Configuration Manual

MSc Research Project
Msc in Cloud Computing

Swathy Menon Balachandran
StudentID:23108568

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Swathy Menon Balachandran
Student ID:	23108568
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1098
Page Count:	13

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:	Swathy Menon Balachandran
Date:	12th August 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

Swathy Menon Balachandran
23108568

1. Introduction

This configuration manual provides detailed instructions for setting up and implementing the proposed Intrusion Detection System(IDS) for IoT networks with title 'Enhancing Computational Efficiency and Time Optimization in Cloud IoT Intrusion Detection Using ANN and Hybrid Deep Learning'. This manual covers the necessary tools, environment setup, data preparation, model development, and evaluation processes to ensure optimal system performance and security.

2. Implementation

2.1 Experimental Setup

i. Dataset collection

Google Drive is used to store and manage both the dataset and the. ipynb file which contains the machine learning code for my analysis (Figure 1). Google Colab connects to Google Drive to access and manage all files stored in it (Figure 2)

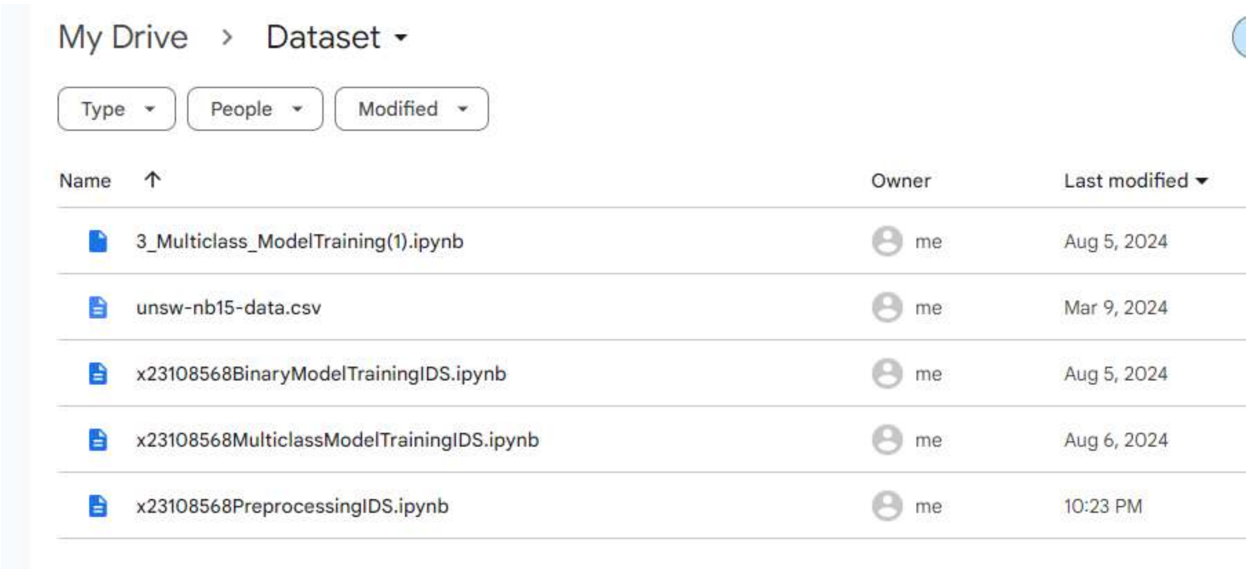


Figure1: Dataset and Code Management in Google Drive

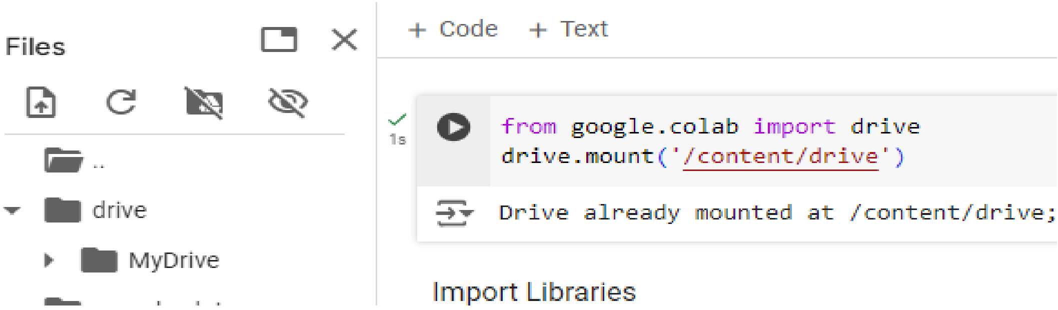
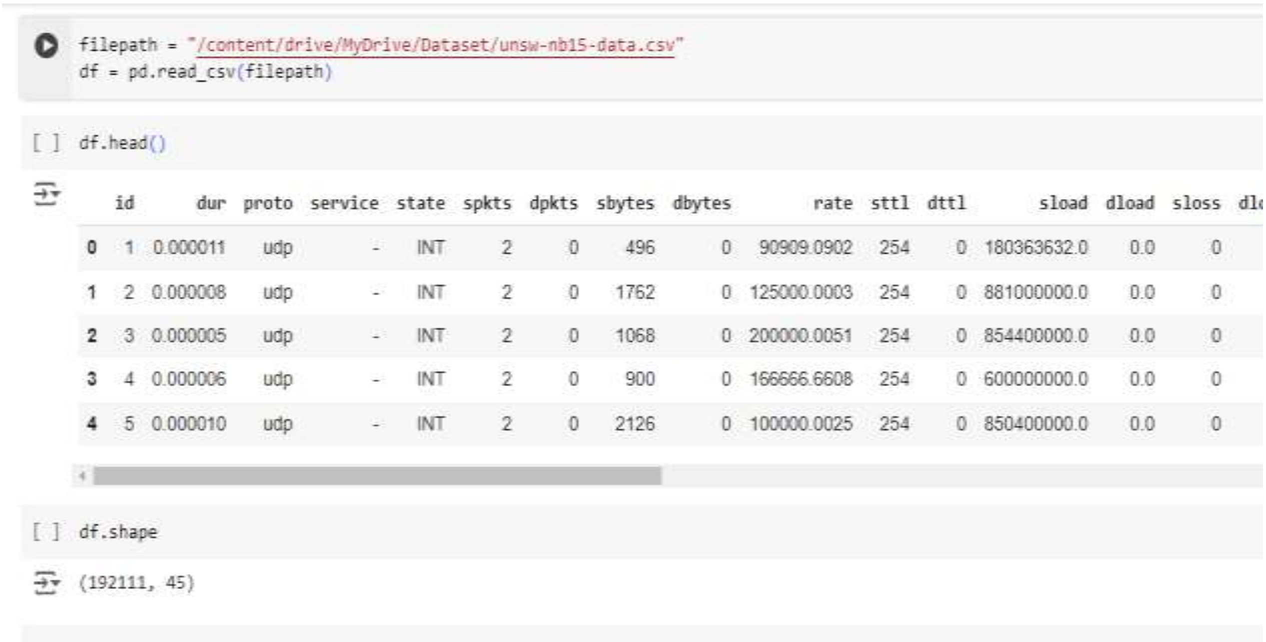


Figure2: Google Drive Mount Code in Google Colab

ii. **Dataset Loading**

Importing the necessary libraries for data manipulation, numerical operations, and visualization defined at the initial phase of the preprocessing. Figure 2 shows the dataset was successfully imported for analysis by uploaded from a specific file location and have error handling in place to handle any issues with file availability



```
filepath = "/content/drive/MyDrive/Dataset/unsw-nb15-data.csv"
df = pd.read_csv(filepath)
```

```
df.head()
```

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss	dl
0	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	254	0	180363632.0	0.0	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	254	0	881000000.0	0.0	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	254	0	854400000.0	0.0	0	
3	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	254	0	600000000.0	0.0	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	254	0	850400000.0	0.0	0	

```
df.shape
```

(192111, 45)

Figure3: Dataset Loading Successfully from Google Drive

3. Libraries and Packages

- **Pandas (Version: 2.1.4):** Used for data manipulation and analysis. It included the data structures and functions required to clean and prepare the dataset.
- **NumPy (Version: 1.26.4)**: For numerical computations, particularly for handling arrays and matrices.
- **Matplotlib and Seaborn (Version: 3.7.1 and 0.13.1):** Used for data visualization. These libraries helped in generating plots and charts to explore and present the data visually.

