Configuration Manual

MSc Research Project
Data Analytics

Mayuri Bhogate
Student ID: x22220453

School of Computing
National College of Ireland

Supervisor: Mr. Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……Miss MAYURI BHOGATE…………….……………………………………………….
Student ID:	………………X22220453………………………………………………………………….……
Programme:	…………MSc Data Analytics…………………………… **Year:** ………2023-24….
Module:	…………MSc Research Project…………………………………………………………………….
Lecturer:	…………Mr. Jaswinder Singh………………………………………………………………………….
Submission Due Date:	……………12/08/2024…………………………………………………………………….………
Project Title:	A Deep Learning approach for Cyber Threat Detection using Intrusion Detection Data – Configuration Manual
Word Count:	………xxx……… **Page Count: 12**

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: ………………Mayuri Bhogate…………………………………………………………………

Date: ………………11/08/2024………………………………………………………………………………

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):	

A Deep Learning approach for Cyber Threat Detection using Intrusion Detection Data - Configuration Manual

Mayuri Bhogate
Student ID: x22220453

1 Introduction

This configuration manual specifically outlines the presentation of the environment and the way the project should be executed. The proposed project includes CNN, LSTM, BiLSTM and Stacked BiLSTM with Self-Attention Mechanism for the classification of cyber threats using a balanced Kaggle dataset.

1.1 Project Objective

The general aim of this research project is to improve the efficiency of identifying cyber threats by improving the existing deep learning models. Due to the rising level of cyber-attacks, the use of conventional approaches to detection of the threats is null and void. The idea of this proposal is to implement and assess several deep learning methodologies, CNN, LSTM, BiLSTM, and potentially S-BiLSTM with Self-Attention for the identification of cyber threats.

2 Configuration Setup

The configuration setup includes both hardware and software requirements for the research project.

2.1 Hardware Configuration

RAM: 16 GB
Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz   2.42 GHz
System: x64-based processor
Operating System: Windows 11 Home (Microsoft Corporation)
GPU: Google Colab (Tesla T4)

2.2 Software Configuration

The software configuration for this research project is done using Google Colab and Jupyter Notebook for the deep learning models. The coding is done in Python, and the version used is 3.10.12. Several Python libraries have been utilized for this research project, including:

- TensorFlow
- Keras
- Matplotlib
- Scikit-learn
- Pandas
- NumPy
- Seaborn

- Plotly

3 Data Gathering

The dataset used in this research is reclaimed from Kaggle. The dataset applied is a fusion of CSE-CIC-IDS2018-AWS, CICIDS2017, and DoS dataset from CIC2016. Such datasets include the labelled data for network traffic with both normal and attack circumstances.

3.1 Downloading the Dataset

The dataset is downloaded and unzipped using the following commands:

```
[ ] '''
    ### to clone dataset from kaggle-------
    ! pip install -q kaggle
    from google.colab import files
    files.upload()
    !mkdir ~/.kaggle
    ! cp kaggle.json ~/.kaggle/
    ! chmod 600 ~/.kaggle/kaggle.json
    !kaggle datasets list

    !kaggle datasets download -d devendra416/ddos-datasets
    '''

    '\n### to clone dataset from kaggle-------\n! pip install -q kaggle\nfrom google.colab import files\nfiles.upload(
    atasets download -d devendra416/ddos-datasets\n'

[ ] #!unzip /content/ddos-datasets.zip -d /content
    #!cp /content/ddos_balanced/final_dataset.csv /content/drive/MyDrive/network_intrusion_detection_cicids/Data

[ ] #!cp /content/ddos_balanced/final_dataset.csv /content/drive/MyDrive/network_intrusion_detection_cicids/Data
```

