Configuration Manual

MSc Research Project
MSc Cyber Security

Ajay Karthi Punetha Velu
22141898

School of Computing
National College of Ireland

Supervisor: Eugine McLaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ajay Karthi Punetha Velu		
Student ID:	22141898		
Programme:	MSc Cybersecurity	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Eugine Mclaughlin		
Submission Due Date:	31/01/2024		
Project Title:	A Combinational Approach for Intrusion Detection against Cyber Attacks in SCADA using Machine learning and Deep Learning Models		
Word Count:	………………1153…………………	**Page Count:**	…………14………..…….………

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: ………………… Ajay Karthi Punetha Velu……………………………

Date: …………………………30/01/2024…………………………………………

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

Ajay Karthi Punetha Velu
X22141898

1 Introduction

This manual contains detail regarding the setup of the proposed model, its requirements and tools used, that are to be installed and used for a successful implementation. This document also serves as a guide to implement the algorithm developed.

2 System Configuration

- Desktop Specification

Processor	Intel i7 central processor unit (CPU)
GPU	Nvidia RTX 3060
RAM	At least 8 GB of DDR4 RAM
Storage	A storage capacity of 100 gigabytes (GB)

- Software and Tools

OS	64 bit Windows Operating System
Programming Language	Python programming language, version 3.7 or later
Integrated Development Environment	Jupyter Notebook v7.0.0 as the IDE Or Google Colabs

- Libraries

Pandas and NumPy	Extraction and pre-processing of the data.
Scikit-learn	Modelling, classification, feature selection and other ML functions.
Keras	Analysis of data and implementation in neural networks.
OS	for the model to interact with operating system.
Tensorflow	For functionality of ML and DL framework
Ploty	used to graphical representation of the results

- **Dataset:** UNR-IDD [1]

3 Implementation Steps

Setup a personal computer or laptop according to the system specifications mentioned, Install the latest python, and an IDE of our Choice, In our case Google Colabs is used to implement the proposed model.