- **Warnings:** During code execution, the warnings module was used to suppress unnecessary warnings.
- **Scikit-learn (MinMaxScaler) (Version: 1.2.3):** Provided tools for scaling features to a range, which is an important step of making the data to run machine learning algorithms
- **TensorFlow/Keras(Version:2.17.0):** An open-source library for building and training machine learning models
- **Python Flask (Version: 2.0.3)**: Provides a lightweight framework for creating Intrusion detection application and integrating with the ML models.

4. Phases

4.1 Data Collection

This research utilizes the UNSW-NB15 dataset (Data World, 2024) sourced from dataworld (https://data.world/victorpuli/useful-unsw-nb15-data) to support the development and evaluation of Intrusion Detection Systems in IoT environments.

4.2 Data Info

The below figure 4 represent the dataset structure

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 192111 entries, 0 to 192110
Data columns (total 44 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   dur                192111 non-null  float64
 1   proto              192111 non-null  object
 2   service            192111 non-null  object
 3   state              192111 non-null  object
 4   spkts              192111 non-null  int64
 5   dpkts              192111 non-null  int64
 6   sbytes             192111 non-null  int64
 7   dbytes             192111 non-null  int64
 8   rate               192111 non-null  float64
 9   sttl               192111 non-null  int64
 10  dttl               192111 non-null  int64
 11  sload              192111 non-null  float64
 12  dload              192111 non-null  float64
 13  sloss              192111 non-null  int64
 14  dloss              192111 non-null  int64
 15  sinpkt             192111 non-null  float64
 16  dinpkt             192111 non-null  float64
 17  sjit               192111 non-null  float64
 18  djit               192111 non-null  float64
 19  swin               192111 non-null  int64
 20  stcpb              192111 non-null  int64
 21  dtcpb              192111 non-null  int64
 22  dwin               192111 non-null  int64
 23  tcprtt             192111 non-null  float64
 24  synack             192111 non-null  float64
 25  ackdat             192111 non-null  float64
 26  smean              192111 non-null  int64
 27  dmean              192111 non-null  int64
 28  trans_depth        192111 non-null  int64
 29  response_body_len  192111 non-null  int64
 30  ct_srv_src         192111 non-null  int64
 31  ct_state_ttl       192111 non-null  int64
 32  ct_dst_ltm         192111 non-null  int64
```

Figure 4: Shows the structure of the dataset used

4.3 Dataset Preprocessing

Figure 5 and Figure 6 represent the dataset reviewed for null values and infinite value analysis, respectively

```
#Nullvalues Analysis
null_result = dict(df.isnull().sum())
null_df = pd.DataFrame(data=null_result, index=[0])
null_df = null_df.T.reset_index()
null_df.columns = ['Feature', 'Nullvalue Count']
null_df
```

	Feature	Nullvalue Count
0	dur	0
1	proto	0
2	service	0
3	state	0
4	spkts	0
5	dpkts	0
6	sbytes	0
7	dbytes	0
8	rate	0
9	sttl	0
10	dttl	0
11	sload	0
12	dload	0
13	sloss	0
14	dloss	0

Figure5: Shows the Null Value Analysis

```
#Infinity values Analysis
features = []
value_counts = []
for feature in df.columns:
    infinity_count = df[feature].isin([np.inf, -np.inf]).sum()
    features.append(feature)
    value_counts.append(infinity_count)

infinity_df = pd.DataFrame()
infinity_df['features'] = features
infinity_df['counts'] = value_counts
infinity_df.head(50)
```

6	sbytes	0
7	dbytes	0
8	rate	0
9	sttl	0
10	dttl	0
11	sload	0
12	dload	0
13	sloss	0
14	dloss	0
15	sinpkt	0
16	dinpkt	0
17	sjit	0
18	djit	0
19	swin	0
20	stcpb	0

Figure 6: Illustrates the analysis of infinite values

4.4 Attack Category Analysis

The distribution of attack categories was analysed (Figure 7) and visualized using bar charts and pie charts to understand the frequency and proportion of each category

(Figure 8). Similarly, the distribution of service categories was visualized with bar charts to depict the count of each service type (Figure 8) and (Figure 9).