Fig 1. Steps to Download dataset

3.2 Loading and Preparing the Dataset

Due to possible computational complexity issues, 100,000 rows of benign data as well as 100,000 rows of DDoS attack data are selected, merged using the. concat method, and then converted into a new file in the form of a CSV format.

```
class1_df = pd.read_csv('/content/drive/My Drive/network_intrusion_detection_cicids/Data/final_dataset.csv',nrows=99999)
col = class1_df.columns.tolist()
class1_df.head(5)

class2_df = pd.read_csv('/content/drive/My Drive/network_intrusion_detection_cicids/Data/final_dataset.csv',skiprows=12694627,nrows=99999)
class2_df.columns = col
class2_df.head(5)

dataframe = pd.concat([class1_df, class2_df])
dataframe.head(5)

[ ] dataframe_saved = pd.concat([class1_df, class2_df])
    #dataframe_saved.to_csv('/content/drive/My Drive/network_intrusion_detection_cicids/dataset.csv')
```

Fig 2. Steps to concatenate dataset

4 Data Transformation

As a result, data transformation is a few processes that must be completed before the training of a model using the dataset. The following steps are carried out for Data Cleaning and management of missing values

```
[19] #Columns that have only one value
     colsToDrop = np.array(['Fwd Byts/b Avg', 'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd Pkts/b Avg', 'Bwd Blk Rate Avg'])

[20] #Drop rows where a column missing values are no more than 5% & Drop columns where missing values are more than 50%
     missing = dataframe.isna().sum()
     missing = pd.DataFrame({'count': missing, '% of total': missing/len(dataframe)*100}, index=dataframe.columns)
     colsToDrop = np.union1d(colsToDrop, missing[missing['% of total'] >= 50].index.values)
     dropnaCols = missing[(missing['% of total'] > 0) & (missing['% of total'] <= 5)].index.values

[21] #Handling faulty data.
     dataframe['Flow Byts/s'].replace(np.inf, np.nan, inplace=True)
     dataframe['Flow Pkts/s'].replace(np.inf, np.nan, inplace=True)
     dropnaCols = np.union1d(dropnaCols, ['Flow Byts/s', 'Flow Pkts/s'])
     colsToDrop
```

Fig 3. Steps to Clean data and handle missing values

5 Data Preprocessing and Splitting

The categorical label column is encoded subsequently, and features normalization is also conducted using the MinMaxScaler. After that, the dataset is divided to training, testing and validation set with the percentages of 60%, 20% and 20% respectively. This further helps in formatting the data correctly so that it can be used adequately for training and even testing the machine learning models.

```
[ ] cat_col = dataframe.select_dtypes(include=['object']).columns.tolist()
    cat_col

    ['Label']

[ ] le = LabelEncoder()
    for i in cat_col:
      dataframe[i] = le.fit_transform(dataframe[i])

    X = dataframe.drop(['Label'], axis='columns')
    y = dataframe['Label']

[ ] #applying minmax scaler for data normalization
    scaler = MinMaxScaler()
    X_s = scaler.fit_transform(X)
    X_s = pd.DataFrame(X_s, columns=X.columns)
    X_s.head()

[ ] #splitting dataset into train, test, val with ratio of 60:20:20
    X_train, X_test, y_train, y_test = train_test_split(X_s, y, test_size=0.2, stratify=y)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)
```

Fig 4. Steps for Data Preprocessing and Splitting

6 Data Visualization

The following code is used to view the target classes and analyze the correlation of the features by heatmap, and find the 15 features most related to the model's output by feature importance plot.

Fig 5. Steps for Data Visualization

7 Model Implementation

7.1 CNN

Below code is used for the CNN model deployment

```
model = Sequential()
model.add(Conv1D(8, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(Flatten())
model.add(Dense(6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dropout(0.45))
model.add(Dense(2, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.00001), metrics=['accuracy'])
model.summary()
```

Fig6. Steps to implement CNN model

Model is evaluated using below measures

1. Accuracy :

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred,axis=1)
print ("CNN Accuracy: ", accuracy_score(y_test_org,y_pred))

1245/1245 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
CNN Accuracy:  0.584416888398012
```

Fig 7. Steps to calculate Accuracy: CNN

