Configuration Manual

MSc Research Project
MSc in Cloud Computing

Venkateshwarlu vanga
Student ID: x22158952

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Venkateshwarlu vanga
Student ID:	x22158952
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	1320
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:	Venkateshwarlu Vanga
Date:	14th December 2023

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

Venkateshwarlu vanga
x22158952

1 Introduction

This document aims to provide a detailed guide for setting up and managing the project efficiently. It covers system architecture, installation steps, configuration choices, execution process, and analysing the results. It also provides an overview of the research project's development for "Securing Cloud Environments Through Real-Time Network Monitoring System for Detecting Network Attacks using Advanced Deep Learning Methods" It's crucial to review this document thoroughly before deploying the project.

2 Prerequisites

This document is for people who are familiar with Ubuntu, Python, basic Deep Learning concepts, and Python Flask. Knowing these things will help you understand and use the information in this document more effectively.

3 Environment Setup

For setting up the environment, I utilized Anaconda for running Jupyter notebook, the same can see at Figure 1.

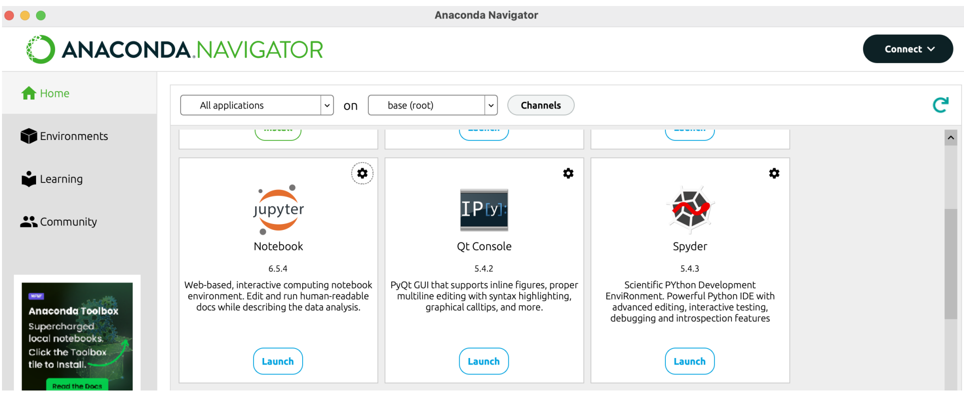


Figure 1: Anaconda Navigator

All necessary libraries, such as Pandas and Numpy, were installed. These libraries played a crucial role in reading, mapping, and visualizing the dataset. Additionally, the

Sklearn library (Scikit-learn) was employed for data analysis and modeling, offering various algorithms for classification. To develop deep learning models, I utilized Tensorflow and Keras. Figure 2.

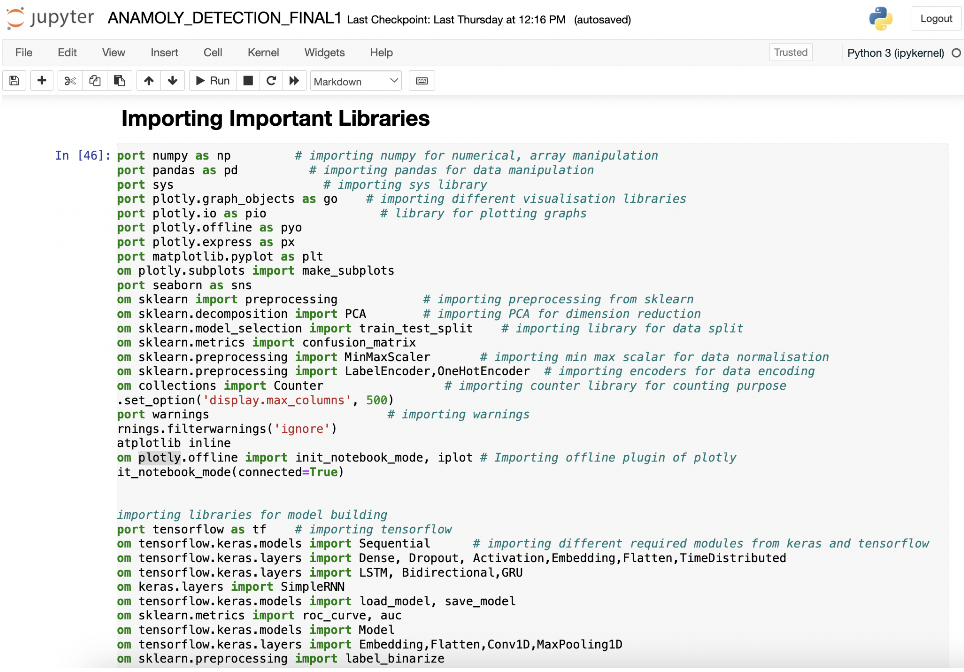


Figure 2: Libraries List

For implementation, AWS Cloud was used, and an EC2 Instance with the latest version of Ubuntu was configured Figure 3. The project primarily leverages the Python programming language, and we ensured the use of the latest version, which can be verified and downloaded from https://www.python.org/downloads/.

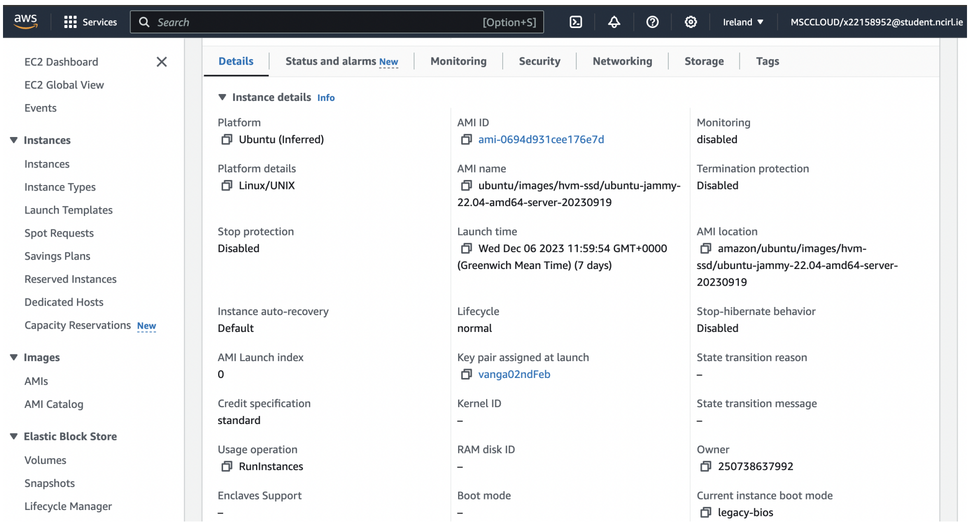


Figure 3: AWS EC2 instance

Multiple libraries, including TensorFlow, Matplotlib, and Scikit-learn, were incorporated into the project using the pip command as shown below. Along with Python Flask.