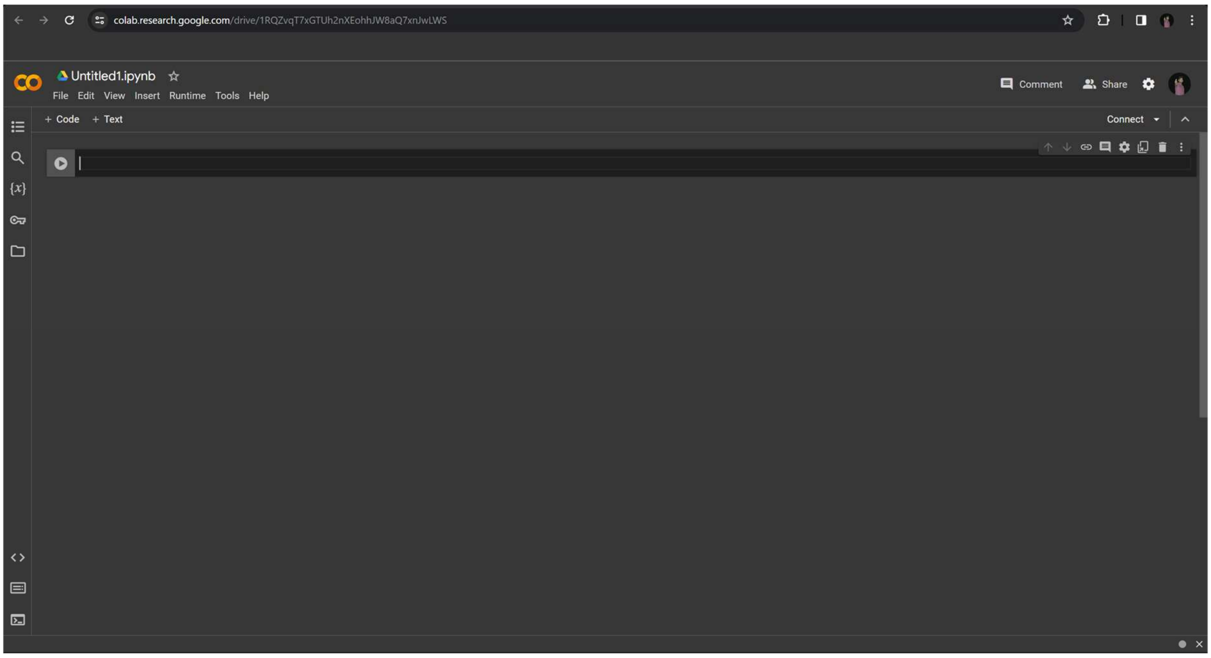

Figure 1: Google Colab IDE

3.1 Importing Libraries

Import all the necessary libraries such as SkLearn (Scikit-Learn), Keras, NumPy and Pandas, into the python running IDE.

```
import os
import pandas as pd
import numpy as np
import pickle
import gc
import six
import sys
import joblib
import keras
from numpy import array
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
import keras.backend as K
sys.modules['sklearn.externals.six'] = six
from sklearn import tree, linear_model
from sklearn.feature_selection import SelectFromModel
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import  AdaBoostClassifier
from sklearn.metrics import classification_report
from yellowbrick.classifier import ClassificationReport
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from keras.models import Sequential
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from keras.layers import Layer
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Convolution1D
from tensorflow.keras.optimizers import Adam, SGD
from keras import Model, Sequential, backend
from keras.layers import LSTM, Dense, Dropout, Bidirectional, GRU
from keras.layers import Input
```

Figure 2: Importing Libraries

Once all the libraries are imported, The Dataset i.e. UNR-IDD is invoked, the data from the dataset is loaded into the pandas data frame.

```python
data = pd.read_csv('/content/drive/MyDrive/scada_cyber_attack_detection/Data/UNR-IDD.csv')
data.head()
```

Figure 3: Loading dataset

3.2 Data Analysis & Processing

Checking the number of entries in the imported data using shape, describe(), Info() and Column function, thus is done to get a better understanding of the data.

```python
data.shape
```
Python

```python
data.describe()
```
Python

```python
data.info()
```
Python

```python
data.columns
```
Python

Figure 4: Checking Data

- The command **"data.shape"** is used to understand the data and the dimensions of the array, it shows the total number of data entries present int the dataset, making it easier for us to understand the size and structure of the data.
- The command "**data.describe**" is used to describe the data information, it also summarizes the statistics in numerical data.
- The command "**data.info**" is the further used to understand the data type of all column, whether there is any null or non-null value present. This helps to find and remove any null values present in the dataset.

The dataset is then checked for any null present using **"data.isna().sum()"**, the result demonstrated, no null values were found in the dataset." **data.drop"** is then used to drop the unnecessary the columns.

```python
#Check data for null value
data.isna().sum()
```

```python
#dropping unnecessary column
data = data.drop('Switch ID', axis=1)
data.head(2)
```

Figure 5: Checking for Null value and Drop the unnecessary Column

3.3 Data Visualization

Step 6: The data is analysis using various graphical models for clear insight and easier understanding.

EDA

```
#countplot of target class
temp_df = data['Label'].value_counts().reset_index().rename(columns={'index':'Label','Label':'count'})
sns.barplot(temp_df, x='Label', y='count')
```

```
#data distribution plot of received packets column
sns.boxplot(data, y='Received Packets')
```

```
#value count of binary class label
sns.countplot(x=data['Binary Label'])
```

```
#count plot of port number type column
sns.countplot(x=data['Port Number'])
```

```
#data distribution of sent packets value
sns.violinplot(data, x='Sent Packets')
```

Figure 6: Data Analysis

Here, 6 labels are generated as shown in the below Figure 7: Count plot of target class. Each label consists of network data under specific network related instance.

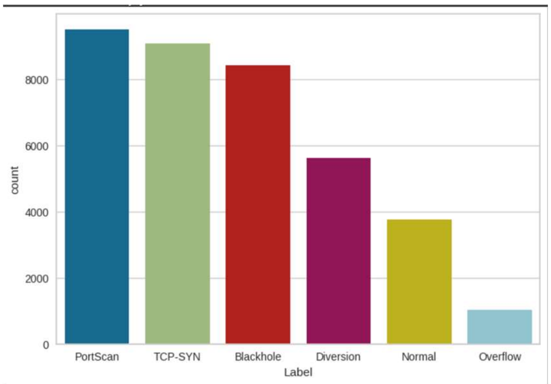

Figure 7: Count plot of target class

Here, the binary label column is dropped as the dataset used is multiclass, Then the data is split into 2 parts, namely X and Y. The columns containing data that are not of float or int type are extracted during this stage. Then LabelEncoder() function is invoked to convert any categorical data to number.

