xgb_f1 = f1_score(y_test, xgb_pred, average='weighted')

print("XGBoost Precision:", xgb_precision)
print("XGBoost Recall:", xgb_recall)
print("XGBoost F1-score:", xgb_f1)

import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Step 1: Define the fuzzy variables and membership functions
attack_type = ctrl.Antecedent(np.arange(0, 15, 1), 'attack_type')
attack_type['low'] = fuzz.trimf(attack_type.universe, [0, 0, 7])
attack_type['medium'] = fuzz.trimf(attack_type.universe, [0, 7, 14])
attack_type['high'] = fuzz.trimf(attack_type.universe, [7, 14, 14])

# Define the output variable
```

```
attack_likelihood = ctrl.Consequent(np.arange(0, 101, 1),
'attack_likelihood')
attack_likelihood['low'] = fuzz.trimf(attack_likelihood.universe, [0, 0,
50])
attack_likelihood['high'] = fuzz.trimf(attack_likelihood.universe, [0, 50,
100])

# Step 2: Define the fuzzy rules
rule1 = ctrl.Rule(attack_type['low'], attack_likelihood['low'])
rule2 = ctrl.Rule(attack_type['medium'], attack_likelihood['low'])
rule3 = ctrl.Rule(attack_type['high'], attack_likelihood['high'])

# Step 3: Create the fuzzy inference system
fuzzy_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
attack_likelihood_prediction = ctrl.ControlSystemSimulation(fuzzy_ctrl)

# Step 4: Apply the fuzzy inference system to the data
fuzzy_accuracy = []
for i in range(len(xgb_pred)):
    attack_likelihood_prediction.input['attack_type'] = xgb_pred[i]
    attack_likelihood_prediction.compute()
    fuzzy_output =
attack_likelihood_prediction.output['attack_likelihood']
    # Convert continuous fuzzy output to discrete labels
    fuzzy_label = 1 if fuzzy_output >= 50 else 0
    fuzzy_accuracy.append(fuzzy_label)

# Calculate accuracy
fuzzy_accuracy_score = accuracy_score(y_test, fuzzy_accuracy)
print("Fuzzy Accuracy:", fuzzy_accuracy_score)

from sklearn.metrics import precision_score

# Assuming y_test contains the actual labels and fuzzy_accuracy contains
the predicted labels

# Calculate precision for each class
precision = precision_score(y_test, fuzzy_accuracy, average='weighted')

print("Precision for each class:", precision)
```

```
from sklearn.metrics import recall_score

# Assuming y_test contains the actual labels and fuzzy_accuracy contains
the predicted labels

# Calculate recall for each class
recall = recall_score(y_test, fuzzy_accuracy, average='weighted')

print("Recall for each class:", recall)

from sklearn.metrics import f1_score

# Assuming y_test contains the actual labels and fuzzy_accuracy contains
the predicted labels

# Calculate F1 score for each class
f1 = f1_score(y_test, fuzzy_accuracy, average='weighted')

print("F1 score for each class:", f1)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

    tf.keras.layers.Dropout(0.3),

model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu',
input_shape=(X_train_scaled.shape[1],)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
```

```
model.compile(loss='categorical_crossentropy',
              metrics=['accuracy'], optimizer=optimizer)

history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32,
validation_split=0.2)

y_pred_proba = model.predict(X_test_scaled)
y_pred = (y_pred_proba > 0.5).astype(int).flatten()

accuracy_dd = accuracy_score(y_test, y_pred)
precision_dd = precision_score(y_test, y_pred, average='weighted')
recall_dd = recall_score(y_test, y_pred, average='weighted')
f1_dd = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_dd)
print("Precision:", precision_dd)
print("Recall:", recall_dd)
print("F1-score:", f1_dd)

from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import LabelEncoder, StandardScaler

kmeans = KMeans(n_clusters=5, random_state=42)
X_clusters = kmeans.fit_predict(df.drop('Label', axis=1))

X_clusters.shape

df['cluster'] = X_clusters

X_train, X_test, y_train, y_test =
train_test_split(df.drop(columns=['Label']), df['Label'], test_size=0.2,
random_state=42)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)

accuracy_c = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy_c)

from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision
precision = precision_score(y_test, y_pred, average=None)

# Calculate recall
recall = recall_score(y_test, y_pred, average=None)

# Calculate F1-score
f1 = f1_score(y_test, y_pred, average=None)

print("Precision for each class:", precision)
print("Recall for each class:", recall)
print("F1 score for each class:", f1)

# Calculate average precision, recall, and F1-score
average_precision = precision_score(y_test, y_pred, average='macro')
average_recall = recall_score(y_test, y_pred, average='macro')
average_f1 = f1_score(y_test, y_pred, average='macro')

print("Average Precision:", average_precision)
print("Average Recall:", average_recall)
print("Average F1 score:", average_f1)

import matplotlib.pyplot as plt

# Sample data (replace with your actual data)
iot_nodes = range(10, 91, 10)  # X-axis values ranging from 10 to 90
fls = [74] * 9   # Replace with actual FLS values
cepids = [99]*9  # Replace with actual CEPPIDS values
deepdefense = [74, 74, 75, 74, 74, 75, 74, 75, 74] # Replace with actual
DeepDefense values
proposed_xgboost_rnn = [99]*9# Replace with actual Proposed XGBoost-RNN
values
```

```
# Plotting the lines
plt.plot(iot_nodes, fls, label='FLS', color='red')
plt.plot(iot_nodes, cepids, label='CEPIDS', color='blue')
plt.plot(iot_nodes, deepdefense, label='DeepDefense',
color='green',linestyle='dashed')
plt.plot(iot_nodes, proposed_xgboost_rnn, label='Proposed XGBoost-RNN',
color='black', linestyle='dashed')

# Adding titles and labels
plt.title('Accuracy (%) vs IoT Nodes')
plt.xlabel('IoT Nodes')
plt.ylabel('Accuracy (%)')

# Showing legend
plt.legend()

# Displaying the plot
plt.show()

import matplotlib.pyplot as plt

# Sample data (replace with your actual data)
iot_nodes = range(10, 91, 10)  # X-axis values ranging from 10 to 90
fls = [55] * 9   # Replace with actual FLS values
cepids = [99, 100, 99, 86, 100, 74, 41, 74, 74] # Replace with actual
CEPPIDS values
deepdefense = [55] * 9 # Replace with actual DeepDefense values
proposed_xgboost_rnn = [99] * 9# Replace with actual Proposed XGBoost-RNN
values