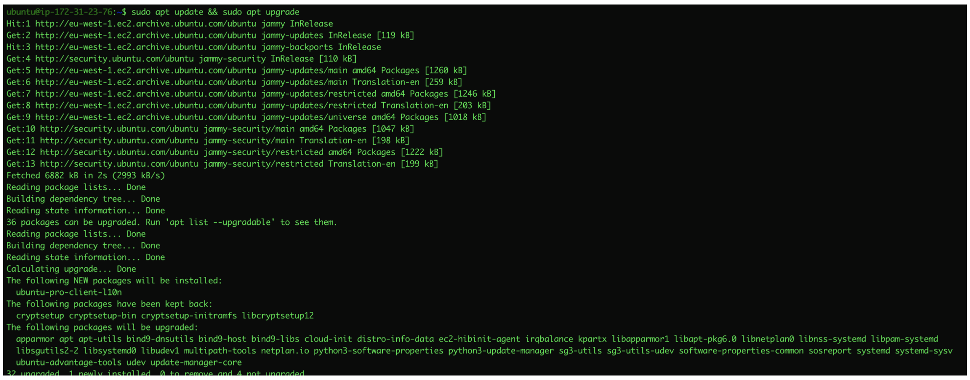


Figure 4: sudo apt update and upgrade

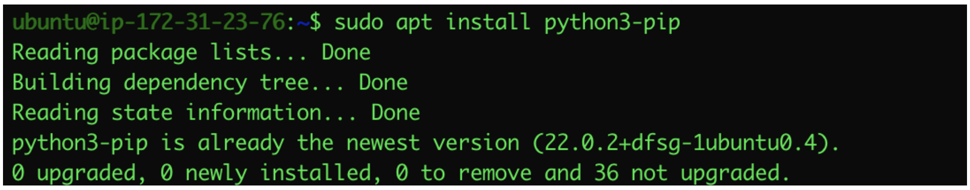


Figure 5: sudo apt install python3-pip

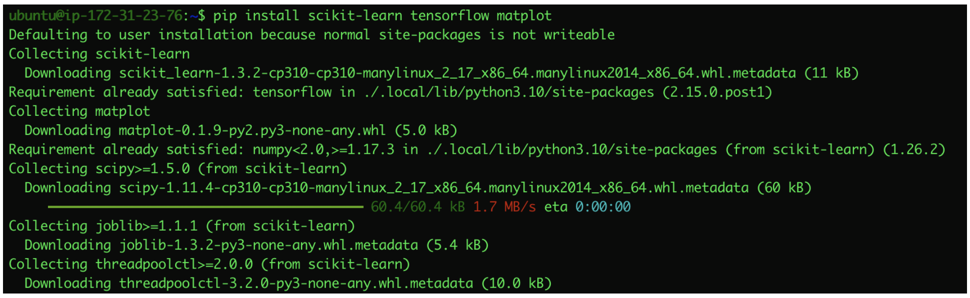


Figure 6: pip install scikit-learn tensorflow matplot

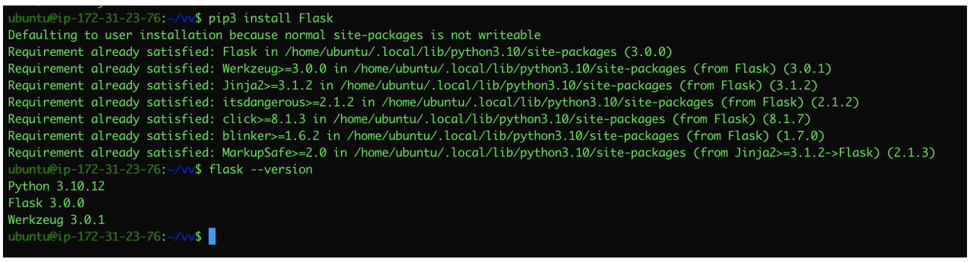


Figure 7: Installing Python Flask

4 Implementation

Our project model contains different components, which can be seen below.

4.1 Dataset collect

The UNSW-NB15 dataset was created by the University of New South Wales for use in the 2015 International Knowledge Discovery and Data Mining Tools Competition. Like the older KDD Cup 1999 dataset, the UNSW-NB15 dataset is intended to help develop effective network intrusion detection systems. The goal of the dataset is to enable the creation of models that can accurately differentiate between malicious and benign network traffic. A key strength of the UNSW-NB15 dataset is that it contains a wide variety of simulated attack types within a modeled university network environment. The audit-friendly data provides rich details on diverse intrusion scenarios. Overall, the UNSW-NB15 dataset represents a valuable research resource to drive continued progress on cybersecurity data analysis and predictive modeling for intrusion detection.

4.2 Data Preprocessing

In this stage first we are looking at the raw data from dataset. To accommodate the large dataset, we divided it into four separate CSV files named UNSW-NB15-1.csv, UNSW-NB15-2.csv, UNSW-NB15-3.csv, and UNSW-NB15-4.csv. These files are defined as df1, df2, df3, and df4, as shown in the Figure 8.

Reading Raw Data

Since this dataset is very large therefore chunked into 4 different csv files, Reading each file separately

```
In [47]: features = pd.read_csv('NUSW-NB15_features.csv',sep=",", encoding='cp1252')   # Reading mapping file of the data
         df1 = pd.read_csv('UNSW-NB15_1.csv')      # reading first file of data
         df2 = pd.read_csv('UNSW-NB15_2.csv')      # reading second file of data
         df3 = pd.read_csv('UNSW-NB15_3.csv')      # reading third file of data
         df4 = pd.read_csv('UNSW-NB15_4.csv')      # reading fourth file of data
```

Figure 8: Raw Data

We printed all the features from the features.csv file Figure 9 and displayed sample data from the first CSV file, df1, using the head and tail commands please refer to the Figure 10 for details.

```
In [48]: print('Features for this data is:')    # printing different features in data along with their meaning
         features

         Features for this data is:
```

Out[48]:

	No.	Name	Type	Description
0	1	srcip	nominal	Source IP address
1	2	sport	integer	Source port number
2	3	dstip	nominal	Destination IP address
3	4	dsport	integer	Destination port number
4	5	proto	nominal	Transaction protocol
5	6	state	nominal	Indicates to the state and its dependent proto...
6	7	dur	Float	Record total duration
7	8	sbytes	Integer	Source to destination transaction bytes
8	9	dbytes	Integer	Destination to source transaction bytes
9	10	sttl	Integer	Source to destination time to live value
10	11	dttl	Integer	Destination to source time to live value

Figure 9: Printing all the Features

```
In [49]: print('shape of data1 is:', df1.shape )      # Reading shape of each data file
         print('shape of data2 is:', df2.shape )
         print('shape of data3 is:', df3.shape )
         print('shape of data4 is:', df4.shape )

shape of data1 is: (700000, 49)
shape of data2 is: (700000, 49)
shape of data3 is: (700000, 49)
shape of data4 is: (440043, 49)
```

```
In [50]: df1.head()          # visualisng the first five rows of data from first file to understand the raw data
Out[50]:
```

