Configuration Manual

MSc Research Project
MS in Cybersecurity

Ranjith Kumar Saravanan
Student ID: X22209751

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ranjith Kumar Saravana
Student ID:	X22209751
Programme:	MS in Cybersecurity **Year:** 2023-2024
Module:	Research Practicum
Lecturer:	Khadija Hafeez
Submission Due Date:	19/08/2024
Project Title:	Optimizing Network Security: Performance Analysis of Neural Network Models for Intrusion Detection

Word Count: 415 **Page Count:** 8

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: Ranjith Kumar Saravana

Date: 19/08/2024

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):	

Configuration Manual

Ranjith Kumar Saravanan
Student ID: X22209751

1 Introduction

This configuration manual can be used to run the code of the model. It has all the steps mentioned to implement the model. This project is based on machine learning where two algorithms have been used, the ANN algorithm and LTSM for intrusion detection.

2 Requirement of Hardware

Operating system: Windows 11 home (64 bit)
RAM: 8GB/16 GB
Storage: 1TB HDD or SSD
Processor: 11th gen - Intel core i7 @ 2.80GHz 2.80 GHz
System type: 64-bit operating system (x64-based processor).

3 Requirement of Software

The following are the software that must be installed to execute the project.

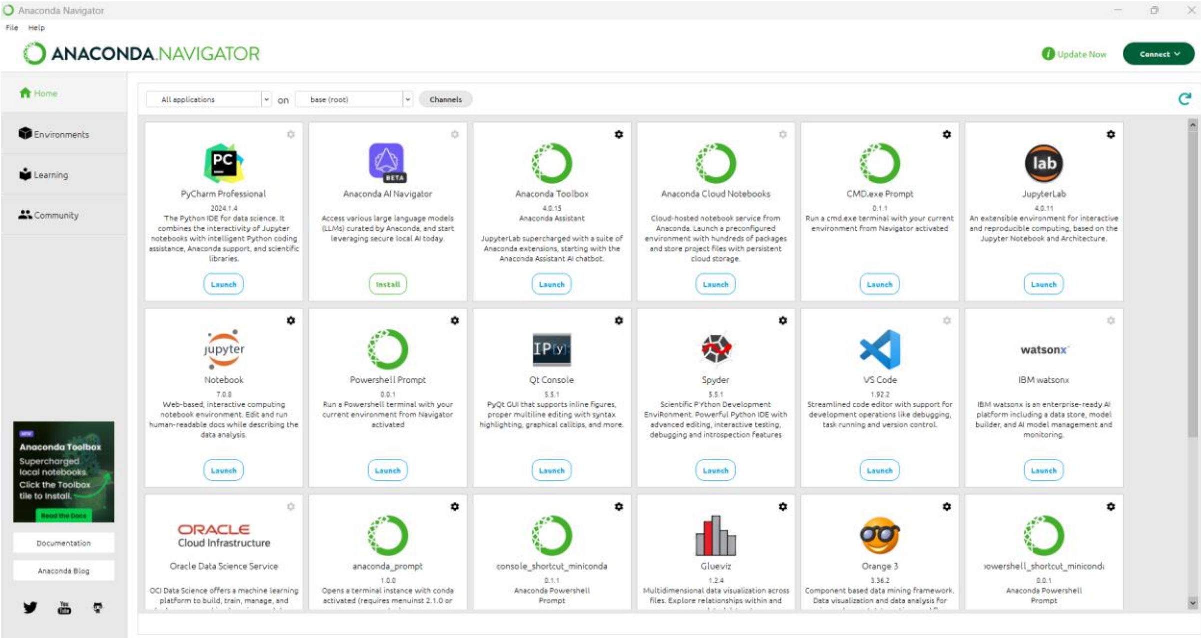
Software	Version
Anaconda	24.5.0
Python	3.7

Anaconda distributed gives a platform to perform AI/ML projects. It has the capability to handle large dataset and process them.

Jupyter Notebook is used for running the python code as it is an IDE, the python libraries can be used in this the visualization and graphs are also supported in this.

Download Anaconda Distribution | Anaconda

Once the Anaconda is installed, the anaconda navigator should be opened and under that Jupyter notebook option will be there it will open the IDE where the notebook can be open or created it comes with pre-installed libraries and python.