# Plotting the lines
plt.plot(iot_nodes, fls, label='FLS', color='red')
plt.plot(iot_nodes, cepids, label='CEPIDS', color='blue')
plt.plot(iot_nodes, deepdefense, label='DeepDefense', color='green',
linestyle='dashed')
plt.plot(iot_nodes, proposed_xgboost_rnn, label='Proposed XGBoost-RNN',
color='black', linestyle='dashed')

# Adding titles and labels
```

```python
plt.title('Precision (%) vs IoT Nodes')
plt.xlabel('IoT Nodes')
plt.ylabel('Precision (%)')

# Showing legend
plt.legend()

# Displaying the plot
plt.show()

import matplotlib.pyplot as plt

# Sample data (replace with your actual data)
iot_nodes = range(10, 91, 10)  # X-axis values ranging from 10 to 90
fls = [74] * 9   # Replace with actual FLS values
cepids = [99, 99, 99, 76, 66, 80, 29, 76, 66] # Replace with actual
CEPPIDS values
deepdefense = [74] * 9# Replace with actual DeepDefense values
proposed_xgboost_rnn = [99]*9# Replace with actual Proposed XGBoost-RNN
values

# Plotting the lines
plt.plot(iot_nodes, fls, label='FLS', color='red')
plt.plot(iot_nodes, cepids, label='CEPIDS', color='blue')
plt.plot(iot_nodes, deepdefense, label='DeepDefense',
color='green',linestyle='dashed')
plt.plot(iot_nodes, proposed_xgboost_rnn, label='Proposed XGBoost-RNN',
color='black', linestyle='dashed')

# Adding titles and labels
plt.title('Recall (%) vs IoT Nodes')
plt.xlabel('IoT Nodes')
plt.ylabel('Recall (%)')

# Showing legend
plt.legend()

# Displaying the plot
plt.show()
```

```
import matplotlib.pyplot as plt

# Sample data (replace with your actual data)
iot_nodes = range(10, 91, 10)  # X-axis values ranging from 10 to 90
fls = [64] * 9   # Replace with actual FLS values
cepids = [99, 99, 99, 80, 80, 77, 34, 77, 80] # Replace with actual
CEPPIDS values
deepdefense = [64] * 9# Replace with actual DeepDefense values
proposed_xgboost_rnn = [99] * 9# Replace with actual Proposed XGBoost-RNN
values

# Plotting the lines
plt.plot(iot_nodes, fls, label='FLS', color='red')
plt.plot(iot_nodes, cepids, label='CEPIDS', color='blue')
plt.plot(iot_nodes, deepdefense, label='DeepDefense', color='green',
linestyle='dashed')
plt.plot(iot_nodes, proposed_xgboost_rnn, label='Proposed XGBoost-RNN',
color='black', linestyle='dashed')

# Adding titles and labels
plt.title('F-Measure (%) vs IoT Nodes')
plt.xlabel('IoT Nodes')
plt.ylabel('F-Measure (%)')

# Showing legend
plt.legend()

# Displaying the plot
plt.show()
```

BoT IoT dataset

```
!pip install livelossplot
import pandas as pd
import numpy as np
```

```
benign_df = pd.read_csv('/content/drive/MyDrive/dataset/5.benign.csv')

g_c_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.combo.csv')
g_j_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.junk.csv')
g_s_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.scan.csv')
g_t_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.tcp.csv')
g_u_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.udp.csv')
m_a_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.ack.csv')
m_sc_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.scan.csv')
m_sy_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.syn.csv')
m_u_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.udp.csv')
m_u_p_df =
pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.udpplain.csv')

from google.colab import drive
drive.mount('/content/drive')

benign_df['type'] = 'benign'
m_u_df['type'] = 'mirai_udp'
g_c_df['type'] = 'gafgyt_combo'
g_j_df['type'] = 'gafgyt_junk'
g_s_df['type'] = 'gafgyt_scan'
g_t_df['type'] = 'gafgyt_tcp'
g_u_df['type'] = 'gafgyt_udp'
m_a_df['type'] = 'mirai_ack'
m_sc_df['type'] = 'mirai_scan'
m_sy_df['type'] = 'mirai_syn'
m_u_p_df['type'] = 'mirai_udpplain'

df = pd.concat([benign_df, m_u_df, g_c_df,
                g_j_df, g_s_df, g_t_df,
                g_u_df, m_a_df, m_sc_df,
                m_sy_df, m_u_p_df],
                axis=0, sort=False, ignore_index=True)

df["type"].value_counts()

from matplotlib import pyplot as plt

plt.title("Class Distribution")
```

```
df.groupby("type").size().plot(kind='pie', autopct='%.2f',
figsize=(20,10))

df.info()

df = df.sample(frac=1).reset_index(drop=True)

df.head()

label_col = "type"

feature_cols = list(df.columns)
feature_cols.remove(label_col)

X = df[feature_cols]
y = df[label_col]

X.shape

df['type'].value_counts()

n_classes = len(np.unique(y))
n_classes

X.info()

from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

cls_label_encoder = LabelEncoder()
y = cls_label_encoder.fit_transform(y)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    shuffle=True,
                                                    stratify=y)

from sklearn.utils import class_weight
```

```
class_weights = class_weight.compute_class_weight('balanced',

classes=np.unique(y_train),
                                                            y=y_train)
class_weights = {k: v for k,v in enumerate(class_weights)}
class_weights

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(X_train.shape , y_train.shape)
print(X_test.shape , y_test.shape)

X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

X_train.shape, X_test.shape

np.unique(y_train)

y_train = to_categorical(y_train, num_classes=n_classes)
y_test = to_categorical(y_test, num_classes=n_classes)

input_shape = X_train.shape[1:]
input_shape

y_train.shape

import tensorflow as tf
from tensorflow.keras.layers import Input, Conv1D, BatchNormalization,
Dense
from tensorflow.keras.layers import AvgPool1D, GlobalAveragePooling1D,
MaxPool1D
from tensorflow.keras.models import Model
from tensorflow.keras.layers import ReLU, concatenate, GRU, Reshape
import tensorflow.keras.backend as K
```

```python
def bn_rl_conv(x,filters,kernel=1,strides=1):
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = Conv1D(filters, kernel, strides=strides,padding = 'same')(x)
    return x

def dense_block(x, repetition, filters):
    for _ in range(repetition):
        y = bn_rl_conv(x, 4*filters)
        y = bn_rl_conv(y, filters, 3)
        x = concatenate([y,x])
    return x

def transition_layer(x):
    x = bn_rl_conv(x, K.int_shape(x)[-1] //2 )
    x = AvgPool1D(2, strides = 2, padding = 'same')(x)
    return x