	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	29	0	0.1	dns	500473.9375	621800.9375	2	2.1	0.2	0.3	0.4	0.5	66	82
0	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	29	0	0	-	87676.08594	50480.17188	4	4	0	0	0	0	132	76
1	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	29	0	0	dns	521894.53130	636282.37500	2	2	0	0	0	0	73	89
2	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	29	0	0	dns	436724.56250	542597.18750	2	2	0	0	0	0	66	82
3	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	29	0	0	dns	499572.25000	609067.56250	2	2	0	0	0	0	73	89
4	59.166.0.0	32119	149.171.126.9	111	udp	CON	0.078339	568	312	31	29	0	0	-	43503.23438	23896.14258	4	4	0	0	0	0	142	78

```
In [51]: df1.tail()        # visualisng the last five rows of data from first file to understand the raw data
Out[51]:
```

	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	29	0	0.1	dns	500473.9375	621800.9375	2	2.1	0.2	0.3
699995	59.166.0.8	12520	149.171.126.6	31010	tcp	FIN	0.020383	320	1874	31	29	1	2	-	1.047932e+05	6.436736e+05	6	8	255	255
699996	59.166.0.0	18895	149.171.126.9	80	tcp	FIN	1.402957	19410	1087890	31	29	2	370	http	1.103783e+05	6.195098e+06	364	746	255	255
699997	59.166.0.0	30103	149.171.126.5	5190	tcp	FIN	0.007108	2158	2464	31	29	6	6	-	2.328644e+06	2.658413e+06	24	24	255	255
699998	59.166.0.6	30388	149.171.126.5	111	udp	CON	0.004435	568	304	31	29	0	0	-	7.684329e+05	4.112740e+05	4	4	0	0
699999	59.166.0.0	6055	149.171.126.5	54145	tcp	FIN	0.072974	4238	60788	31	29	7	30	-	4.582454e+05	6.571546e+06	72	72	255	255

Figure 10: head and tail of the data

As the above data lacks headers, we added headers to all four CSV files, combined the data, and printed a sample with headers. Additionally, we provided an overview of the data, including the number of columns and rows. Furthermore, we printed the "label" data along with the count of 0's and 1's can be seen at Figure 11

```
In [53]: df_f = pd.concat([df1, df2, df3, df4], axis=0)      # concating the all data

In [54]: df_f.head()        # visualising the data after concating to assure the correct concatination
Out[54]:
```

	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	Sload	Dload	Spkts	Dpkts
0	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	29	0	0	-	87676.08594	50480.17188	4	4
1	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	29	0	0	dns	521894.53130	636282.37500	2	2
2	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	29	0	0	dns	436724.56250	542597.18750	2	2
3	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	29	0	0	dns	499572.25000	609067.56250	2	2
4	59.166.0.0	32119	149.171.126.9	111	udp	CON	0.078339	568	312	31	29	0	0	-	43503.23438	23896.14258	4	4

```
In [55]: df_f.shape     # shape of all data
Out[55]: (2540043, 49)

In [56]: df_f['Label'].value_counts()      # value counts of each class in target variable
Out[56]: 0    2218760
         1     321283
         Name: Label, dtype: int64
```

Figure 11: Concating the all data

Next, we printed the first and last 5 lines of the final data and checked the data types of each column. Following this, we examined numerical and categorical features, and duplicates were removed. Figure 12

```
In [62]: final_df.describe()          # checking statistics of the final dataset
Out[62]:
```

	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	Sload	Dload	Spkts
count	700000.000000	7.000000e+05	7.000000e+05	700000.000000	700000.000000	700000.000000	700000.000000	7.000000e+05	7.000000e+05	700000.000000
mean	0.700324	4.923738e+03	2.425832e+04	130.291137	36.596279	4.196073	10.904693	5.954284e+07	1.518594e+06	24.081077
std	15.359416	1.060344e+05	1.458921e+05	108.700437	69.186048	40.402237	52.162741	1.364941e+08	3.525351e+06	95.483510
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000
25%	0.000008	1.140000e+02	0.000000e+00	31.000000	0.000000	0.000000	0.000000	2.784314e+05	0.000000e+00	2.000000
50%	0.001102	2.640000e+02	1.780000e+02	60.000000	29.000000	0.000000	0.000000	1.478006e+06	8.407858e+03	2.000000
75%	0.114749	2.334000e+03	3.380000e+03	254.000000	29.000000	6.000000	6.000000	6.514286e+07	6.778242e+05	22.000000
max	8760.776367	1.435577e+07	1.465753e+07	255.000000	254.000000	5319.000000	5507.000000	5.988000e+09	4.807207e+07	10646.000000

```
In [63]: # dropping duplicate from the data
final_df = final_df.drop_duplicates()
print('Dimension of data after dropping duplicates: ',final_df.shape)    # getting dimension of data after dropping

Dimension of data after dropping duplicates:  (453771, 49)

In [19]: ımerical_features=[features for features in final_df.columns if final_df[features].dtypes !="O"]   # getting numerica
ıtegorical_features=[features for features in final_df.columns if final_df[features].dtypes =="O"]  # getting categor
int("Numerical Features Count {}".format(len(numerical_features)))
int(numerical_features)
int("Categorical Features Count {}".format(len(categorical_features)))
int(categorical_features)

Numerical Features Count 40
['dur', 'sbytes', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'Sload', 'Dload', 'Spkts', 'Dpkts', 'swin', 'dwin',
'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt',
```

Figure 12: Removing Duplicate data

We replaced missing values in the 'attack-cat' column with 'normal', substituted '-' with 'none', performed a fundamental transformation of IP addresses to decimal, and checked for null values. Figure 13

```
In [28]: def ip_to_decimal(ip):
    if isinstance(ip, int):
        return ip  # If already an integer, return it as is
    octets = ip.split('.')
    binary = '{0:08b}{1:08b}{2:08b}{3:08b}'.format(*map(int, octets))
    decimal = int(binary, 2)  # Convert binary to decimal
    return decimal

final_df['srcip'] = final_df['srcip'].apply(ip_to_decimal)
final_df['dstip'] = final_df['dstip'].apply(ip_to_decimal)

In [29]: # checking null values in data
print('Null values in each column are:\n',final_df.isnull().sum())

Null values in each column are:
 srcip                0
sport               0
dstip               0
dsport              0
proto               0
state               0
dur                 0
```

