National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Prajesh Nikhil Rajendrasubbu
Student ID: 22233784

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: PRAJESH NIKHIL RAJENDRASUBBU

Student ID: X22233784

Programme: MSCCLOUD **Year:** 2023 - 2024

Module: MSCCLOUD Research Project

Lecturer: Aqeel Kazmi

Submission Due Date: 12/08/2024

Project Title: A Comparative Analysis of Hybrid Machine Learning Techniques for Network Intrusion Detection in Cloud Environments.

Word Count: 1100 **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: PRAJESH NIKHIL RAJENDRASUBBU

Date: 12/08/2024

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

Prajesh Nikhil Rajendrasubbu
Student ID: 22233784

1 Introduction

This configuration manual is designed to help with the setup and implementation of the Network Intrusion Detection System (NIDS) which guides the setup of advanced machine learning system techniques. The main goal of the research project is to improve cloud security by using the machine learning models Random Forest (RF) and Long Short-Term Memory (LSTM) models extracting their features to create a hybrid NIDS that is capable of detecting a new and evolving threat in the cloud networks.

The system is developed using Google Colab, which is a cloud-based platform that provides the computational resources which are required for training the model and deploying the models. This manual will give detailed information on how to set up the Google Colab environment step-by-step, the dataset preprocessing, Machine learning model training, and deployment of NIDS for detection in real-time intrusion. The UNSW-NB15 dataset is used for the project which is a wide range and diverse collection of the network traffic data that includes a wide range of normal and malicious activities. The trained model now for deploying the NIDS Streamlit web application is used for the NIDS the Streamlit app allows the users to communicate with the model the network traffic data is given as the input to receive the real-time predictions of the network traffic. By following this guide, it should be able to replicate the setup and gain an understanding of the technical processes involved in developing a strong security solution for cloud environments.

2 System Requirements

1. **Hardware**:

Cloud Environment:

Google Colab was used for this project to develop and deploy the model, All the necessary computational resources which is needed for the development are provided by this cloud-based platform which also includes GPU support for deep learning models.

The requirements for this are a stable internet connection, and Google account access for using the Google Drive service where the datasets are loaded, and models will be saved.

2. **Software**:

Programming Language:

Python: Python programming language is used for this project, making use of its wide range of libraries for data processing, machine learning, and deployment.

Python Libraries:

The required Python libraries for this project are:

- Pandas: Data manipulation and preprocessing.
- NumPy: Numerical operations and handling arrays.
- Scikit-learn: Machine learning algorithms, data preprocessing, and evaluation metrics.
- TensorFlow/Keras: Building and training the LSTM model.
- Streamlit: Creating the user interface for model deployment.
- Joblib: Saving and loading trained models and preprocessing objects (like scalers and encoders).

Installation Commands:

- !pip install pandas numpy scikit-learn tensorflow streamlit joblib

Setting Up Google Colab:

For accessing the datasets and to save the models trained you are required to mount your data in Google Drive in the Colab environment. Required a Google account permissions access must be given to access the data loaded.

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

GPU Acceleration:

Enable GPU: To activate GPU acceleration in Google Colab, click the Runtime form the menu Runtime > Change runtime type, and select GPU from the Hardware Accelerator menu.

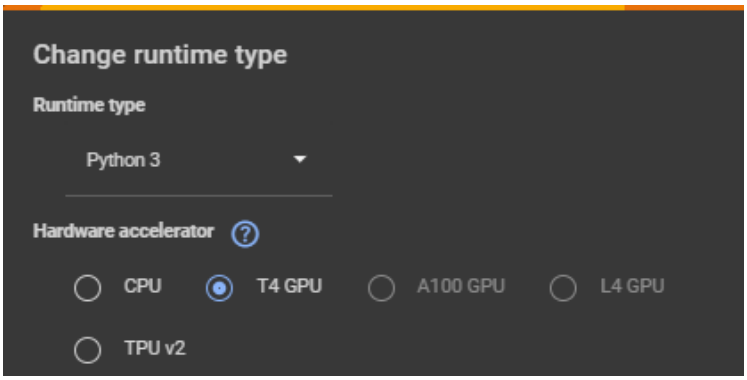

Fig.1 Google Colab environments setup.