```
#Train and Validation Accuracy
plt.plot(history.history['accuracy'], 'g', marker='*', label='Train Accuracy')
plt.plot(history.history['val_accuracy'], 'b', marker='*', label='Val Accuracy')
plt.title('Train and Val Accuracy')
plt.legend()
plt.figure()
#Train and Validation Loss
plt.plot(history.history['loss'], 'g', marker='*', label='Train Loss')
plt.plot(history.history['val_loss'], 'b', marker='*', label='Val Loss')
plt.title('Train and Val Loss')
plt.legend()
plt.show()
```

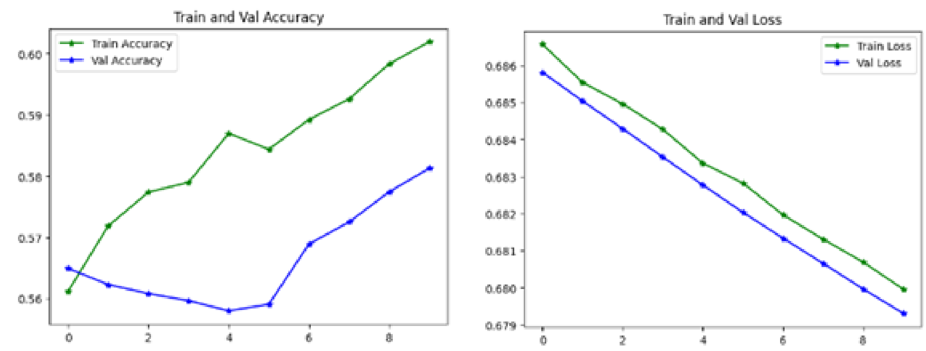

Fig8. Steps to plot Train and Val accuracy : CNN

2. Confusion Matrix

Fig 9. Steps to calculate confusion Matrix: CNN

3. Classification Report

```
#Classification Report
print("Classification Report : ")
print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.55      0.94      0.69     19844
           1       0.80      0.23      0.36     19994

    accuracy                           0.58     39838
   macro avg       0.67      0.59      0.52     39838
weighted avg       0.67      0.58      0.52     39838
```

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(2):
  prec, recall, _, _ = precision_recall_fscore_support(y_test_org==l,
                                                       y_pred==l,
                                                       pos_label=True, average=None)
  sensitivity = recall[0] if recall.size > 0 else 0
  specificity = recall[1] if recall.size > 1 else 0
  res.append([l, sensitivity, specificity])
pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Fig10. Steps for Classification Report: CNN

4. Specificity and Sensitivity

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(2):
  prec, recall, _, _ = precision_recall_fscore_support(y_test_org==l,
                                                       y_pred==l,
                                                       pos_label=True, average=None)
  sensitivity = recall[0] if recall.size > 0 else 0
  specificity = recall[1] if recall.size > 1 else 0
  res.append([l, sensitivity, specificity])
pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.229569	0.941947
1	1	0.941947	0.229569

Fig11. Steps for Specificity and Sensitivity: CNN

7.2 LSTM

Below code is used for LSTM model Deployment

```
model=Sequential()
model.add(LSTM(12, return_sequences=False, input_shape=(X_train.shape[1],1)))
model.add(Dense(units=2))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=0.0001),metrics=['accuracy'])
model.summary()

Model: "sequential_1"
```

Fig12. Steps to implement LSTM model

Model is evaluated using below measures

1. Accuracy

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred,axis=1)
print ("LSTM Accuracy: ", accuracy_score(y_test_org,y_pred))

1245/1245 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
LSTM Accuracy:  0.5479692755660425
```

Fig 13. Steps to calculate Accuracy: LSTM

```
#Train and Validation Accuracy
plt.plot(history.history['accuracy'], 'g', marker='*', label='Train Accuracy')
plt.plot(history.history['val_accuracy'], 'b', marker='*', label='Val Accuracy')
plt.title('Train and Val Accuracy')
plt.legend()
plt.figure()
#Train and Validation Loss
plt.plot(history.history['loss'], 'g', marker='*', label='Train Loss')
plt.plot(history.history['val_loss'], 'b', marker='*', label='Val Loss')
plt.title('Train and Val Loss')
plt.legend()
plt.show()
```

Fig14. Steps to plot Train and Val accuracy: LSTM

2. Confusion Matrix

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test_org, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```