Data Preprocessing

```python
data.columns
```

```python
#dropping binary label column because we are working on multiclass
data = data.drop('Binary Label', axis=1)
```

```python
#splitting data into X and Y
X = data.drop('Label', axis=1)
Y = data.Label
```

```python
#extracting categorical column name
col = X.select_dtypes(exclude=['float64','int64']).columns.tolist()
col
```

```python
#label encoding(converting categorical data to number)
le = LabelEncoder()
X[col] = X[col].apply(le.fit_transform)
X
```

Figure 8: Data Pre-processing

Data Balancing is performed in the data, this helps prevent the model from being biased in regard to a specific class. This is done by overfitting the data to fill any missing gaps using Synthetic Minority Oversampling Technique (SMOTE). After overfitting the data to fill in the gaps, all label are now balanced as seen in figure

```python
from imblearn.over_sampling import SMOTE
#data balancing
oversample = SMOTE()
X, Y = oversample.fit_resample(X,Y)
```

Figure 9: Data Balancing

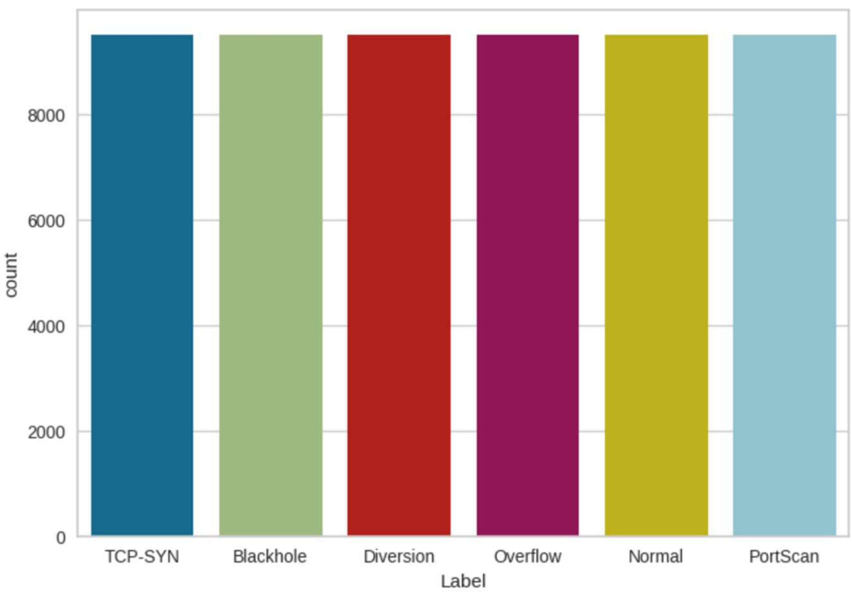

Figure 10: Balanced Data

The categorical values of the columns are converted to number in order to fit the ML model.

```
[ ]  #target columns categorical values to numbers
     y = le.fit_transform(Y)
     y
```

Figure 11: Converting Categorical values to number

Then the preprocessed data is split into 2 parts, for training and testing purposes in a ratio of 90:10, meaning 90% of the data is utilized for training and the rest 10% is used for testing purpose.

Splitting data into train and test

```
#splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1, stratify=y)
```

Figure 12: Splitting data to test and train

3.4 Model Building:

3.5.1 Machine Learning Model

3.5.1.1 AdaBoost Classifier

Adaboost Classifier is invoked to fit the training dataset in order to perform intrusion detecting prediction using the training data to identify malign and clean data. The accuracy demonstrated by the Adaboost Classifier is 65%.