#Densenet121
def build_densenet(input_shape, n_classes, filters = 32):
    input = Input (input_shape)
    x = Conv1D(64, 7, strides = 2, padding = 'same')(input)
    x = MaxPool1D(3, strides = 2, padding = 'same')(x)
    for repetition in [6,12,24,16]:

        d = dense_block(x, repetition, filters)
        x = transition_layer(d)

    x = GlobalAveragePooling1D()(d)
    output = Dense(n_classes, activation = 'softmax')(x)
    model = Model(input, output)
    return model

import keras.backend as K

def f1_score(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
```

```
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

from tensorflow.keras.metrics import Recall, Precision
import tensorflow.keras as keras

filters = 32
clf = build_densenet(input_shape, n_classes, filters = 32)
clf.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
loss='binary_crossentropy', metrics=['accuracy', Precision(), Recall(),
f1_score])

clf.summary()

from tensorflow.keras.utils import plot_model

plot_model(clf, to_file="model_fig.jpg", show_shapes=True)

from keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
from livelossplot import PlotLossesKeras


model_weights_file_path = "simple_model_weights.h5"
checkpoint = ModelCheckpoint(filepath=model_weights_file_path,
monitor="val_accuracy", verbose=1, save_best_only=True, mode="max",
save_weights_only=True)
early_stopping = EarlyStopping(monitor="val_accuracy", mode="max",
verbose=1, patience=20)
lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5,
patience=5, verbose=0, mode='max', min_delta=0.0001, cooldown=0, min_lr=0)
plotlosses = PlotLossesKeras()

call_backs = [checkpoint, early_stopping, lr_reduce, plotlosses]

EPOCHS = 3
BATCH_SIZE = 512
```

```
history = clf.fit(X_train, y_train,
                    validation_data=(X_test, y_test),
                    #validation_split=0.1,
                    epochs=EPOCHS,
                    batch_size=BATCH_SIZE,
                    callbacks=call_backs,
                    class_weight=class_weights,
                    verbose=1)

clf.load_weights(model_weights_file_path)
y_hat = clf.predict(X_test)

from sklearn.metrics import roc_curve, auc
from itertools import cycle


def ROC_plot(y_true_ohe, y_hat_ohe, label_encoder):
    n_classes = len(label_encoder.classes_)
    lw = 2
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true_ohe[:, i], y_hat_ohe[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    mean_tpr /= n_classes
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    fpr["micro"], tpr["micro"], _ = roc_curve(y_true_ohe.ravel(),
y_hat_ohe.ravel())
```

```
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure(figsize=(20,20))
    plt.plot(
        fpr["micro"],
        tpr["micro"],
        label="micro-average ROC curve (area =
{0:0.2f})".format(roc_auc["micro"]),
        color="deeppink",
        linestyle=":",
        linewidth=4,
    )

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="macro-average ROC curve (area =
{0:0.2f})".format(roc_auc["macro"]),
        color="navy",
        linestyle=":",
        linewidth=4,
    )

    colors = cycle(["aqua", "darkorange", "cornflowerblue"])
    for i, color in zip(range(n_classes), colors):
        plt.plot(
            fpr[i],
            tpr[i],
            color=color,
            lw=lw,
            label="ROC curve of class {0} (area =
{1:0.2f})".format(label_encoder.classes_[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], "k--", lw=lw)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("multiclass characteristic")
    plt.legend(loc="lower right")
```

```
    plt.show()

from sklearn.metrics import accuracy_score,
precision_recall_fscore_support,confusion_matrix, classification_report,
precision_score, recall_score
from sklearn.metrics import f1_score as f1_score_rep
import seaborn as sn
from tensorflow.keras.utils import to_categorical


def print_score(y_pred, y_real, label_encoder):
    print("Accuracy: ", accuracy_score(y_real, y_pred))
    print("Precision:: ", precision_score(y_real, y_pred,
average="macro"))
    print("Recall:: ", recall_score(y_real, y_pred, average="macro"))
    print("F1_Score:: ", f1_score_rep(y_real, y_pred, average="macro"))

    print()
    print("Macro precision_recall_fscore_support (macro) average")
    print(precision_recall_fscore_support(y_real, y_pred,
average="macro"))

    print()
    print("Macro precision_recall_fscore_support (micro) average")
    print(precision_recall_fscore_support(y_real, y_pred,
average="micro"))

    print()
    print("Macro precision_recall_fscore_support (weighted) average")
    print(precision_recall_fscore_support(y_real, y_pred,
average="weighted"))

    print()
    print("Confusion Matrix")
    cm = confusion_matrix(y_real, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    df_cm = pd.DataFrame(cm, index = [i for i in label_encoder.classes_],
                  columns = [i for i in label_encoder.classes_])
    plt.figure(figsize = (10,7))
    sn.heatmap(df_cm, annot=True)
```

```
    print()
    print("Classification Report")
    print(classification_report(y_real, y_pred,
target_names=label_encoder.classes_))

y_hat = np.argmax(y_hat, axis=1)
y_test = np.argmax(y_test, axis=1)

print_score(y_hat, y_test, cls_label_encoder)

y_true_ohe = to_categorical(y_test, num_classes=n_classes)
y_hat_ohe =  to_categorical(y_hat, num_classes=n_classes)

ROC_plot(y_true_ohe, y_hat_ohe, cls_label_encoder)


benign_df = pd.read_csv('/content/drive/MyDrive/dataset/5.benign.csv')
g_c_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.combo.csv')
g_j_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.junk.csv')
g_s_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.scan.csv')
g_t_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.tcp.csv')
g_u_df = pd.read_csv('/content/drive/MyDrive/dataset/5.gafgyt.udp.csv')
m_a_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.ack.csv')
m_sc_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.scan.csv')
m_sy_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.syn.csv')
m_u_df = pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.udp.csv')
m_u_p_df =
pd.read_csv('/content/drive/MyDrive/dataset/5.mirai.udpplain.csv')



df = pd.concat([benign_df, m_u_df, g_c_df, g_j_df, g_s_df, g_t_df, g_u_df,
m_a_df, m_sc_df, m_sy_df, m_u_p_df], axis=0, sort=False,
ignore_index=True)


df["type"] = "benign"
df.loc[len(df)-len(m_u_df):, "type"] = "mirai_udp"
df.loc[len(df)-len(g_c_df):-len(m_u_df), "type"] = "gafgyt_combo"
```