Fig15. Steps for confusion matrix: LSTM

3. Classification Report

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))

    Classification Report :
                  precision    recall  f1-score   support

               0       0.61      0.26      0.37     19844
               1       0.53      0.83      0.65     19994

        accuracy                           0.55     39838
       macro avg       0.57      0.55      0.51     39838
    weighted avg       0.57      0.55      0.51     39838
```

Fig16. Steps Classification Report: LSTM

4. Specificity and Sensitivity

```
[ ] from sklearn.metrics import precision_recall_fscore_support
    res = []
    for l in range(2):
      prec, recall, _, _ = precision_recall_fscore_support(y_test_org==l,
                                                           y_pred==l,
                                                           pos_label=True, average=None)
      sensitivity = recall[0] if recall.size > 0 else 0
      specificity = recall[1] if recall.size > 1 else 0
      res.append([l, sensitivity, specificity])
    pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.830999	0.262800
1	1	0.262800	0.830999

Fig17. Steps for Specificity and Sensitivity: LSTM

7.3 BiLSTM

Below code is used for LSTM model Deployment

```
model=Sequential()
model.add(Bidirectional(LSTM(5,return_sequences=False,input_shape=(X_train.shape[1],1))))
model.add(Dense(units=2))
model.add(Activation('sigmoid'))
model.compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
model.summary()
```

Fig18. Steps to implement BiLSTM model

Model is evaluated using below measures

1. Accuracy

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred,axis=1)
print ("BILSTM Accuracy: ", accuracy_score(y_test_org,y_pred))
```

```
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step
BILSTM Accuracy:  0.9615693558913601
```

Fig 19. Steps to calculate Accuracy: BiLSTM

```
#Train and Validation Accuracy
plt.plot(history.history['accuracy'], 'g', marker='*', label='Train Accuracy')
plt.plot(history.history['val_accuracy'], 'b', marker='*', label='Val Accuracy')
plt.title('Train and Val Accuracy')
plt.legend()
plt.figure()
#Train and Validation Loss
plt.plot(history.history['loss'], 'g', marker='*', label='Train Loss')
plt.plot(history.history['val_loss'], 'b', marker='*', label='Val Loss')
plt.title('Train and Val Loss')
plt.legend()
plt.show()
```

Fig20. Steps to plot Train and Val accuracy: BiLSTM

2. Confusion Matrix

```
#confusion Matrix
matrix=confusion_matrix(y_test_org, y_pred)
fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
fig.show()
```

Fig21. Steps to plot Train and Val accuracy: LSTM

3. Classification Report

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))

    Classification Report :
                  precision    recall  f1-score   support

               0       0.98      0.94      0.96     19844
               1       0.94      0.98      0.96     19994

        accuracy                           0.96     39838
       macro avg       0.96      0.96      0.96     39838
    weighted avg       0.96      0.96      0.96     39838
```

Fig22. Steps for Classification Report: BiLSTM

4. Specificity and Sensitivity

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(2):
  prec, recall, _, _ = precision_recall_fscore_support(y_test_org==l,
                                                       y_pred==l,
                                                       pos_label=True, average=None)
  sensitivity = recall[0] if recall.size > 0 else 0
  specificity = recall[1] if recall.size > 1 else 0
  res.append([l, sensitivity, specificity])
pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.981344	0.941645
1	1	0.941645	0.981344

Fig23. Steps for Sensitivity and Specificity Report: BiLSTM

7.4 STACKED BILSTM WITH SELF ATTENTION MECHANISM

Below code is used for Stacked BiLSTM with Self attention mechanism model Deployment

```
class Attention(Layer):
  def __init__(self,**kwargs):
    super(Attention,self).__init__(**kwargs)

  def build(self,input_shape):
    self.W=self.add_weight(name="att_weight",shape=(input_shape[-1],1),initializer="normal")
    self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1),initializer="zeros")
    super(Attention, self).build(input_shape)

  def call(self,x):
    et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
    at=K.softmax(et)
    at=K.expand_dims(at,axis=-1)
    output=x*at
    return K.sum(output,axis=1)

  def compute_output_shape(self,input_shape):
    return (input_shape[0],input_shape[-1])

  def get_config(self):
    return super(Attention,self).get_config()
```