```
[ ] attack_counts = df['attack_cat'].value_counts()
    categories = attack_counts.index
    counts = attack_counts.values
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    bar_chart = axes[0].bar(categories, counts, color='skyblue')
    axes[0].set_title('Attack Category Distribution')
    axes[0].set_xlabel('Attack Category')
    axes[0].set_ylabel('Count')
    axes[0].set_xticklabels(categories, rotation=45, ha='right')
    for bar in bar_chart:
        height = bar.get_height()
        axes[0].text(bar.get_x() + bar.get_width() / 2, height, f'{height}', ha='center', va='bottom')

    wedges, texts, autotexts = axes[1].pie(counts, labels=categories, autopct='%1.1f%%', startangle=90,
                                           wedgeprops=dict(width=0.4, edgecolor='w'), pctdistance=0.80)
    plt.setp(autotexts, size=10, weight='bold')
    axes[1].set_title('Attack Category Proportion')
    centre_circle = plt.Circle((0, 0), 0.25, color='white', edgecolor='black', linewidth=0.5)
    axes[1].add_artist(centre_circle)
    plt.tight_layout()
    plt.show()
```

Figure 7: Attack Category Distribution

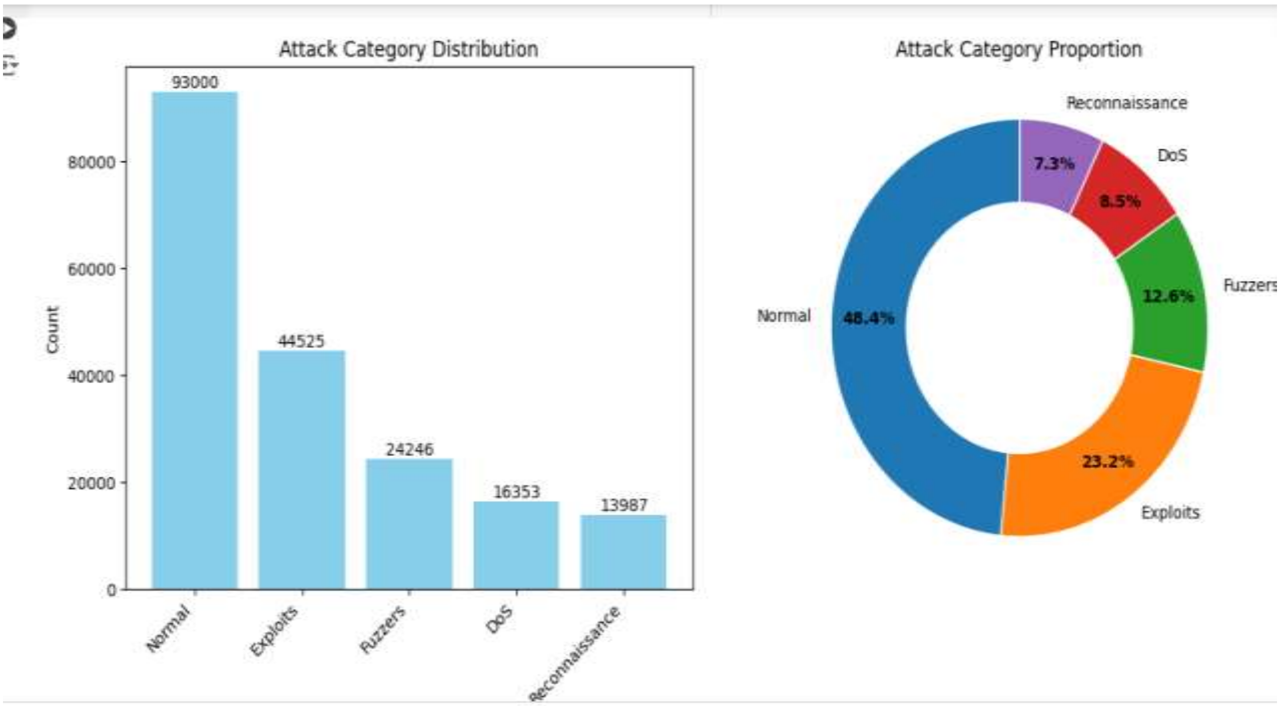


Figure 8: Visualization of Attack Category Distribution