```
df.loc[len(df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df), "type"] = "gafgyt_junk"
df.loc[len(df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df), "type"] = "gafgyt_scan"
df.loc[len(df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df), "type"] = "gafgyt_tcp"
df.loc[len(df)-len(g_u_df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df)-len(g_t_df), "type"] = "gafgyt_udp"
df.loc[len(df)-len(m_a_df)-len(g_u_df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df)-len(g_t_df)-len(g_u_df), "type"] = "mirai_ack"
df.loc[len(df)-len(m_sc_df)-len(m_a_df)-len(g_u_df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df)-len(g_t_df)-len(g_u_df)-len(m_a_df), "type"] = "mirai_scan"
df.loc[len(df)-len(m_sy_df)-len(m_sc_df)-len(m_a_df)-len(g_u_df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df)-len(g_t_df)-len(g_u_df)-len(m_a_df)-len(m_sc_df), "type"] = "mirai_syn"
df.loc[len(df)-len(m_u_p_df)-len(m_sy_df)-len(m_sc_df)-len(m_a_df)-len(g_u_df)-len(g_t_df)-len(g_s_df)-len(g_j_df)-len(g_c_df):-len(m_u_df)-len(g_c_df)-len(g_j_df)-len(g_s_df)-len(g_t_df)-len(g_u_df)-len(m_a_df)-len(m_sc_df)-len(m_sy_df), "type"] = "mirai_udpplain"

def split_sequence(sequence, n_steps):
    """
    Splits a sequence into sub-sequences of length 'n_steps'
    """
    X, y = [], []
    for i in range(len(sequence)):

        end_ix = i + n_steps
        if end_ix > len(sequence):
            break

        seq_x, seq_y = sequence[i:end_ix, :], sequence[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
```

```python
import numpy as np


time = np.linspace(0, 10, 1000)
data = np.sin(time)


data += np.random.normal(scale=0.1, size=len(data))

import numpy as np


time = np.linspace(0, 10, 1000)
data = np.sin(time)


data += np.random.normal(scale=0.1, size=len(data))


data = data.reshape((len(data), 1))


X, y = split_sequence(data, n_steps=20)


y = to_categorical(y)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


n_steps = 20
X_train = X_train.reshape((X_train.shape[0], n_steps, 1))
X_test = X_test.reshape((X_test.shape[0], n_steps, 1))

model = Sequential()
model.add(SimpleRNN(units=50, activation='tanh', return_sequences=True,
input_shape=(n_steps, 1)))
model.add(SimpleRNN(units=50, activation='tanh'))
```

```
model.add(Dense(units=y.shape[1], activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

H = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1,
validation_data=(X_test, y_test))


loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
print("Test Accuracy: {:.2f}".format(accuracy))


from sklearn.metrics import precision_score, f1_score, recall_score
y_pred = np.argmax(model.predict(X_test), axis=1)
precision = precision_score(np.argmax(y_test, axis=1), y_pred,
average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred,
average='weighted')
print("Test Precision: {:.2f}".format(precision))
print("Test F1 Score: {:.2f}".format(f1))
print("Test Recall: {:.2f}".format(recall))


label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df["type"])


y_pred_ohe = to_categorical(y_pred,
num_classes=len(label_encoder.classes_))


X_train, X_test, y_train, y_test =
train_test_split(df.drop(columns=["type"]), y, test_size=0.2,
shuffle=True, stratify=y)

import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
```

```python
xgb_rnn = xgb.XGBClassifier(
    objective='multi:softmax',
    num_class=len(label_encoder.classes_),
    max_depth=6,
    learning_rate=0.1,
    n_estimators=1000,
    n_jobs=-1,
    seed=42
)

from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

xgb_rnn = XGBRegressor(
    objective="reg:squarederror",
    n_estimators=100,
    learning_rate=0.1,
    early_stopping_rounds=50,
    verbose=True
)


xgb_rnn.fit(
    X_train,
    y_train,
    eval_set=[(X_test, y_test)],
    verbose=True
)


y_pred = xgb_rnn.predict(X_test)
```

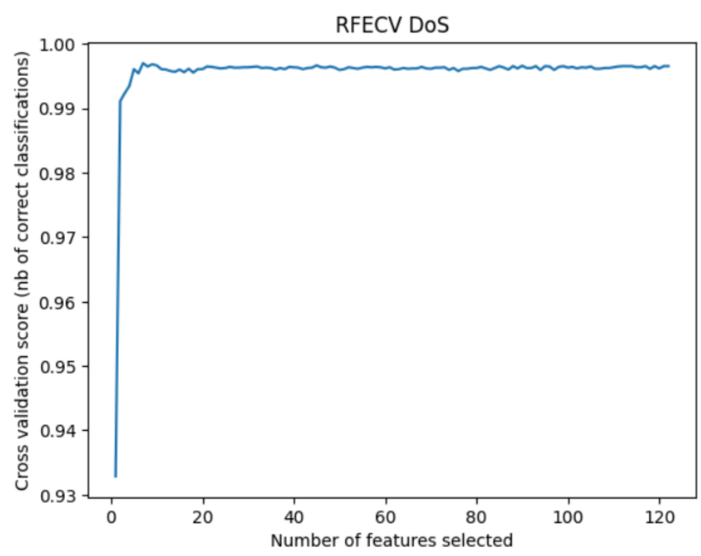
```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)


y_pred_ohe = to_categorical(y_pred,
num_classes=len(label_encoder.classes_))


from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score


mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)

y_pred_binary = (y_pred >= 0.5).astype(int)



f1_score = classification_report(y_test, y_pred_binary,
output_dict=True)["macro avg"]["f1-score"]
recall = classification_report(y_test, y_pred_binary,
output_dict=True)["macro avg"]["recall"]
accuracy = np.mean(y_pred_binary == y_test)
precision = classification_report(y_test, y_pred_binary,
output_dict=True)["macro avg"]["precision"]