AdaBoost Classifier

```
adaboost_model = AdaBoostClassifier(n_estimators=4,)
adb = adaboost_model
adb.fit(X_train,y_train)
y_pred = adb.predict(X_test)
acc1 = accuracy_score(y_test,y_pred)
```

```
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1,2,3,4,5]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

```
Accuracy Score:  0.649298245614035
```

Figure 13: Adaboost Classifier

3.5.1.2 XGBoost Classifier

Then the XGBoost Classifier is invoked to fit the training dataset, in order to predict intrusion detection in SCADA environment. The Accuracy demonstrated by the XGBoost Classifier is 70%, the confusion matrix is given below.

XGBoost Classifier

```
#Xgboost Classifier
xgboost_model = XGBClassifier(max_depth=2, n_estimators=2)
xg = xgboost_model
xg.fit(X_train,y_train)
y_pred = xg.predict(X_test)
acc2 = accuracy_score(y_test,y_pred)
```

```
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1,2,3,4,5]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

```
Accuracy Score:  0.6956140350877194
```

Figure 14: XGBoost Classifier

3.5.2 Deep Learning Model

The deep learning capabilities are invoked by using LabelBinarizer function on the class "Label" inorder to convert the categorical values in it to binary values, Then fit_transform function is invoked for the new data to fit the label class. This is done to transform the label class into 3-dimensional array. Here the data is split into 2 parts namely train and evaluate, with 90% of data being used for training and the rest 10% for evaluating. The argmax function is used on the training dataset to find the best features from the training dataset. Then the expan_dim() is invoked through numpy to increase the size of the array for both training and evaluating dataset.

Deep Learning Models

```
class_labels = LabelBinarizer()
Y = class_labels.fit_transform(Y)
cls = len(class_labels.classes_)
class_labels.classes_
```

```
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.1, stratify=Y)
```

```
y_test_new = np.argmax(y_test,axis=1)
```

```
X_train1 = np.expand_dims(X_train,axis=2)
X_test1 = np.expand_dims(X_test,axis=2)
```

```
X_test1.shape
```

Figure 15: Invoking Deep Learning Capabilities

3.5.2.1 GRU + LSTM Model

All necessary libraries are imported for the GRU + LSTM, The "softmax" function is invoked in order to normalize the data.

GRU + LSTM

```
model=Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(X_train1.shape[1],1)))
model.add(GRU(64, input_shape=(X_train1.shape[1],1)))
model.add(Dense(32))
model.add(Dropout(0.4))
model.add(Dense(cls))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 31, 64)            16896

 gru (GRU)                   (None, 64)                24960

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 6)                 198

 activation (Activation)     (None, 6)                 0

=================================================================
Total params: 44134 (172.40 KB)
Trainable params: 44134 (172.40 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 16: Invoking GRU + LSTM

The GRU + LSTM model is being trained.

```
history = model.fit(X_train1,y_train,batch_size=64,epochs=10,verbose=1, validation_data=(X_test1, y_test))
```

```
Epoch 1/10
802/802 [==============================] - 18s 10ms/step - loss: 1.2020 - accuracy: 0.4736 - val_loss: 0.9850 - val_accuracy: 0.5596
Epoch 2/10
802/802 [==============================] - 6s 8ms/step - loss: 0.8345 - accuracy: 0.6448 - val_loss: 0.7585 - val_accuracy: 0.6647
Epoch 3/10
802/802 [==============================] - 7s 9ms/step - loss: 0.7433 - accuracy: 0.6798 - val_loss: 0.6744 - val_accuracy: 0.7016
Epoch 4/10
802/802 [==============================] - 6s 7ms/step - loss: 0.6992 - accuracy: 0.6984 - val_loss: 0.6422 - val_accuracy: 0.7121
Epoch 5/10
802/802 [==============================] - 6s 8ms/step - loss: 0.6665 - accuracy: 0.7073 - val_loss: 0.6054 - val_accuracy: 0.7205
Epoch 6/10
802/802 [==============================] - 7s 8ms/step - loss: 0.6408 - accuracy: 0.7171 - val_loss: 0.6320 - val_accuracy: 0.7089
Epoch 7/10
802/802 [==============================] - 6s 7ms/step - loss: 0.6189 - accuracy: 0.7278 - val_loss: 0.5913 - val_accuracy: 0.7337
Epoch 8/10
802/802 [==============================] - 8s 9ms/step - loss: 0.6075 - accuracy: 0.7309 - val_loss: 0.5615 - val_accuracy: 0.7402
Epoch 9/10
802/802 [==============================] - 7s 8ms/step - loss: 0.5991 - accuracy: 0.7388 - val_loss: 0.5886 - val_accuracy: 0.7275
Epoch 10/10
802/802 [==============================] - 8s 10ms/step - loss: 0.5812 - accuracy: 0.7467 - val_loss: 0.5614 - val_accuracy: 0.7468
```

Figure 17: Training the model

After sufficient training, the model in now made to predict result based on the training data. After successful compilation, the model now has the capability to predict new data.