Next, the following commands shown in the fig.2 are imported for the further processing of the model training and evaluation.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import to_categorical
import joblib
import matplotlib.pyplot as plt
```

Fig.2 Imported libraries and packages for Python

3 Data Collection

The dataset used in this project is the UNSW-NB15 dataset which is a diverse collection of network data is a licensed dataset (CC BY-NC-SA 4.0) that contains normal and malicious activities this data set is loaded into Google Drive in Parquet format which is more efficient for larger dataset size then that is used for the evaluation of the NIDS.

Dataset Link: https://www.kaggle.com/datasets/dhoogla/unswnb15
License of dataset: https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.en

Loading Data:
Use the below command to read

- import pandas as pd

```
file_path = '/content/drive/My Drive/UNSW_NB15_testing-set.parquet'
data = pd.read_parquet(file_path)
```

Fig.3 Loading the dataset into Pandas data frame form Google drive.

4 Data Preprocessing

Data Preprocessing:
Data preprocessing is a critical step to ensure that the data is in a suitable format for training machine learning models.

Handling Missing Values:

Missing values in the dataset are handled and identified in this project missing values in the numerical column are filled with the mean in the columns

- data.fillna(data.mean(), inplace=True)

Encoding Categorical Variables:
The categorical values in the data set are now here converted into numerical format using the feature called ‘LabelEncoder’ form the Python Library Scikit-learn.

- from sklearn.preprocessing import LabelEncoder

- categorical_columns = ['protocol', 'service', 'flag'] # Example categorical columns
encoders = {}

for col in categorical_columns:
    encoder = LabelEncoder()
    data[col] = encoder.fit_transform(data[col])
    encoders[col] = encoder

```
# Step 1: Load and preprocess the dataset
file_path = './UNSW_NB15_testing-set.parquet'  # Replace with your actual file path
data = pd.read_parquet(file_path)

# Remove the 'label' column
data = data.drop('label', axis=1)

# Encode categorical variables
categorical_columns = ['proto', 'service', 'state', 'attack_cat']
label_encoders = {}

for col in categorical_columns:
    label_encoders[col] = LabelEncoder()
    data[col] = label_encoders[col].fit_transform(data[col])
    np.save(f'classes_{col}.npy', label_encoders[col].classes_)

# Exclude specified categories from the dataset
exclude_categories = ['Analysis', 'Backdoor', 'Shellcode', 'Worms','DoS']
for category in exclude_categories:
    encoded_value = label_encoders['attack_cat'].transform([category])[0]
    data = data[data['attack_cat'] != encoded_value]

# Separate features and target
X = data.drop('attack_cat', axis=1)
y = data['attack_cat']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert target variable to one-hot encoded format
y_encoded = to_categorical(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)
```

Fig.4.Data preprocessing code

Standardizing Features:

All the numerical values must be ensured that there are in the same scale they have to standardised.

- from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
  data[numerical_columns] = scaler.fit_transform(data[numerical_columns])

Saving Encoders and Scalers for Deployment:

To save the label encoder values trained use `LabelEncoder` and `StandardScaler` objects using `joblib` so that they can be used when needed during the training.

- import joblib
  joblib.dump(encoders, 'encoders.pkl')
  joblib.dump(scaler, 'scaler.pkl')

Splitting the Dataset:

Data split is needed before the model training here are the code for the data split.

- from sklearn.model_selection import train_test_split
  X = data.drop(columns=['label', 'attack_cat'])
  y = data['label']
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

5 Model Development

5.1 Random Forest:

```
# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the RF model
y_pred_rf = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf, average='weighted')
rf_recall = recall_score(y_test, y_pred_rf, average='weighted')
```

Fig.5 Random Forest Model Traning

At first random forest alone is trained and accuracy, F1 score , Recal data is collected for comparion with the hybvird model. Visual Aids are added for visal comparison of the performance.

5.2 LSTM :