Figure 13: Checking null values

4.3 Data analysis and visualisation

After that we provided a comprehensive view of Source-to-Destination (S/D) and Destination-to-Source (D/S) transaction bytes, highlighting their majority. Additionally, we visualized the data structure using bar plots Figure 14 and displayed different attack categories with respect to labels, using both bubble scatter plots and histogram plots Figure 15

```
In [35]: # bar plot of Different Service Count with respect to Label
         # Set the figure size
         plt.figure(figsize=(12, 8))

         # Plot the countplot with customizable figure size and bar color
         sns.countplot(x='service', hue='Label', data=final_df, palette='Set2')  # You can change 'Set2' to your preferred co

         # Set title and show the plot
         plt.title('Different Service Count with respect to Label')
         plt.show()
```

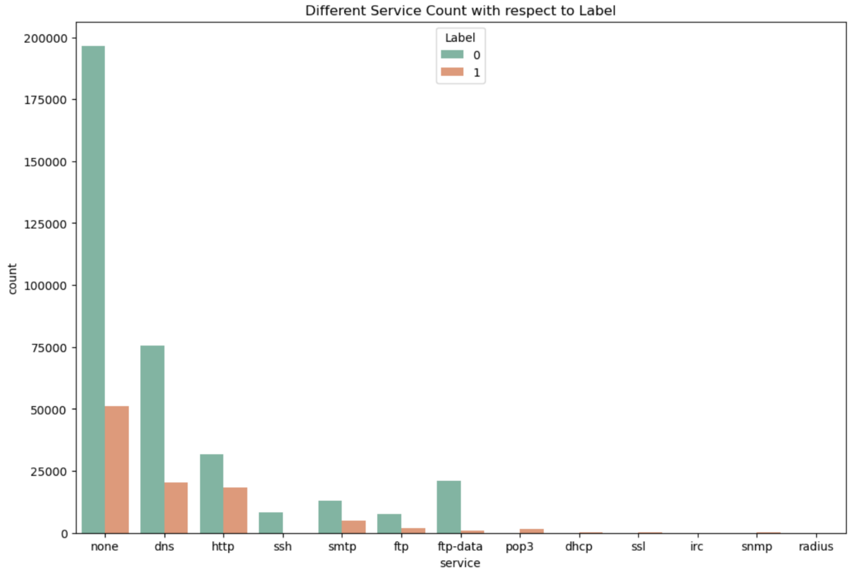


Figure 14: Bar Plot

```
In [37]: # Creating a bubble scatter plot to visualize the S/D Packet count and D/S Packets count with Attacks
         # Defining a dictionary to map each label to a specific size
         mapp = {
             0: 5,
             1: 15
         }

         final_df['bubble_size'] = final_df['Label'].map(mapp)

         custom_colors = ['#FB2576', '#3F0071', '#FF8E00', '#00AF91']
         fig = px.scatter(
             final_df,
             x='Spkts',
             y='Dpkts',
             size='bubble_size',
             color_discrete_sequence=custom_colors,
             color='Label',
             title="S/D Packet count and D/S Packets count with Attacks"
         )
         fig.show()
```

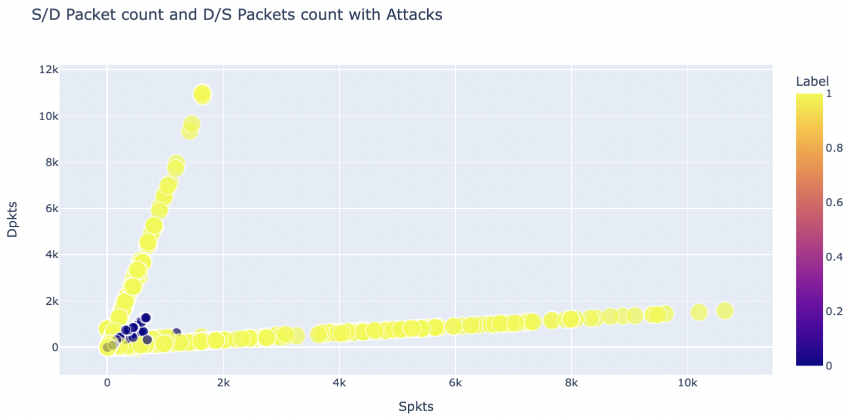


Figure 15: Bubble Scatter Plots and Histogram Plots

4.4 Feature Engineering

Feature engineering is a crucial step in optimizing the dataset for deep learning applications. Categorical columns in the dataset were transformed into numerical values using 'Label Encoding' to achieve both balanced Figure 16 and imbalanced Figure 17 representation. Please refer to the image for more details.

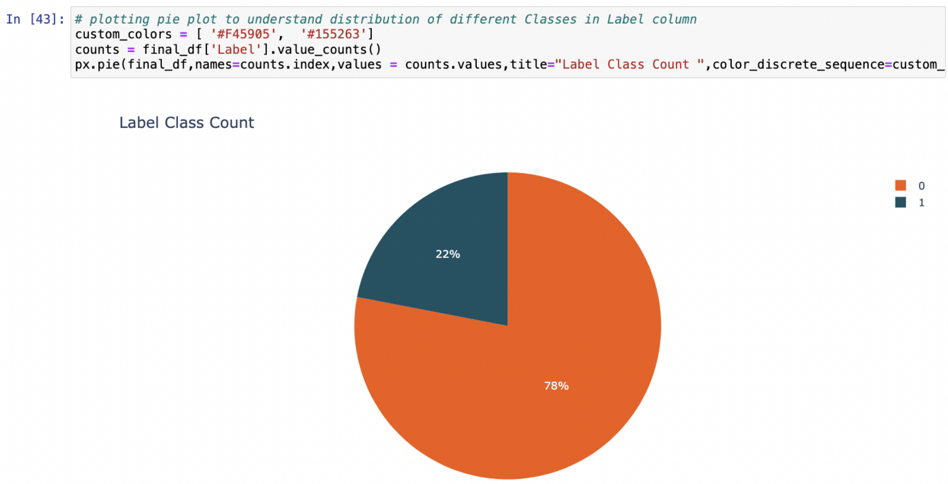


Figure 16: Imbalanced Class in Label Column

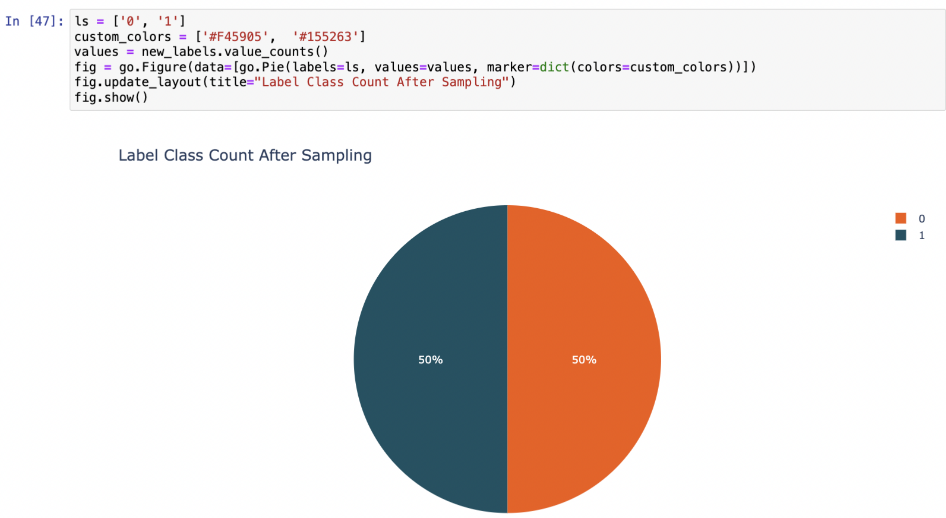


Figure 17: Balanced Class in Label Class After Sampling

To manage resources effectively and mitigate the risk of overfitting, Principal Component Analysis (PCA) was used to reduce the dimensionality of the data Figure 18