```
[38] pred = model.predict(X_test1)
     y_pred = np.argmax(pred,axis=1)
     y_test_new = np.argmax(y_test,axis=1)
     acc3 = accuracy_score(y_test_new,y_pred)

     179/179 [==============================] - 1s 3ms/step
```

Figure 18: Prediction

The accuracy demonstrated by the GRU + LSTM model is 75%. Based on the results generated a confusion matrix is developed.

3.5.2.1 GRU + BILSTM

All necessary libraries are imported for the GRU + LSTM, The softmax function is invoked in order to normalize the data.

GRU + BILSTM

```
model = Sequential()
model.add(GRU(256, return_sequences=True, input_shape=(X_train1.shape[1],1)))
model.add(Bidirectional(LSTM(256, return_sequences=True), input_shape=(X_train1.shape[1],1)))
model.add(Dense(128, activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(cls, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_1 (GRU)                 (None, 31, 256)           198912

 bidirectional (Bidirection  (None, 31, 512)           1050624
 al)

 dense_2 (Dense)             (None, 31, 128)           65664

 flatten (Flatten)           (None, 3968)              0

 dense_3 (Dense)             (None, 64)                254016

 dropout_1 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 6)                 390

=================================================================
Total params: 1569606 (5.99 MB)
Trainable params: 1569606 (5.99 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 19: Invoking GRU + BILSTM

The GRU + BILSTM model is trained using the training data.

```
[43] history = model.fit(X_train1,y_train,batch_size=64,epochs=10,verbose=1, validation_data=(X_test1, y_test))

Epoch 1/10
802/802 [============================] - 19s 15ms/step - loss: 1.0077 - accuracy: 0.5619 - val_loss: 0.6904 - val_accuracy: 0.7051
Epoch 2/10
802/802 [============================] - 10s 13ms/step - loss: 0.7269 - accuracy: 0.6880 - val_loss: 0.5757 - val_accuracy: 0.7521
Epoch 3/10
802/802 [============================] - 11s 14ms/step - loss: 0.6497 - accuracy: 0.7208 - val_loss: 0.5590 - val_accuracy: 0.7449
Epoch 4/10
802/802 [============================] - 11s 13ms/step - loss: 0.6016 - accuracy: 0.7451 - val_loss: 0.3676 - val_accuracy: 0.8544
Epoch 5/10
802/802 [============================] - 11s 13ms/step - loss: 0.3907 - accuracy: 0.8460 - val_loss: 0.3131 - val_accuracy: 0.8765
Epoch 6/10
802/802 [============================] - 11s 14ms/step - loss: 0.5163 - accuracy: 0.7750 - val_loss: 0.4447 - val_accuracy: 0.7958
Epoch 7/10
802/802 [============================] - 11s 14ms/step - loss: 0.4786 - accuracy: 0.7896 - val_loss: 0.4250 - val_accuracy: 0.8111
Epoch 8/10
802/802 [============================] - 11s 13ms/step - loss: 0.4328 - accuracy: 0.8163 - val_loss: 0.3577 - val_accuracy: 0.8453
Epoch 9/10
802/802 [============================] - 11s 14ms/step - loss: 0.3660 - accuracy: 0.8498 - val_loss: 0.3269 - val_accuracy: 0.8681
Epoch 10/10
802/802 [============================] - 11s 14ms/step - loss: 0.2892 - accuracy: 0.8787 - val_loss: 0.2511 - val_accuracy: 0.8937
```

Figure 20: training the GRU+BILSTM model

After sufficient training, the model in now made to predict result based on the training data. After successful compilation, the model now has the capability to predict new data.

```
pred = model.predict(X_test1)
y_pred = np.argmax(pred,axis=1)
y_test_new = np.argmax(y_test,axis=1)
acc4 = accuracy_score(y_test_new,y_pred)

179/179 [==============================] - 2s 6ms/step
```