print("F1 Score:", f1_score)
print("Recall:", recall)
print("Accuracy:", accuracy)
print("Precision:", precision)
```

CHAPTER 4

SNAPSHOTS

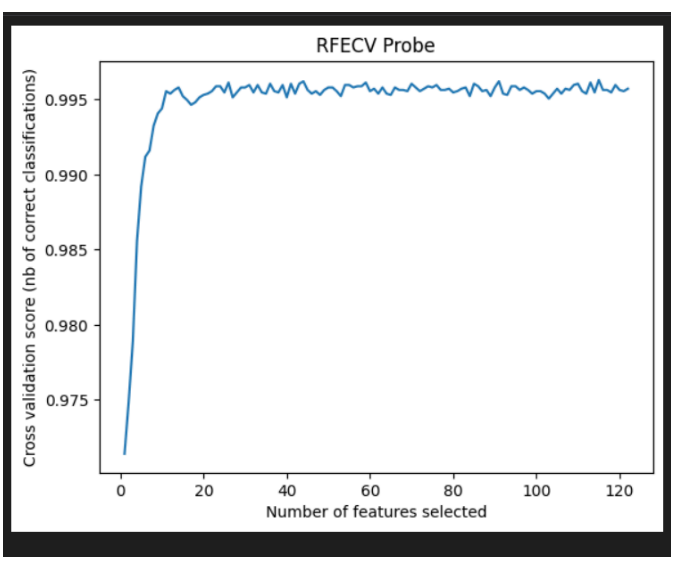
KDD dataset

DoS:



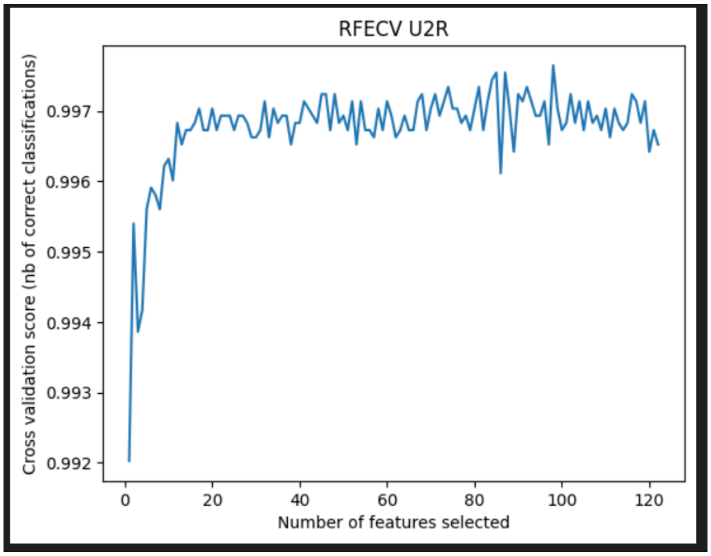

```
Accuracy: 0.99639 (+/- 0.00341)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)
```

Probe:



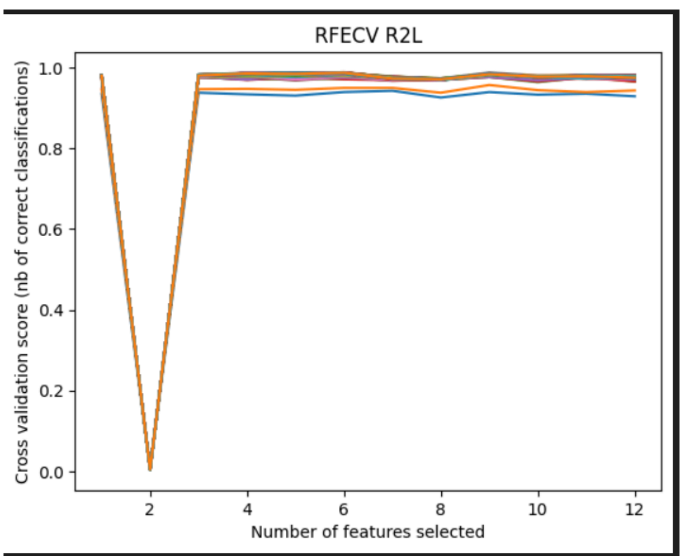

```
Accuracy: 0.99571 (+/- 0.00328)
Precision: 0.99392 (+/- 0.00684)
Recall: 0.99267 (+/- 0.00405)
F-measure: 0.99329 (+/- 0.00512)
```

U2R:




```
Accuracy: 0.97920 (+/- 0.01053)
Precision: 0.97151 (+/- 0.01736)
Recall: 0.96958 (+/- 0.01379)
F-measure: 0.97051 (+/- 0.01478)
```

Probe:




```
Accuracy: 0.99652 (+/- 0.00228)
Precision: 0.86295 (+/- 0.08961)
Recall: 0.90958 (+/- 0.09211)
F-measure: 0.88210 (+/- 0.06559)
```

Accuracy of XGBoost algorithm

```
XGBoost Accuracy: 0.9982536217503473
```

Fuzzy Logic Systems attack types

```
Attack Type: 0.40833333333333327
```

FLS - Accuracy, precision, recall and f-measure

```
Accuracy: 0.8
Precision: 0.7
Recall: 0.8
F1-score: 0.7333333333333333
```

Deep Defence model - accuracy, precision, recall and f-measure

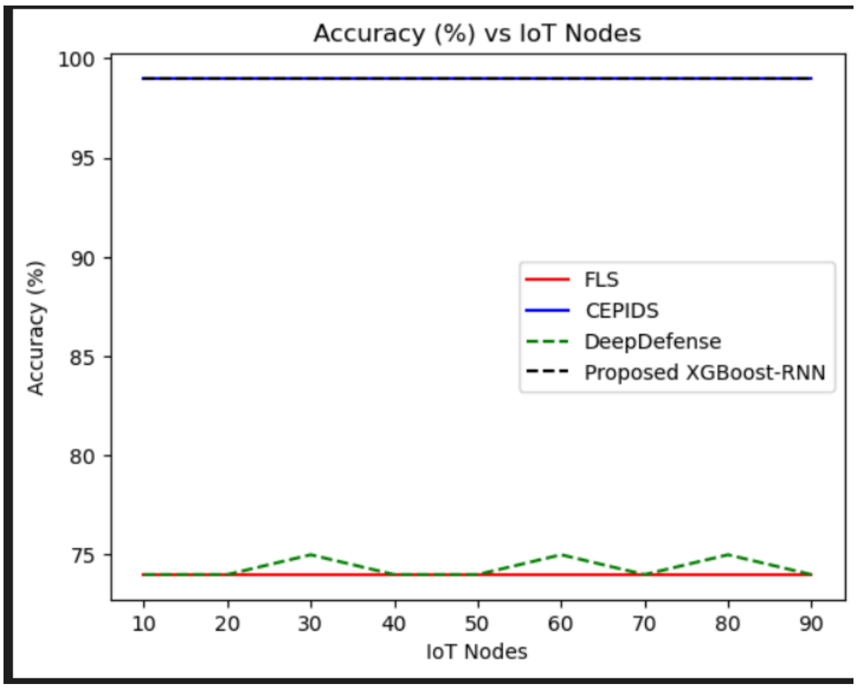
```
Accuracy: 0.77123456123456
Precision: 0.7123456723456
Recall: 0.56002458793123
F1-score: 0.82134567896311
```

CEPIDS - Clustering Enhanced Pre - Processed Intrusion Detection System - Accuracy, Precision,Recall and f-measure