Implementing Principle Component Analysis

```
In [49]: import plotly.express as px
         from sklearn.decomposition import PCA
         pca = PCA().fit(scaled_data)
         # Cumulative explained variance
         cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
         # Creating a DataFrame for plotting
         df_plot = pd.DataFrame({'Number of Components': np.arange(1, len(cumulative_variance) + 1),
                                 'Cumulative Variance': cumulative_variance})

         # Creating a combined bar and line graph using Plotly Express
         fig = px.bar(df_plot, x='Number of Components', y='Cumulative Variance',
                      title='The Number of Components Needed to Explain Variance',
                      labels={'Cumulative Variance': 'Cumulative Variance (%)'},
                      color_discrete_sequence=['rgba(0,100,80,0.6)'])

         # Adding a line for the 95% cut-off threshold
         fig.add_trace(go.Scatter(x=df_plot['Number of Components'], y=df_plot['Cumulative Variance'],
                                  mode='lines', name='Cumulative Variance',
                                  line=dict(color='red', dash='dash')))


         fig.show()
```

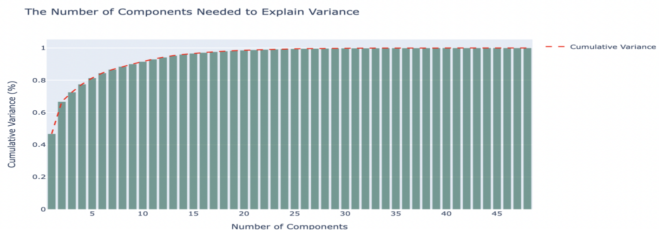


Figure 18: Principal Component Analysis

Since the 18 components capture 98 percent of the information, this not only makes computations faster but also assists the model in handling new data. This refined data is then used for training and evaluating the model, where we split it into training and test datasets. Figure 19

```
In [50]: pca_comp = PCA(n_components = 18)      # 18 components are capturing 98% of Info from the data
         pca_scaled_data = pca_comp.fit_transform(scaled_data)      # indicates the varience of each component
         print(np.sum(pca_comp.explained_variance_ratio_))

         0.9800545654248592

In [51]: pca_scaled_data.shape       # shape of data after PCA implementation

Out[51]: (199292, 18)

In [52]: # train and test split of data
         from sklearn.model_selection import train_test_split  # importing train test split libraray
         X_train, X_test, y_train, y_test = train_test_split(pca_scaled_data,new_labels, test_size =0.3,random_state = 1,shuff

In [53]: print(X_train.shape)      # shape of training data
         print(X_test.shape)      # shaping of test data

         (139504, 18)
         (59788, 18)
```

Figure 19: Splitting data into training and test datasets

4.5 Model Training

Here we are using 3 machine learning models Recurrent Neural Networks (RNN), Autoencoder, and Graph Neural Networks (GNN) algorithms. The training data was reshaped to maintain temporal relationships, utilizing binary cross-entropy loss and the Adam optimizer over 10 epochs with 512-sample batches to underlying patterns in the data.
Figure 20 RNN
Figure 21 Autoencoder
Figure 22 GNN

Recurrent Neural Network (RNN) Model:

```
In [57]: model1=Sequential()                                   # adding first layera as seq layer
         model1.add(tf.keras.layers.Input(shape=(1, X_train.shape[1])))    # adding input layer
         #model1.add(tf.keras.layers.BatchNormalization())    # adding batch normalization layer
         model1.add(SimpleRNN(5))      # Adding SimpleRNN layer with 5 neurons
         model1.add(Dropout(0.2))   # adding dropout layer to avoid overfitting of model
         # model1.add(SimpleRNN(20))      # Adding SimpleRNN layer with 20 neurons
         # model1.add(Dropout(0.2))   # adding dropout layer to avoid overfitting of model
         # model1.add(Dense(10,activation='linear'))  # adding dense layer with 10 neurons
         # model1.add(Dropout(0.2))   # adding dropout layer to avoid overfitting of model
         model1.add(Dense(1,activation="sigmoid"))  # adding output layer with 1 neurons
         model1.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])  # compiling model
         model1.summary()
```

```
In [58]: history1=model1.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size= 512, epochs= 10)   # training RN

Epoch 1/10
273/273 [==============================] - 5s 9ms/step - loss: 0.6106 - accuracy: 0.6747 - val_loss: 0.3858 - val_a
ccuracy: 0.9698
Epoch 2/10
273/273 [==============================] - 1s 3ms/step - loss: 0.2776 - accuracy: 0.9727 - val_loss: 0.1744 - val_a
ccuracy: 0.9896
Epoch 3/10
273/273 [==============================] - 1s 2ms/step - loss: 0.1519 - accuracy: 0.9874 - val_loss: 0.1009 - val_a
ccuracy: 0.9904
Epoch 4/10
273/273 [==============================] - 1s 2ms/step - loss: 0.1021 - accuracy: 0.9899 - val_loss: 0.0709 - val_a
ccuracy: 0.9908
Epoch 5/10
273/273 [==============================] - 1s 2ms/step - loss: 0.0794 - accuracy: 0.9906 - val_loss: 0.0568 - val_a
ccuracy: 0.9912
Epoch 6/10
273/273 [==============================] - 1s 2ms/step - loss: 0.0667 - accuracy: 0.9909 - val_loss: 0.0494 - val_a
ccuracy: 0.9915
Epoch 7/10
273/273 [==============================] - 1s 2ms/step - loss: 0.0599 - accuracy: 0.9911 - val_loss: 0.0453 - val_a
ccuracy: 0.9917
Epoch 8/10
273/273 [==============================] - 1s 2ms/step - loss: 0.0547 - accuracy: 0.9914 - val_loss: 0.0428 - val_a
ccuracy: 0.9918
Epoch 9/10
273/273 [==============================] - 1s 3ms/step - loss: 0.0522 - accuracy: 0.9915 - val_loss: 0.0412 - val_a
ccuracy: 0.9918
Epoch 10/10
273/273 [==============================] - 1s 3ms/step - loss: 0.0503 - accuracy: 0.9916 - val_loss: 0.0401 - val_a
ccuracy: 0.9918
```