Figure 21: Prediction

4 Comparison of Used Models

Model Name	Accuracy
AdaBoost Classifier	65%
XGBoost Classifier	70%
GRU + LSTM	75%
GRU + BILSTM	89%

```
[48] sns.barplot(x=['AdaBoost', 'XGBoost', 'GRULSTM', 'GRUBILSTM'], y=[acc1,acc2,acc3,acc4])
```

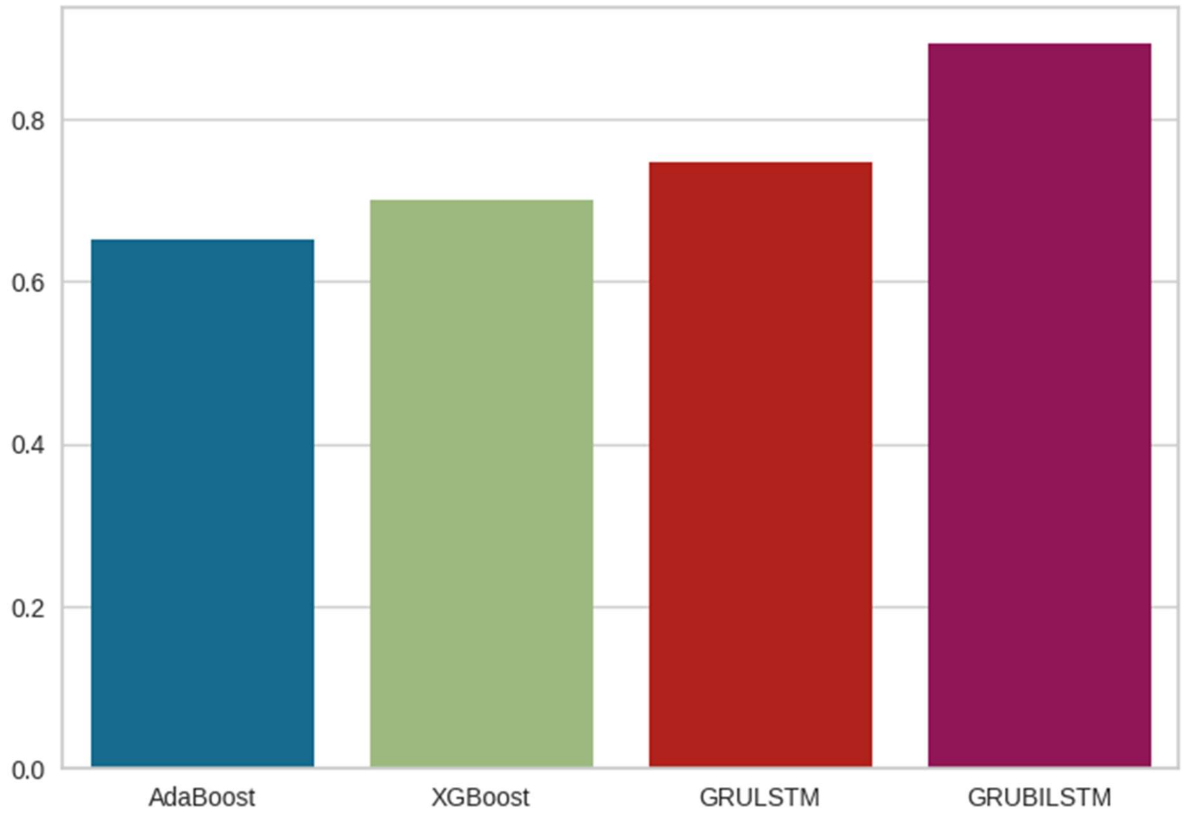

Figure 22: Comparison of Used Models

5 Conclusion

For intrusion detection in SCADA, a combination a 4 algorithm namely AdaBoost Classifier, XGBoost Classifier, GRU + LSTM, GRU + BILSTM were used, and out of all the GRU BISTM has the best accuracy. This approach may be helpful in detection of anomalies and attacks in the SCADA network more precisely.

6 References

[1] Das, T., Osama Abu Hamdan, Raj Mani Shukla, Sengupta, S. and Arslan, E. (2023). UNR-IDD: Intrusion Detection Dataset using Network Port Statistics. *OPAL (Open@LaTrobe) (La Trobe University)*. doi:https://doi.org/10.1109/ccnc51644.2023.10059640.