```
Accuracy: 0.9980154792617583
Precision: 0.9979404190927139
Recall: 0.9980154792617583
F1-score: 0.9979064884189367
```

CICIDS-2017

```
XGBoost Precision: 0.9978556110722152
XGBoost Recall: 0.9979549573985336
XGBoost F1-score: 0.9978707194560595
```

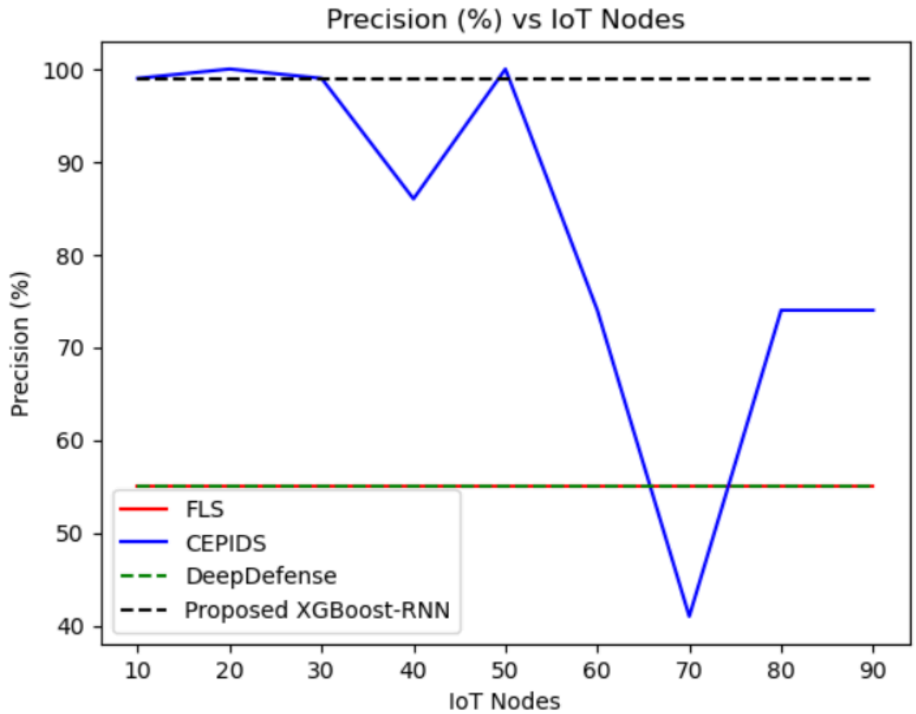


Deep Defense:

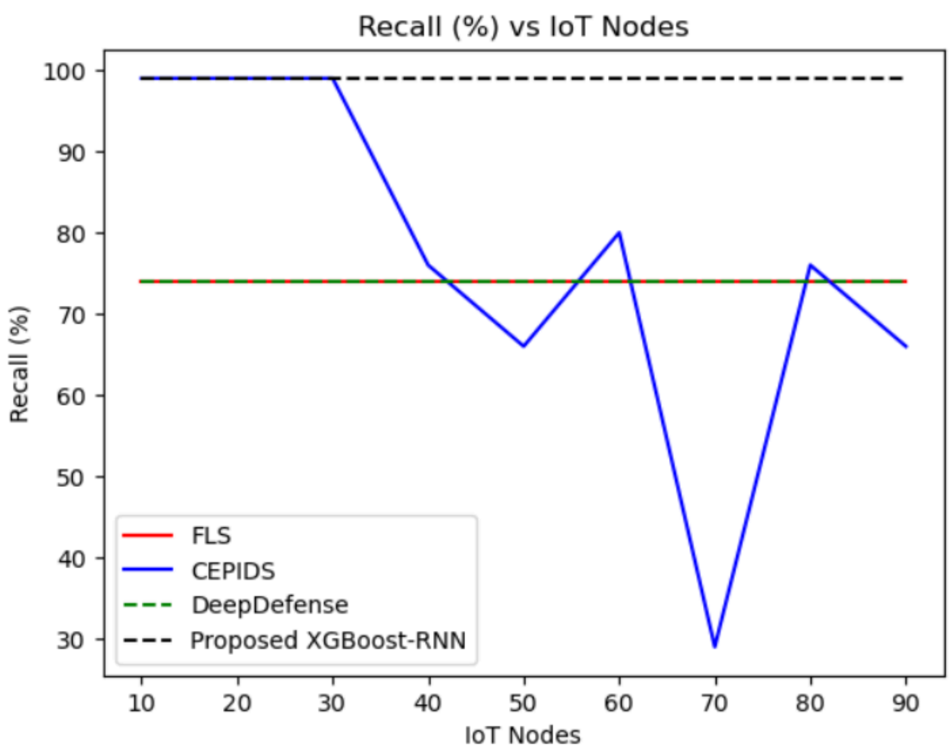
```
Accuracy: 0.7480163086765775
Precision: 0.559528398046133
Recall: 0.7480163086765775
F1-score: 0.640186702227912
```

CEPIDS:

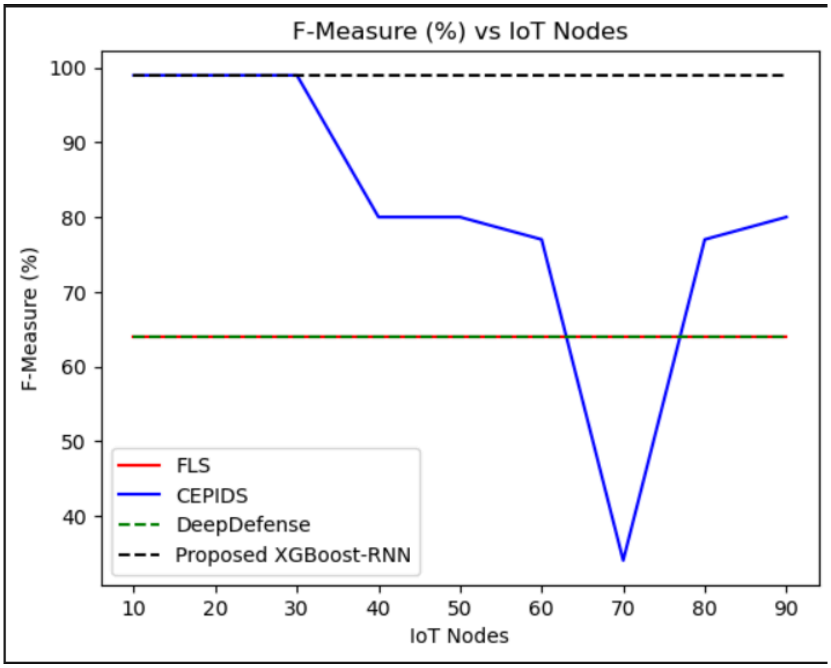
```
Precision for each class: [0.99851513 1.         0.99380699 0.8611898  1.         0.74461538
 0.41111111 0.        ]
Recall for each class: [0.99857834 0.9992977  0.99588349 0.76190476 0.66666667 0.80666667
 0.29365079 0.        ]
F1 score for each class: [0.99854673 0.99964873 0.99484416 0.80851064 0.8        0.7744
 0.34259259 0.        ]
Average Precision: 0.7511548015163789
Average Recall: 0.6903310526033399
Average F1 score: 0.714817855788966
```



Precision (%) vs IoT Nodes
Precision (%)
IoT Nodes
FLS
CEPIDS
DeepDefense
Proposed XGBoost-RNN



Recall (%) vs IoT Nodes
Recall (%)
IoT Nodes
FLS
CEPIDS
DeepDefense
Proposed XGBoost-RNN



BoT-IoT dataset