Figure 20: Recurrent Neural Network (RNN)

AutoEncoder Model

```
In [64]: #AutoEncoder model2
         n_features = X_train.shape[1]
         model2 = Sequential()
         model2.add(tf.keras.layers.Input(shape=(1, n_features)))
         # Encoder Layer 1
         model2.add(tf.keras.layers.Conv1D(filters=n_features*4, kernel_size=1, activation='relu'))
         model2.add(Dropout(0.1))
         # Encoder Layer 2
         model2.add(tf.keras.layers.Conv1D(filters=n_features*2, kernel_size=1, activation='relu'))
         model2.add(Dropout(0.1))
         # Encoder Layer 3
         model2.add(tf.keras.layers.Conv1D(filters=n_features, kernel_size=1, activation='relu'))
         model2.add(Dropout(0.1))

         # BottleNeck
         model2.add(tf.keras.layers.BatchNormalization())
         model2.add(tf.keras.layers.Flatten())
         model2.add(tf.keras.layers.Dense(n_features,activation='relu'))
         model2.add(tf.keras.layers.Reshape((1,n_features)))
```

```
In [65]: =model2.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size= 512, epochs= 10)   # training autoencoder

Epoch 1/10
273/273 [==============================] - 19s 20ms/step - loss: 0.1344 - accuracy: 0.9463 - val_loss: 0.0350 - val
_accuracy: 0.9919
Epoch 2/10
273/273 [==============================] - 4s 14ms/step - loss: 0.0323 - accuracy: 0.9917 - val_loss: 0.0236 - val_
accuracy: 0.9924
Epoch 3/10
273/273 [==============================] - 4s 15ms/step - loss: 0.0222 - accuracy: 0.9927 - val_loss: 0.0099 - val_
accuracy: 0.9984
Epoch 4/10
273/273 [==============================] - 4s 15ms/step - loss: 0.0111 - accuracy: 0.9965 - val_loss: 0.0024 - val_
accuracy: 0.9995
Epoch 5/10
273/273 [==============================] - 4s 16ms/step - loss: 0.0072 - accuracy: 0.9978 - val_loss: 0.0011 - val_
accuracy: 0.9998
Epoch 6/10
273/273 [==============================] - 3s 13ms/step - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.0011 - val_
accuracy: 0.9996
Epoch 7/10
273/273 [==============================] - 5s 17ms/step - loss: 0.0029 - accuracy: 0.9991 - val_loss: 7.2860e-04 -
val_accuracy: 0.9998
Epoch 8/10
273/273 [==============================] - 4s 15ms/step - loss: 0.0028 - accuracy: 0.9992 - val_loss: 7.8217e-04 -
val_accuracy: 0.9998
Epoch 9/10
273/273 [==============================] - 5s 18ms/step - loss: 0.0026 - accuracy: 0.9992 - val_loss: 5.7153e-04 -
val_accuracy: 0.9999
Epoch 10/10
273/273 [==============================] - 4s 15ms/step - loss: 0.0024 - accuracy: 0.9993 - val_loss: 5.8106e-04 -
val_accuracy: 0.9999
```

Figure 21: AutoEncoder

Graph Neural Network (GNN) Model:

```
In [70]: from keras.models import Model
         from keras.layers import Input, Dense, Flatten, Concatenate
         import numpy as np

         # Reshaping the data to remove the unnecessary singleton dimension
         x_train2 = x_train.reshape(x_train.shape[0], x_train.shape[2])
         x_test2 = x_test.reshape(x_test.shape[0], x_test.shape[2])


         num_nodes = x_train2.shape[1]
         adjacency_matrix = np.ones((num_nodes, num_nodes))

         # Converting adjacency matrix to edge list
         edges = np.column_stack(np.where(adjacency_matrix == 1))
         graph_input = Input(shape=(num_nodes, num_nodes,))
         feature_input = Input(shape=(x_train2.shape[1],))

         # dense layers for node feature processing
         dense1 = Dense(128, activation='relu')(feature_input)
         dense2 = Dense(64, activation='relu')(dense1)
         dense3 = Dense(32, activation='relu')(dense2)
```

```
Epoch 1/10
3488/3488 [==============================] - 14s 4ms/step - loss: 0.0282 - accuracy: 0.9923 - val_loss: 0.0059 - va
l_accuracy: 0.9987
Epoch 2/10
3488/3488 [==============================] - 13s 4ms/step - loss: 0.0075 - accuracy: 0.9978 - val_loss: 0.0191 - va
l_accuracy: 0.9923
Epoch 3/10
3488/3488 [==============================] - 14s 4ms/step - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0028 - va
l_accuracy: 0.9991
Epoch 4/10
3488/3488 [==============================] - 13s 4ms/step - loss: 0.0035 - accuracy: 0.9989 - val_loss: 0.0029 - va
l_accuracy: 0.9992
Epoch 5/10
3488/3488 [==============================] - 10s 3ms/step - loss: 0.0027 - accuracy: 0.9992 - val_loss: 0.0013 - va
l_accuracy: 0.9997
Epoch 6/10
3488/3488 [==============================] - 10s 3ms/step - loss: 0.0025 - accuracy: 0.9992 - val_loss: 0.0028 - va
l_accuracy: 0.9993
Epoch 7/10
3488/3488 [==============================] - 9s 3ms/step - loss: 0.0022 - accuracy: 0.9994 - val_loss: 0.0016 - val
_accuracy: 0.9996
Epoch 8/10
3488/3488 [==============================] - 12s 3ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 8.8814e-04
- val_accuracy: 0.9998
Epoch 9/10
3488/3488 [==============================] - 13s 4ms/step - loss: 0.0026 - accuracy: 0.9992 - val_loss: 0.0012 - va
l_accuracy: 0.9998
Epoch 10/10
3488/3488 [==============================] - 15s 4ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 0.0010 - va
l_accuracy: 0.9999
```

Figure 22: Graph Neural Network (GNN)

4.6 Model Evaluation and Results

For model evaluation, we assessed performance using metrics such as accuracy, sensitivity, false positive rate, and specificity, offering nuanced insights into the model's ability to distinguish between normal and anomalous instances in the test data. The evaluation also included the use of the ROC-AUC curve, and based on the results, the Autoencoder demonstrated good performance Figure 24.
The 'Autoencoder ROC-AUC curve' can be seen at Figure 23. And similar execution for RNN and GNN models.