```
# Train the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(50, input_shape=(1, X_train.shape[1]), return_sequences=True))
lstm_model.add(LSTM(50))
lstm_model.add(Dense(y_train_categorical.shape[1], activation='softmax'))

lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
history = lstm_model.fit(X_train_3d, y_train_categorical, epochs=10, batch_size=64, validation_data=(X_test_3d, y_test_categorical), callbacks=[early_stopping])

# Evaluate the LSTM model
loss, lstm_accuracy = lstm_model.evaluate(X_test_3d, y_test_categorical)
y_pred_lstm = np.argmax(lstm_model.predict(X_test_3d), axis=1)
lstm_f1 = f1_score(y_test, y_pred_lstm, average='weighted')
lstm_recall = recall_score(y_test, y_pred_lstm, average='weighted')
```

Fig.6 LSTM Model traning.

Next, LSTM alone is trained and accuracy, F1 score , Recal data is collected for comparion with the hybvird model. Visual Aids are added for visal comparison of the performance.

5.3 Hybrid Model:

```
# Step 2: Train a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, np.argmax(y_train, axis=1))

# Save the Random Forest model
joblib.dump(rf_model, 'rf_model.pkl')

# Generate predictions from the Random Forest model
rf_train_predictions = rf_model.predict_proba(X_train)
rf_test_predictions = rf_model.predict_proba(X_test)

# Step 3: Train an LSTM model using Random Forest predictions
# Reshape the RF predictions to 3D for LSTM [samples, timesteps, features]
rf_train_predictions_3d = np.reshape(rf_train_predictions, (rf_train_predictions.shape[0], 1, rf_train_predictions.shape[1]))
rf_test_predictions_3d = np.reshape(rf_test_predictions, (rf_test_predictions.shape[0], 1, rf_test_predictions.shape[1]))

# Define the LSTM model
num_classes = y_encoded.shape[1]
lstm_model = Sequential()
lstm_model.add(Input(shape=(rf_train_predictions_3d.shape[1], rf_train_predictions_3d.shape[2])))
lstm_model.add(LSTM(100, return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(50,return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(25))
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(num_classes, activation='softmax'))

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

# Save the best model during training
checkpoint = ModelCheckpoint('hybrid_model.keras', monitor='val_accuracy', save_best_only=True, mode='max')

# Train the LSTM model
history = lstm_model.fit(rf_train_predictions_3d, y_train, epochs=10, batch_size=64, validation_split=0.2, callbacks=[checkpoint])

# Evaluate the LSTM model on the test set
loss, accuracy = lstm_model.evaluate(rf_test_predictions_3d, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

# Make predictions
y_pred = lstm_model.predict(rf_test_predictions_3d)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)
```

Fig.7 Hybrid model training combining both Random Forest and LSTM features for prediction.

The Major feature of random forest is extracted and combined with the features of LSTM here for the hybrid model accuracy, F1 score , Recal data is collected for comparion with the hybvird model. Visual Aids are added for visal comparison of the performance. Model accuracy, Model loss and confusion matix graphs and visuals are plotted.