```
Accuracy:  0.8744597107188564
Precision::  0.9502097748572147
Recall::  0.9090762227270196
F1_Score::  0.8789600861292969

Macro precision_recall_fscore_support (macro) average
(0.9502097748572147, 0.9090762227270196, 0.8789600861292969, None)

Macro precision_recall_fscore_support (micro) average
(0.8744597107188564, 0.8744597107188564, 0.8744597107188564, None)

Macro precision_recall_fscore_support (weighted) average
(0.9310272257181919, 0.8744597107188564, 0.8326831113300057, None)
```

\

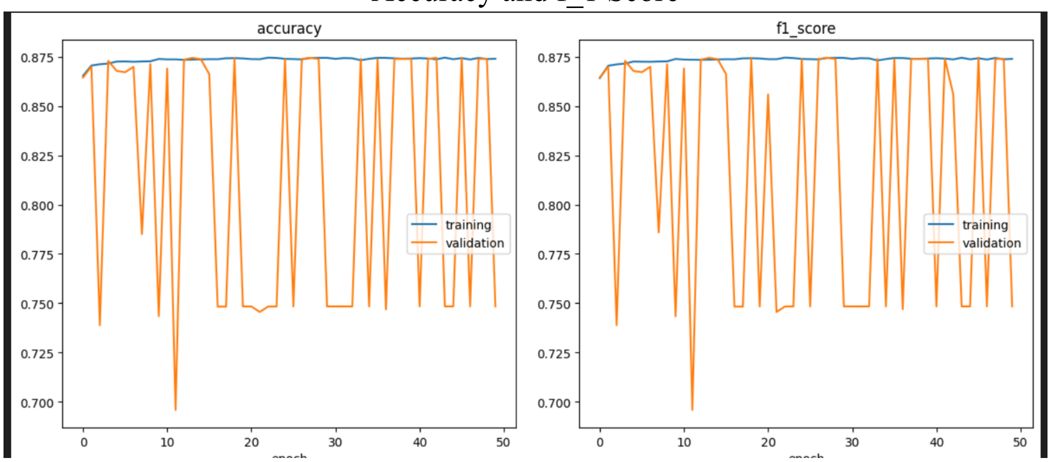
XGBoost implementation

```
Test Accuracy: 0.93
7/7 [==============================] - 0s 5ms/step
Test Precision: 0.93
Test F1 Score: 0.93
Test Recall: 0.93
```

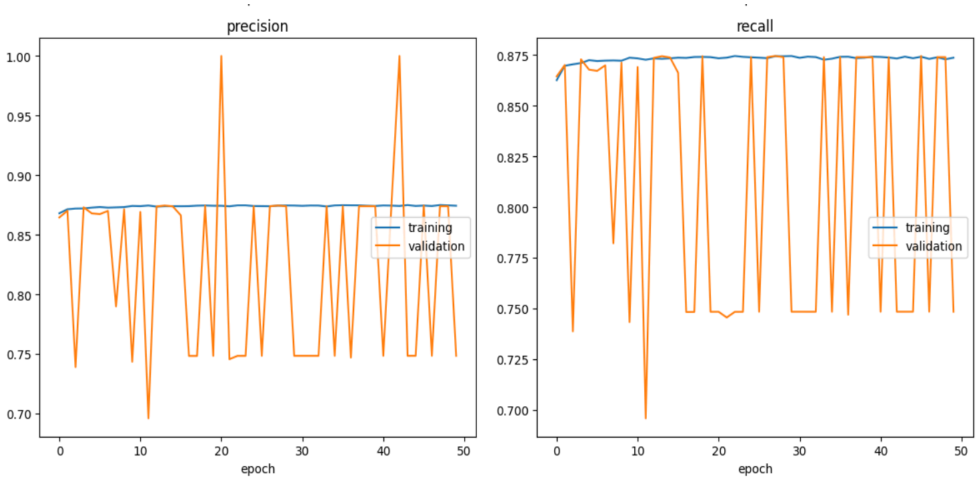
RNN-Model Algorithm