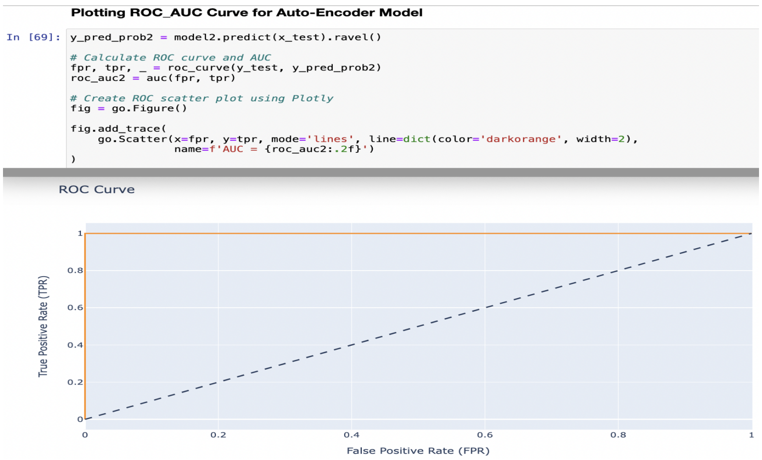


Figure 23: Autoencoder ROC-AUC curve

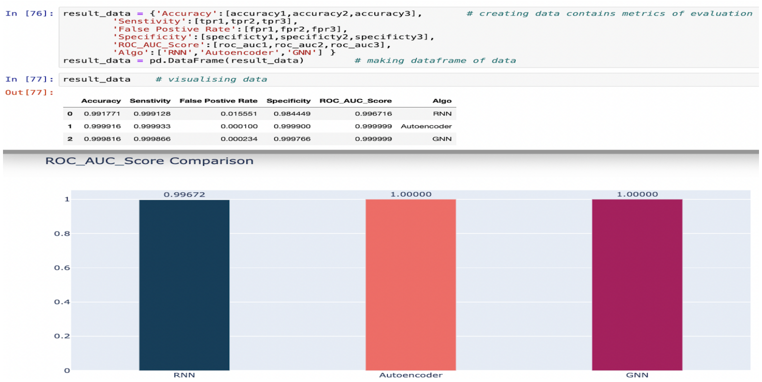


Figure 24: ROC-AUC-Score Comparison

4.7 Web-UI Implementation And Execution

Monitoring the cloud environment in real-time is a crucial task for timely detection and response to anomalies or attacks. We developed a web interface for live network monitoring and alerting users, employing a server-client model. The web application, developed in 'Visual Studio code' in Python Flask Figure 25 Flask is a lightweight and efficient Python web framework, and it will utilizes our 'Autoencoder' model to predict the cloud network traffic sent by the client.

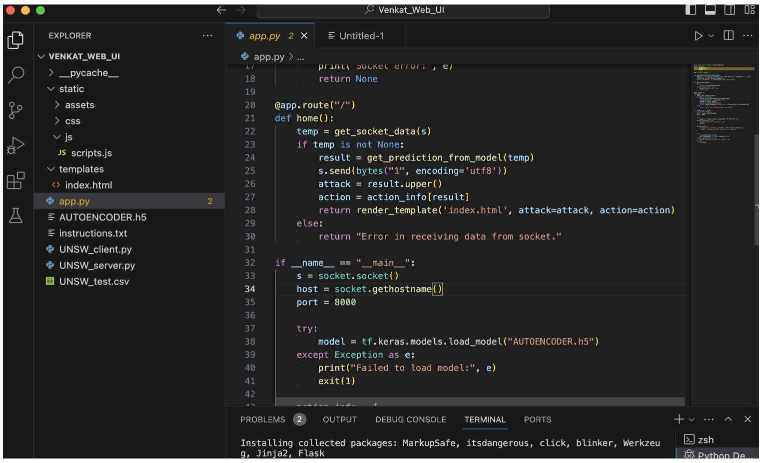
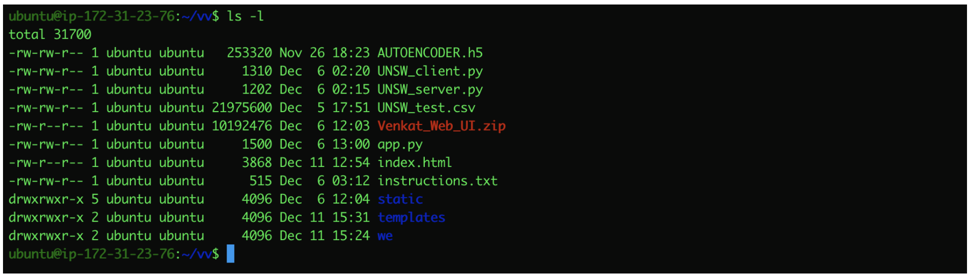


Figure 25: Visual Studio

Python socket programming was used to enable real-time data communication between the client and server, as shown in the image. The application utilizes our final model to analyze network data and provides real-time predictions of network anomalies, classifying them as 'Normal' or 'Anomalous'. Please refer to the below images for reference.
Now, follow the below steps.
1. Connect to ec2 instance via command line or putty or RDP. We need two terminals, one is to run the Client script and another is to run the application.
ssh -i "key.pem" ec2-user@ec2.aws.com
2. Copy all the files related to our Web application from 'Visual Studio' to ec2 intance and extract them Figure 26
3. In one terminal run the 'UNSW-client.py' client script Figure 29
3. In second terminal run the application file i.e 'app.py'. It will ask us to connect to the URL "http://x.x.x.x:5000" to access the application Figure 30.
4. The Client sends the packets one after the other Figure 29
5. Our Deep Learning model predicts the incoming packets. And show's a "Green Color" 31 for "Normal" and "RED Color" 32 for "Anomalous"

Figure 26 Shows the list of flies in the web application
Figure 27 Shows the 'app.py' file, it has the best performing model i.e 'AUTOENCODER.h5' that will predict the incoming packets.
Figure 28 Shows the 'UNSW-client.py' client script, it will use the test data "UNSW-test.csv"
Figure 29 We are running the Client script and we can see it sending the packets to the server
Figure 30 Shows the Server receiving a response from a client
Figure: 31 32 shows the UI that we are accessing at "http://ip.address:5000" the public IP of our EC2 instance followed by port number.



```
ubuntu@ip-172-31-23-76:~/vv$ ls -l
total 31700
-rw-rw-r-- 1 ubuntu ubuntu   253320 Nov 26 18:23 AUTOENCODER.h5
-rw-rw-r-- 1 ubuntu ubuntu     1310 Dec  6 02:20 UNSW_client.py
-rw-rw-r-- 1 ubuntu ubuntu     1202 Dec  6 02:15 UNSW_server.py
-rw-rw-r-- 1 ubuntu ubuntu 21975600 Dec  5 17:51 UNSW_test.csv
-rw-r--r-- 1 ubuntu ubuntu 10192476 Dec  6 12:03 Venkat_Web_UI.zip
-rw-rw-r-- 1 ubuntu ubuntu     1500 Dec  6 13:00 app.py
-rw-r--r-- 1 ubuntu ubuntu     3868 Dec 11 12:54 index.html
-rw-rw-r-- 1 ubuntu ubuntu      515 Dec  6 03:12 instructions.txt
drwxrwxr-x 5 ubuntu ubuntu     4096 Dec  6 12:04 static
drwxrwxr-x 2 ubuntu ubuntu     4096 Dec 11 15:31 templates
drwxrwxr-x 2 ubuntu ubuntu     4096 Dec 11 15:24 we
ubuntu@ip-172-31-23-76:~/vv$
```