Fig24. Steps for Self-Attention Mechanism

```
model= Sequential()
model.add(tf.keras.layers.Bidirectional(LSTM(12,return_sequences=True,input_shape=(X_train.shape[1],1))))
model.add(tf.keras.layers.Bidirectional(LSTM(12,return_sequences=True)))
model.add(Attention())
model.add(Dense(10,activation='relu'))
model.add(Flatten())
model.add(Dense(6,activation='relu'))
model.add(Dropout(0.55))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
model.summary()
```

Fig25. Steps to implement Stacked BiLSTM with self-attention mechanisms

Model is evaluated using below measures

1. Accuracy

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred,axis=1)
print ("STACKED BILSTM WITH SELF ATTENTION Accuracy: ", accuracy_score(y_test_org,y_pred))

1245/1245 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step
STACKED BILSTM WITH SELF ATTENTION Accuracy:  0.9839098348310659
```

Fig26. Steps to calculate accuracy: Stacked BiLSTM with self-attention mechanisms

```
#Train and Validation Accuracy
plt.plot(history.history['accuracy'], 'g', marker='*', label='Train Accuracy')
plt.plot(history.history['val_accuracy'], 'b', marker='*', label='Val Accuracy')
plt.title('Train and Val Accuracy')
plt.legend()
plt.figure()
#Train and Validation Loss
plt.plot(history.history['loss'], 'g', marker='*', label='Train Loss')
plt.plot(history.history['val_loss'], 'b', marker='*', label='Val Loss')
plt.title('Train and Val Loss')
plt.legend()
plt.show()
```

Fig27. Steps to plot Train and Val accuracy: Stacked BiLSTM with self-attention mechanisms

2. Confusion Matrix

```
#confusion Matrix
matrix=confusion_matrix(y_test_org, y_pred)
fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
fig.show()
```

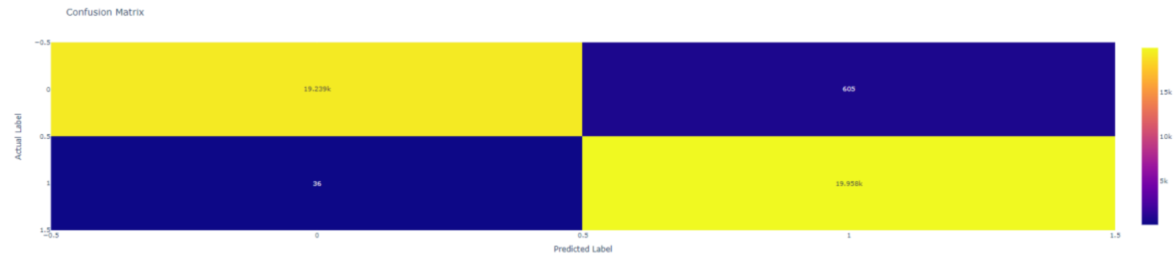

Fig28. Steps to plot Confusion Matrix: Stacked BiLSTM with self-attention mechanisms

3. Classification Report

```
#Classification Report
print("Classification Report : ")
print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     19844
           1       0.97      1.00      0.98     19994

    accuracy                           0.98     39838
   macro avg       0.98      0.98      0.98     39838
weighted avg       0.98      0.98      0.98     39838
```

Fig14. Steps to plot Classification Report: Stacked BiLSTM with self-attention mechanisms

4. Specificity and Sensitivity

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(2):
  prec, recall, _, _ = precision_recall_fscore_support(y_test_org==l,
                                                     y_pred==l,
                                                     pos_label=True, average=None)
  sensitivity = recall[0] if recall.size > 0 else 0
  specificity = recall[1] if recall.size > 1 else 0
  res.append([l, sensitivity, specificity])
pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.998199	0.969512
1	1	0.969512	0.998199

Fig14. Steps to calculate Specificity and Sensitivity: Stacked BiLSTM with self-attention mechanisms