```
[ ] attack_counts = df['service'].value_counts()
    categories = attack_counts.index
    counts = attack_counts.values

    with plt.style.context(style="fivethirtyeight"):
        plt.figure(figsize=(18,8))
        plt.rcParams['font.size'] = 15
        bar_chart = plt.bar(categories, counts, color='skyblue',)
        plt.title('Service Category Distribution')
        plt.xlabel('Service Category')
        plt.ylabel('Count')
        for bar in bar_chart:
            height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height}', ha='center', va='bottom')
        plt.show()
```

Figure 9: Service Category Distribution

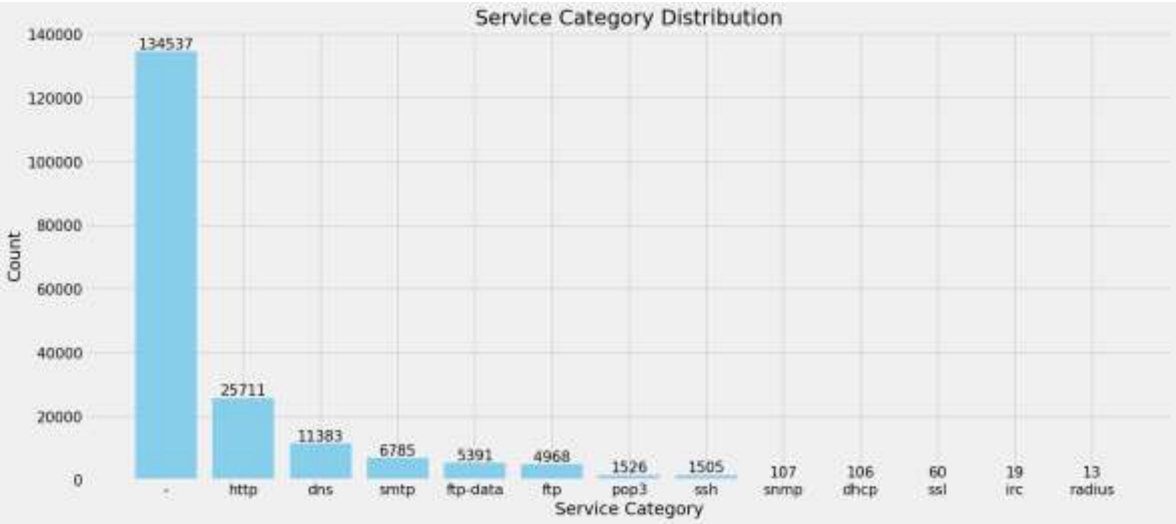


Figure 10: Visualization of Service Category Distribution

4.5 Scaling Features Using MinMaxScaler

Each feature is scaled by the MinMaxScaler to a specific range, usually between 0 and 1. This scaling is especially important to algorithms that are sensitive to feature scale, since it helps standardize features to ensure all contributes equally to the model (Figure 11).

```
[ ] scaler = MinMaxScaler()
    scaler = scaler.fit(X.values)

    X_scaled = scaler.transform(X.values)
    binary_df = pd.DataFrame(data=X_scaled, columns=X.columns)

    binary_df.head()
```

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload
0	1.500000e-07	0.909091	1.000000	0.5	0.000094	0.000000	0.000012	0.000000	0.111111	0.996078	0.000000	1.484450e-02
1	3.229819e-02	0.856061	1.000000	0.4	0.001597	0.001634	0.000067	0.000181	0.000018	0.243137	0.992126	6.459790e-07
2	1.666667e-07	0.909091	1.000000	0.5	0.000094	0.000000	0.000012	0.000000	0.100000	0.996078	0.000000	1.336005e-02
3	8.134501e-04	0.856061	0.166667	0.4	0.004791	0.004901	0.000203	0.000255	0.002151	0.121569	0.114173	7.878016e-05
4	1.840144e-02	0.856061	0.166667	0.4	0.001033	0.001089	0.000042	0.000047	0.000021	0.996078	0.992126	7.042522e-07

Figure 11: Feature Scaling of Dataset Using Min-Max Scaler

4.6 Plotting Feature Importance Using Line Chart

Feature selection and analysis of feature importance for both binary and multiclass classification tasks are performed in this section. The most important features that significantly impact classification outcomes were identified. This process ensures that only the most influential features are retained, with features having negative or missing correlation values being removed. Figure 12 and Figure 13 shows the feature Selection for Binary Classification and Multiclass Classification.