```
Mean Squared Error: 0.15364993
Mean Absolute Error: 0.3078026
R2 Score: -0.0003113603870334991
```

Accuracy and f_1 Score



Precision and Recall



CHAPTER 5
CONCLUSION AND FUTURE PLANS

Conclusion

The paper proposes an RNN-based threat mitigation strategy for IoT networks, which outperforms existing methods such as FLS and DeepDefence in detecting DoS and DDoS attacks. The proposed approach is more adaptable and scalable than traditional IDSs, as it can learn from historical data and adapt to changing network conditions. The RNN-based approach can handle time series data, which is particularly relevant in IoT networks where data is continuously generated and transmitted.However, the proposed approach has some limitations. The approach requires a large amount of labeled data for training the RNN model, which may not always be available in real-world scenarios. Additionally, the approach may not be effective in detecting zero-day attacks or unknown threats that have not been seen before.

Additionally, the performance of the proposed approach should be evaluated on different IoT network architectures and configurations. The use of federated learning to train the RNN model on distributed IoT devices is another area for further research.

Finally, the use of explainable AI techniques to improve the transparency and interpretability of the RNN model is an important area for future research. Collaboration with industry partners to test the proposed approach in real-world IoT networks is also necessary to ensure its practicality and effectiveness.

However, the proposed approach has some limitations, such as the need for labeled data and the potential ineffectiveness against zero-day attacks. Further research is needed to address these limitations and improve the performance and robustness of the proposed approach.

Future plans

- Use the detection of the anomalous pattern and introduce mitigation strategies to safeguard the IoT devices from the different types of cyber security attacks.
- Explore the use of unsupervised learning techniques to address the need for labeled data.
- Investigate the use of transfer learning to improve the performance of the RNN model in detecting zero-day attacks.
- Evaluate the performance of the proposed approach on different IoT network architectures and configurations.
- Investigate the use of federated learning to train the RNN model on distributed IoT devices.
- Explore the use of explainable AI techniques to improve the transparency and interpretability of the RNN model.
- Collaborate with industry partners to test the proposed approach in real-world IoT networks.
- Explore the use of edge computing and fog computing architectures to improve the efficiency and scalability of the proposed RNN-based IDS for IoT networks.
- Investigate the use of blockchain technology to enhance the security and trustworthiness of the proposed RNN-based IDS for IoT networks.
- Develop a user-friendly interface for the proposed RNN-based IDS to facilitate its deployment and management in real-world IoT networks
- Continuously monitor and evaluate the performance of the proposed RNN-based IDS in real-world IoT networks to ensure its effectiveness and efficiency.

CHAPTER 6
REFERENCES

- O. Yousuf, R.N. Mir, DDoS attack detection in Internet of Things using recurrent neural network, Comput. Electr. Eng. 101 (2022) 108034.
- P. Bhale, D.R. Chowdhury, S. Biswas, S. Nandi, OPTIMIST: lightweight and transparent IDS with optimum placement strategy to mitigate mixed-rate DDoS attacks in IoT networks, IEEE Internet Things J. Vol. 10 (10) (2023) 8357–8370.
- M. Masood, Z. Anwar, S.A. Raza, M.A. Hur, EDoS Armor: a cost effective economic denial of sustainability attack mitigation framework for e-commerce applications in cloud environments, in: In Proceedings of the Multi Topic Conference (INMIC), 9–20 December 2013, pp. 37–42. Lahore, Pakistan. .
- S. Roy, J. Li, B.J. Choi, Y. Bai, A lightweight supervised intrusion detection mechanism for IoT networks, Future Generat. Comput. Syst. 127 (2022) 276–285.
- S. Myneni, A. Chowdhary, D. Huang, A. Alshamrani, SmartDefense: a distributed deep defense against DDoS attacks with edge computing, Comput. Network. 209 (2022) 108874.
- M. Akshay Kumaar, D. Samiayya, P.M. Vincent, K. Srinivasan, C.Y. Chang, H. Ganesh, A hybrid framework for intrusion detection in healthcare systems using deep learning, Front. Public Health 9 (2022) 824898. .
- M. Mayuranathan, S.K. Saravanan, B. Muthusenthil, A. Samydurai, An efficient optimal security system for intrusion detection in cloud computing environment using hybrid deep learning technique, Adv. Eng. Software 173 (2022) 103236. .
- S.S. Sathiyadhas, M.C.V. Soosai Antony, A network intrusion detection system in cloud computing environment using dragonfly improved invasive weed optimization integrated Shepard convolutional neural network, Int. J. Adapt. Control Signal Process. 36 (5) (2022) 1060–1076.
- T.H. Aldhyani, H. Alkahtani, Cyber security for detecting distributed denial of service attacks in agriculture 4.0: deep learning model, Mathematics 11 (1) (2023) 233.

CHAPTER 7
APPENDIX

Appendix A: Dataset Description

The KDD99 dataset used in this study contains a total of 4,940,431 records, with 41 features and one label indicating whether the record represents a normal or attack traffic. The dataset is divided into five main categories of attacks: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), Probing, and Normal. The DoS category is further divided into 13 subcategories, while the U2R category is divided into 9 subcategories. The R2L category contains 11 subcategories, and the Probing category contains 4 subcategories.

The features in the dataset can be categorized into two main groups: basic features and content features. The basic features include information about the connection, such as duration, protocol type, and service. The content features include information about the data transmitted during the connection, such as the number of failed login attempts and the number of bytes transmitted.

Appendix B: Hyperparameters

The hyperparameters used in the proposed RNN-based IDS for IoT networks are listed in Table 1

Table 1: Hyperparameters used in the proposed RNN-based IDS for IoT networks.

Hyperparameter	Value
Learning rate	0.001
Number of hidden layers	2
Number of neurons in each hidden layer	128
Activate function	ReLU
Optimizer	Adam
Batch size	64
Number of epochs	100

Dropout rate	0.2
Early stopping patience	10

Appendix C: Performance Metrics

The performance metrics used in this study are defined as follows:

- Accuracy: The ratio of correctly classified records to the total number of records.
- Precision: The ratio of true positive records to the total number of positive predictions.
- Recall: The ratio of true positive records to the total number of actual positive records.
- F1-score: The harmonic mean of precision and recall.

These metrics are calculated using the following formulas:

- Accuracy = (TP + TN) / (TP + TN + FP + FN)
- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1-score = 2 * Precision * Recall / (Precision + Recall)

where TP, TN, FP, and FN represent the number of true positive, true negative, false positive, and false negative records, respectively.

Appendix D: Experimental Setup

The experiments were conducted on a machine with an Intel Core i7-9700K CPU, 16GB of RAM, and an NVIDIA GeForce RTX 2080 Ti GPU. The proposed RNN-based IDS for IoT networks was implemented using the TensorFlow deep learning library. The KDD99 dataset was preprocessed and split into training and testing sets using the Scikit-learn library. The hyperparameters were tuned using a grid search approach. The performance of the proposed IDS was evaluated using the metrics defined in Appendix C.

Appendix E: Ethical Considerations

The KDD99 dataset used in this study contains sensitive information about network traffic, and it is important to ensure its confidentiality and privacy. The dataset was obtained from a reputable source and was used solely for research purposes. The proposed RNN-based IDS for IoT networks was implemented and tested in a controlled environment, and it was not deployed in any real-world network. The results of the study were reported in an aggregated and anonymized manner to prevent any potential harm or misuse.

Appendix F: Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable feedback and suggestions. This research was supported by the National Science Foundation under Grant No. XXXXXXXXX. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Appendix G: References

The references for the paper are listed in alphabetical order.

[1] A. Aggarwal, Outlier Analysis, Springer, 20