```
# Save the Random Forest model
joblib.dump(rf_model, 'rf_model.pkl')
```

Fig.8 Saving the trained models using joblib

The model are saved using the joblib.dump command rf_model.plk files here saves the trained model here.

6. Testing and Evaluation

In the section now the model is trained and for testing the model a sample values of the dataset network data is passed to the code and the predictions are printed out wheather the passed input values are normal traffic or threat

The below figure shows the evaluated output values the first sample is a normal traffic and the second sample is of attack with the class “Fuzzers”.

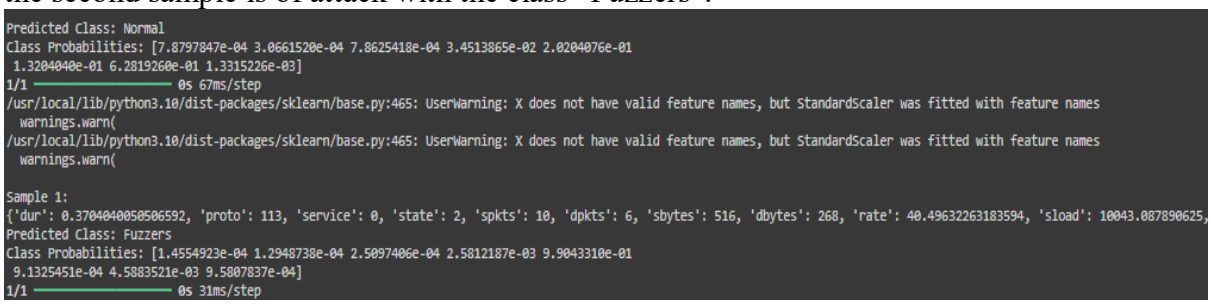

Fig.9 Evalution of the output showing the predictions results of normal traffic and sample of attack with class “Fuzzers”

7. Model Deploment of UI.

Now the trained model needed to presented with the UI for the user for the a python framework called Streamlit is used to create a UI the interface is created based on the .plk file given by the trained model to match the features in the implementation.

Command to install Streamlit

- !pip install streamlit
- !pip install -r requirements.txt

Input values field are entered here as the input for the model to predicted are shown in the figure. The Streamlit python file is created based on the reqirements where the values can be entered by the used in the boxes which as + and – features to incerase or decrease numerical values or we can manually enter the values for features such as duration, src_bytes, dst_bytes, etc.

- duration = st.number_input("Duration", min_value=0.0, max_value=1000.0, step=0.1)
  src_bytes = st.number_input("Source Bytes", min_value=0, max_value=1000000, step=100)

And Text box for the catagorical values features such as protocol, service, and state, which is based on encoded classes saved during the preprocessing stage (e.g., classes_proto.npy, classes_service.npy).

- protocol = st.selectbox("Protocol", options=['tcp', 'udp', 'icmp']) service = st.selectbox("Service", options=['http', 'ftp', 'smtp', 'dns'])

To run the python file and Steamlit file use the following commands

- python app.py
- streamlit run app.py

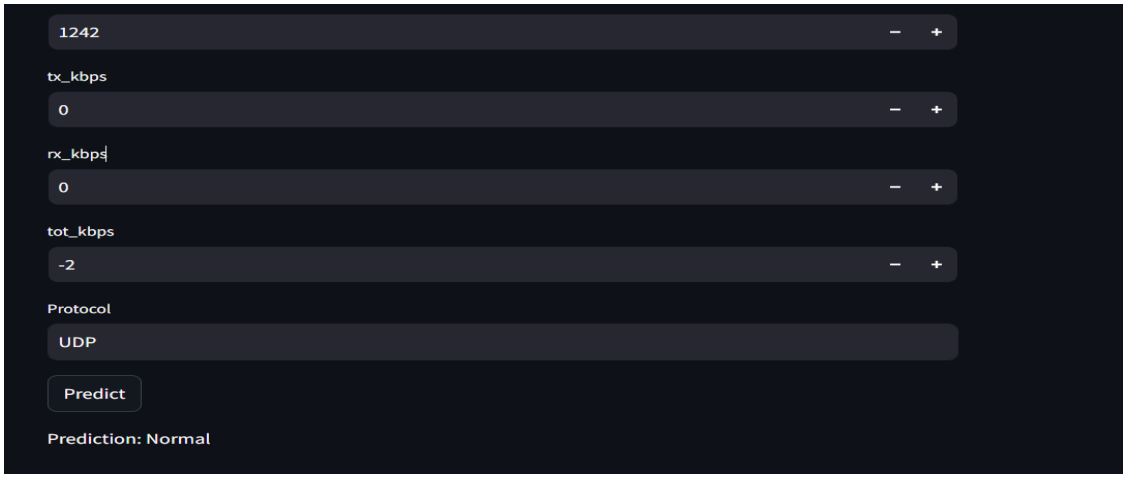

Fig.10 Streamlit user interface with inputs for normal traffic and reviving the real time Predictions.

Use the browser navigate to the following URL to see the web application.

URL: http://localhost:8503

In case of browser did not open automatically,then manually open the URL provided in the terminal.