4 Pre-requisite

Following libraries has to be installed for this project:

- import pandas as pd
- from pathlib import Path
- import numpy as np
- import matplotlib.pyplot as plt
- import pickle
- import yagmail

```
import warnings
warnings.filterwarnings("ignore")

import os
import pandas as pd
from pathlib import Path
pd.set_option("display.max_columns", None)
import numpy as np
import matplotlib.pyplot as plt
from lib_file import lib_path
%matplotlib inline

import random
import seaborn as sns
import pickle
from sklearn.utils import resample
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

5 DataSet

In this project NSL-KDD dataset has been used, it has been taken from Kaggle it has about 41 features and 125,973 records. Each of the data have a label which marks them weather they are normal or malicious. This data is good for intrusion detection system as it has all the information about the network.

https://www.kaggle.com/datasets/hassan06/nslkdd

6 Data Loading and Preprocessing (1_IDS_Final_Preprocessing_File.ipynb)

Data importing

Data Loading

```
[ ] df = pd.read_csv("concatenated_file.csv")
    df.shape
```

```
(148517, 42)
```

Data Preprocessing

Data Preprocessing

```
[ ] df.head(10)
```

	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
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0
5	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0
6	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	tcp	remote_job	S0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0

Converting to string

```
[ ] # Check data type of 'labels' column
    print(df['labels'].dtype)

    # Convert to string if necessary
    df['labels'] = df['labels'].astype(str)
```

Resampling the data

```
[ ] label_counts = df['labels'].value_counts()

    # Step 2: Define labels to keep
    labels_to_keep = label_counts[label_counts >= 1000].index

    # Step 3: Filter data to keep only the selected labels
    cleaned_data = df[df['labels'].isin(labels_to_keep)]

    # Step 4: Resample each label to have exactly 5000 samples
    resampled_data = []
    for label in labels_to_keep:
        label_data = cleaned_data[cleaned_data['labels'] == label]
        if len(label_data) < 5000:
            # Upsample if there are fewer than 5000 samples
            resampled_label_data = resample(label_data, n_samples=5000, replace=True)
        else:
            # Downsample if there are more than 5000 samples
            resampled_label_data = resample(label_data, n_samples=5000)

        resampled_data.append(resampled_label_data)

    # Combine the resampled data
    final_data = pd.concat(resampled_data, ignore_index=True)

    # Print the shapes and counts
    print("Filtered data shape:", final_data.shape)
    print("Label counts:\n", final_data['labels'].value_counts())
```

Spliting the dataset

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, stratify=y)
    print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

    (32000, 41) (32000, 1) (8000, 41) (8000, 1)
```

7 Model Training and Testing(2_IDS_Final_ModelTraining.ipynb)

Algorithm: ArtificialNeuralNetwork

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, Input
from tensorflow.keras.regularizers import L2
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping
```

```
model = Sequential()

model.add(Input(shape=(X_train.shape[1],)))

model.add(Dense(units=128, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=512, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=8, activation='sigmoid'))

# Adding optimizer
optimizer = Adam(learning_rate=0.001)
```

Result Analysis

```
ANN_accuracy = accuracy_score(
    y_true=true_labels,
    y_pred=predicted_labels
)

print(f"Validation accuracy of ArtificialNeuralNetwork model is {ANN_accuracy*100:.2f}%")
```

```
Validation accuracy of ArtificialNeuralNetwork model is 66.70%
```

Algorithm:2 LongShortTermMemory

```
# x_train=np.reshape(a=X_train.values,newshape=(X_train.shape[0],X_train.shape[1],1))
# x_test=np.reshape(a=X_test.values,newshape=(X_test.shape[0],X_test.shape[1],1))

time_steps = 1
num_features = X_train.shape[1]  # 41

X_train_reshaped = np.reshape(X_train.values, (X_train.shape[0], time_steps, num_features))
X_test_reshaped = np.reshape(X_test.values, (X_test.shape[0], time_steps, num_features))

print(X_train_reshaped.shape, X_test_reshaped.shape)  # Should print (86845, 1, 41) and (21712, 1, 41)
```

```
[ ] y_train_np = y_train.values.flatten()
    y_test_np = y_test.values.flatten()
    num_classes = len(np.unique(y_train_np))
    num_classes
    print("Unique values in y_train_np:", np.unique(y_train_np))
    print("Unique values in y_test_np:", np.unique(y_test_np))

    num_classes = len(np.unique(y_train_np))

    y_train_one_hot = to_categorical(y_train_np, num_classes=num_classes)
    y_test_one_hot = to_categorical(y_test_np, num_classes=num_classes)