Figure 26: list of flies - ls -l

```
import tensorflow as tf
import numpy as np

app = Flask(__name__)

def get_prediction_from_model(data):
    temp_arr = np.array(list(map(float, data.split()))).reshape((-1, 1, 18))
    prediction = model.predict(temp_arr)
    return 'Anomalous' if prediction > 0.5 else 'Normal'

def get_socket_data(s):
    try:
        return s.recv(1024).decode()
    except socket.error as e:
        print("Socket error:", e)
        return None

@app.route("/")
def home():
    temp = get_socket_data(s)
    if temp is not None:
        result = get_prediction_from_model(temp)
        s.send(bytes("1", encoding='utf8'))
        attack = result.upper()
        action = action_info[result]
        return render_template('index.html', attack=attack, action=action)
    else:
        return "Error in receiving data from socket."

if __name__ == "__main__":
    s = socket.socket()
    host = socket.gethostname()
    port = 8000

    try:
        model = tf.keras.models.load_model("AUTOENCODER.h5")
    except Exception as e:
        print("Failed to load model:", e)
        exit(1)

    action_info = {
        'Anomalous': 'System is attacked, Take actions Immediately',
        'Normal': 'System is Safe, No action is required'
    }

    try:
        s.connect((host, port))
        app.run(debug=True, use_reloader=False, host='0.0.0.0')
    except socket.error as e:
        print("Failed to connect to socket:", e)
    finally:
```

Figure 27: vi app.py

```
import socket
import time
import pandas as pd

def main():
    try:
        # Load the test data from CSV
        test_data = pd.read_csv("UNSW_test.csv")
    except Exception as e:
        print("Failed to load data:", e)
        return

    host = socket.gethostname()
    port = 8000
    s = socket.socket()
    s.bind((host, port))
    s.listen(5)
    print(f"Server is listening on {host}:{port}")

    try:
        while True:
            print("Waiting for a client connection...")
            c, addr = s.accept()
            print('Got connection from', addr)

            idx = 0
            try:
                while True:
                    row_data = " ".join(test_data.iloc[idx].values.astype("str"))
                    c.sendall(row_data.encode('utf-8'))
                    print(f"Sent Network Packet {idx + 1}")
                    idx = (idx + 1) % len(test_data)
                    time.sleep(10)
            except socket.error as e:
                print("Socket error:", e)
                print("Client disconnected. Waiting for a new connection.")
            finally:
                c.close()
    except KeyboardInterrupt:
        print("Server is shutting down.")
    finally:
        s.close()
        print("Socket closed.")

if __name__ == "__main__":
    main()
```

Figure 28: vi UNSW-client.py

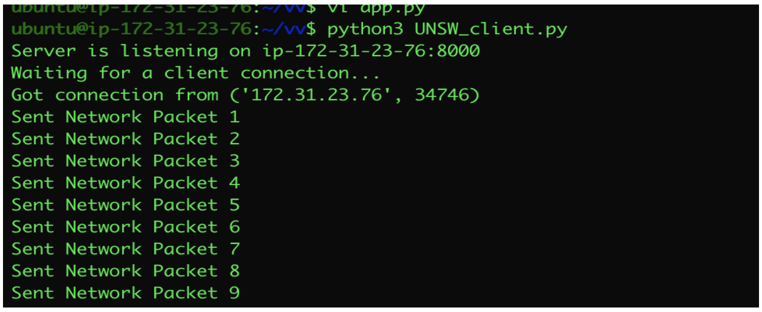


Figure 29: python3 UNSW-client.py

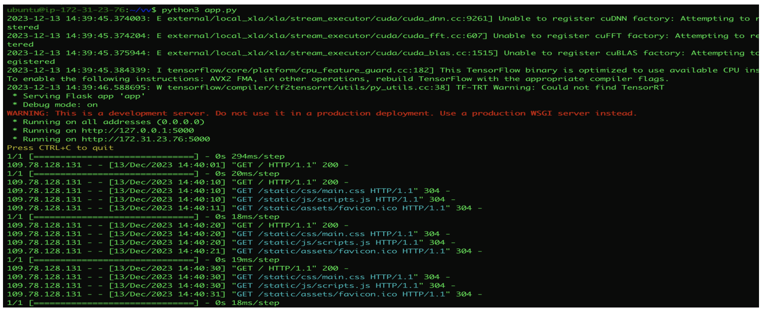


Figure 30: python3 app.py

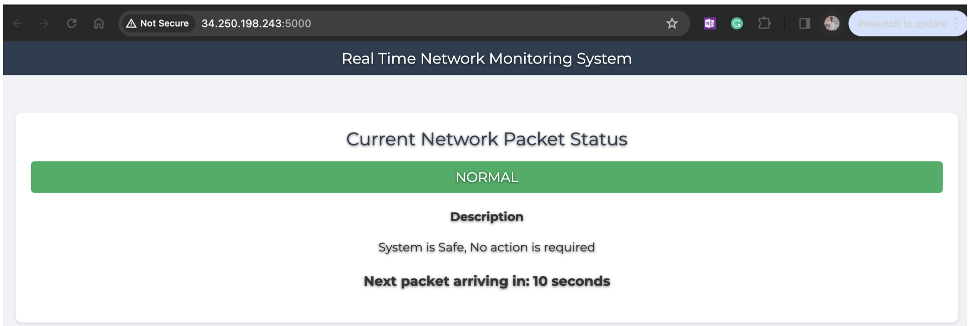


Figure 31: Networking Monitoring System Predicting Normal Traffic

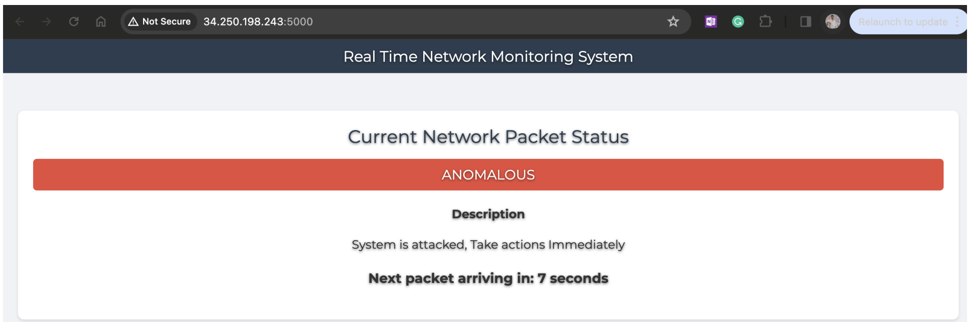


Figure 32: Network Monitoring System Predicting Anomalous Traffic