```
corr_df = corr_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)
corr_df = corr_df.dropna()
corr_df = corr_df.loc[corr_df['Importance'] >= -0.118324]
corr_df
```

1	ct_state_ttl	0.378927
2	state	0.352153
3	rate	0.250590
4	dttl	0.232449
5	sload	0.145450
6	ct_dst_sport_ltm	0.137669
7	ackdat	0.128211
8	tcprtt	0.101027
9	dur	0.084857
10	service	0.070108
11	smean	0.068975
12	synack	0.064648
13	trans_depth	0.036346
14	sbytes	0.034919
15	sloss	0.022528
16	is_ftp_login	0.021812
17	ct_flw_http_mthd	0.021701

Figure 12: Feature Selection for Binary Classification

```
corr_df = corr_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)
corr_df = corr_df.dropna()
corr_df = corr_df.loc[corr_df['Importance'] >= -0.089132]
corr_df
```

3	proto	0.213793
4	ackdat	0.207249
5	dwin	0.192081
6	swin	0.192063
7	dttl	0.188320
8	ct_srv_src	0.177802
9	service	0.164420
10	dtcpb	0.145886
11	stcpb	0.145642
12	ct_srv_dst	0.134899
13	sjit	0.130426
14	dinpkt	0.128180
15	sinpkt	0.089132
16	smean	0.084700
17	sload	0.045156
18	ct_src_dport_ltm	0.034136
19	djit	0.026128
20	ct_dst_ltm	0.024634

Figure 13: Feature Selection for Multiclass Classification

4.7 Dataset Oversampling

The SMOTE (Synthetic Minority Over-Sampling Technique) algorithm is applied in Figure 14 and Figure 15 code to address the issue of class imbalance in a binary and multiclass classification dataset.

```
[ ] %%time
    binary_adasyn = SMOTE(random_state=seed_value)
    binary_X_resampled, binary_y_resampled = binary_adasyn.fit_resample(binary_X.values, binary_y.values.ravel())

    binary_df = pd.DataFrame(data=binary_X_resampled, columns=binary_X.columns)
    binary_df['label'] = binary_y_resampled

    binary_df.head()

    CPU times: user 56.5 s, sys: 133 ms, total: 56.6 s
    Wall time: 57 s
```

	sttl	ct_state_ttl	state	rate	dttl	sload	ct_dst_sport_ltm	ackdat	tcprtt	dur
0	0.996078	0.333333	0.5	0.111111	0.000000	1.484450e-02	0.022222	0.000000	0.000000	1.500000e-07
1	0.243137	0.166667	0.4	0.000018	0.992126	6.459790e-07	0.000000	0.024859	0.041620	3.229819e-02
2	0.996078	0.333333	0.5	0.100000	0.000000	1.336005e-02	0.088889	0.000000	0.000000	1.666667e-07
3	0.121569	0.000000	0.4	0.002151	0.114173	7.878016e-05	0.000000	0.000040	0.000154	8.134501e-04
4	0.996078	0.166667	0.4	0.000021	0.992126	7.042522e-07	0.000000	0.016648	0.034323	1.840144e-02

Figure 14: Dataset oversampling for binary classification

```
%%time
multiclass_adasyn = SMOTE(random_state=seed_value)
multiclass_X_resampled, multiclass_y_resampled = multiclass_adasyn.fit_resample(multiclass_X.values, multiclass_y.values.ravel()

multiclass_df = pd.DataFrame(data=multiclass_X_resampled, columns=multiclass_X.columns)
multiclass_df['attack_cat'] = multiclass_y_resampled

multiclass_df.head()

CPU times: user 8.03 s, sys: 42.6 ms, total: 8.07 s
Wall time: 8.16 s
```

	sttl	synack	tcprtt	proto	ackdat	dwin	swin	dttl	ct_srv_src	service	dtcpb	stcpb	ct_srv_dst	s
0	0.996078	0.000000	0.000000	0.909091	0.000000	0.0	0.0	0.000000	0.064516	1.000000	0.000000	0.000000	0.065574	0.000
1	0.243137	0.026727	0.041620	0.856061	0.024859	1.0	1.0	0.992126	0.000000	1.000000	0.433494	0.053225	0.000000	0.004
2	0.996078	0.000000	0.000000	0.909091	0.000000	0.0	0.0	0.000000	0.129032	1.000000	0.000000	0.000000	0.131148	0.000
3	0.996078	0.025538	0.034323	0.856061	0.016648	1.0	1.0	0.992126	0.032258	0.166667	0.834802	0.448724	0.016393	0.003
4	0.996078	0.000000	0.000000	0.030303	0.000000	0.0	0.0	0.000000	0.145161	1.000000	0.000000	0.000000	0.147541	0.000

Figure 15: Dataset oversampling for multiclass classification

4.8 Dataset Splitting

Figures 16 illustrate dataset split into 80% training and 20% testing

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

    (158577, 33) (39645, 33) (158577, 2) (39645, 2)
```