    print(y_train_one_hot.shape, y_test_one_hot.shape)
```

```
Unique values in y_train_np: [0 1 2 3 4 5 6 7]
Unique values in y_test_np: [0 1 2 3 4 5 6 7]
(32000, 8) (8000, 8)
```

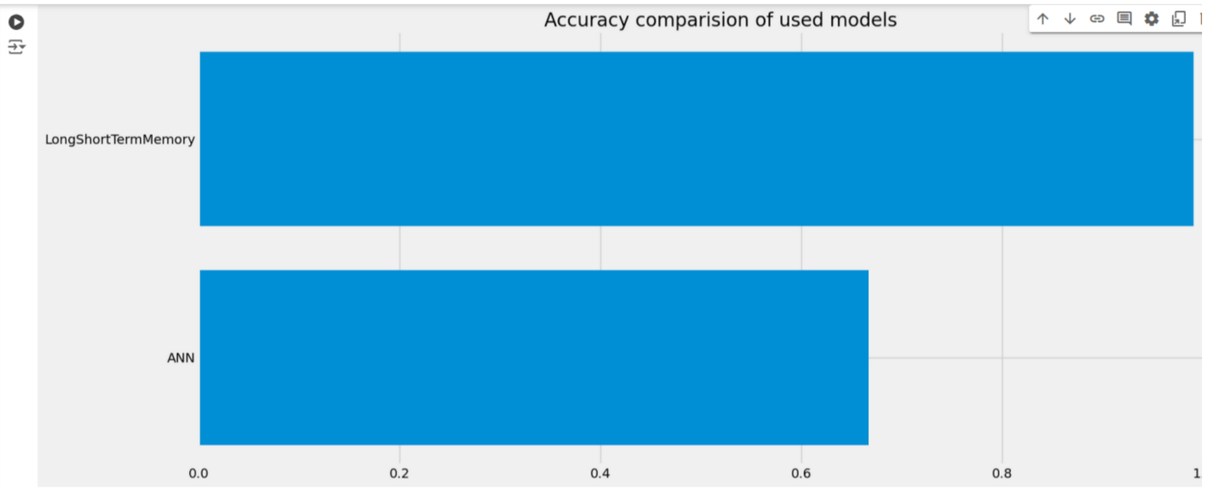
Result Analysis

Accuracy Score

```
[ ] lstm_accuracy=accuracy_score(y_true=y_true,y_pred=lstm_pred)
    print("LSTM model accuracy is {:.2f}%".format(lstm_accuracy*100.0))
```

```
LSTM model accuracy is 99.10%
```

Comparision of both the models



8 Inference File (3_Final_TestFile-checkpoint.ipynb)

- Real time Intrusion detection: The LSTM model has been implemented on the inference system for the monitoring of network traffic.
- Treat response: The system first checks the IP is present in the block list if not it immediately adds the IP to the block list.
- Better Accuracy: With LSTM model the system can be able to protect the network more accurately and efficiently.

Load trained model

```
[ ] # Load the LSTM model
    model = load_model('models/LongShortTermMemory_model.h5', compile=False)
```

Checking if the IP is present in the Blocklist

```
def phase_1_verification(filepath):
    df = pd.read_csv(filepath)
    ip_df = pd.read_csv("Block_IP_List.csv")
    input_ip_address = df.pop('ip_address').values[0].strip()

    if input_ip_address in ip_df['IP Address'].values.tolist():
        history_attack = ip_df.loc[ip_df['IP Address'] == input_ip_address]['Found Attack'].values[0]

        return {"STATUS": True, "IP ADDRESS": input_ip_address, "ATTACK": history_attack}
    else:
        return {"STATUS": False}
```

Model prediction

```
if result["STATUS"] == False:
    # Reshape input data to match the expected shape of the LSTM model
    input_data_reshaped = np.expand_dims(input_data.values, axis=1)

    # Make predictions with the reshaped data
    prediction = model.predict(input_data_reshaped)
    prediction = np.argmax(prediction, axis=1)


    ClassIndex = prediction[0]
    ClassLabel = class_labels[ClassIndex]
    if ClassLabel != 'normal':
        print(f'Model predicted class is: {ClassIndex}')
        print(f'Model predicted label is: {ClassLabel}')
        blockIP = f"{df['ip_address'][0]} IP Address is added in Block List."
        ip = df['ip_address'].tolist()
        print(blockIP)
    else:
        print(f'Model predicted class is: {ClassIndex}')
        print(f'Model predicted label is: {ClassLabel}')
        blockIP = f"{df['ip_address'][0]} is a Genuine IP Address."
        print(blockIP)
else:
    print("Blocked Client Found.")
```

```
Model predicted class is: 3
Model predicted label is: nmap
192.168.1.21 IP Address is added in Block List.
```

```
[ ] if result["STATUS"] == False:
        def update_logfile(ip_address=None, predicted_attack=None):
            new_data = {'IP Address': [str(ip_address).strip()],
                        'Found Attack': [predicted_attack]}
            new_row_df = pd.DataFrame(new_data)

            try:
                df = pd.read_csv("Block_IP_List.csv")
            except FileNotFoundError:
                df = pd.DataFrame(columns=['IP Address', 'Found Attack'])

            df = pd.concat([df, new_row_df], ignore_index=True)
            df.to_csv("Block_IP_List.csv", index=False)
            return True

        if ClassLabel != 'normal':
            update_logfile(ip_address=ip[0], predicted_attack=ClassLabel)
            print("It's a Attack File & IP Address added in Block List")
        else:
            print("It's a Normal File.")
    else:
        print("Blocked Client Found.")
```

```
It's a Attack File & IP Address added in Block List
```