Figure 16: Dataset Splitting Process

5. Model Architecture

5.1 Artificial Neural Network

The sequential network model of the Artificial Neural Network (ANN) is shown in Figure17, which highlights the complexity and structure of the model by providing an overview of the number of parameters and the output shape of each layer.

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	2,176
batch_normalization (BatchNormalization)	(None, 64)	256
dense_1 (Dense)	(None, 128)	8,320
batch_normalization_1 (BatchNormalization)	(None, 128)	512
dense_2 (Dense)	(None, 128)	16,512
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dense_3 (Dense)	(None, 128)	16,512
batch_normalization_3 (BatchNormalization)	(None, 128)	512
dense_4 (Dense)	(None, 256)	33,024
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dense_5 (Dense)	(None, 2)	514

```
Total params: 79,874 (312.01 KB)
Trainable params: 78,466 (306.51 KB)
Non-trainable params: 1,408 (5.50 KB)
```

Figure 17: Model Architecture of Artificial Neural Network

5.2 Convolutional-Gated Recurrent Unit

The hybrid model, Convolutional-Gated Recurrent Unit (CGRU) algorithm is outlined in the model summary of Figure 18. It combines convolutional and recurrent layers to process sequential data with complex feature dependencies effectively.

```
model.summary()

 Layer (type)                  Output Shape        Param #    Connected to
==========================================================================================
 input_1 (InputLayer)         [(None, 26, 1)]     0          []

 conv1d (Conv1D)              (None, 26, 32)      96         ['input_1[0][0]']

 conv1d_1 (Conv1D)            (None, 26, 32)      2080       ['conv1d[0][0]']

 conv1d_2 (Conv1D)            (None, 26, 32)      64         ['input_1[0][0]']

 add (Add)                    (None, 26, 32)      0          ['conv1d_1[0][0]',
                                                              'conv1d_2[0][0]']

 batch_normalization (BatchNorm  (None, 26, 32)   128        ['add[0][0]']
 alization)

 max_pooling1d (MaxPooling1D)  (None, 13, 32)     0          ['batch_normalization[0][0]']

 conv1d_3 (Conv1D)            (None, 13, 64)      4160       ['max_pooling1d[0][0]']

 conv1d_4 (Conv1D)            (None, 13, 64)      8256       ['conv1d_3[0][0]']

 conv1d_5 (Conv1D)            (None, 13, 64)      4160       ['conv1d_4[0][0]']

 add_1 (Add)                  (None, 13, 64)      0          ['conv1d_4[0][0]',
                                                              'conv1d_5[0][0]']

 batch_normalization_1 (BatchNo  (None, 13, 64)   256        ['add_1[0][0]']
 rmalization)

 max_pooling1d_1 (MaxPooling1D)  (None, 6, 64)    0          ['batch_normalization_1[0][0]']

 conv1d_6 (Conv1D)            (None, 6, 128)      16512      ['max_pooling1d_1[0][0]']

 conv1d_7 (Conv1D)            (None, 6, 128)      32896      ['conv1d_6[0][0]']

 conv1d_8 (Conv1D)            (None, 6, 128)      16512      ['conv1d_7[0][0]']
```

Figure 18: Model Architecture of Convolutional-Gated Recurrent Unit

6. Performance Evaluation

6.1 Performance Metrics for Binary Classification

Performance Metrics for Binary Classification Using ANN are illustrated in Figures 19 and Figure 20.

```
[ ] model_accuracy = accuracy_score(
        y_true=true_labels,
        y_pred=predicted_labels
    )

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

    Validation accuracy of ArtificialNeuralNetwork model is 90.79%
```

Figure 19: Accuracy Achieved for Binary Classification

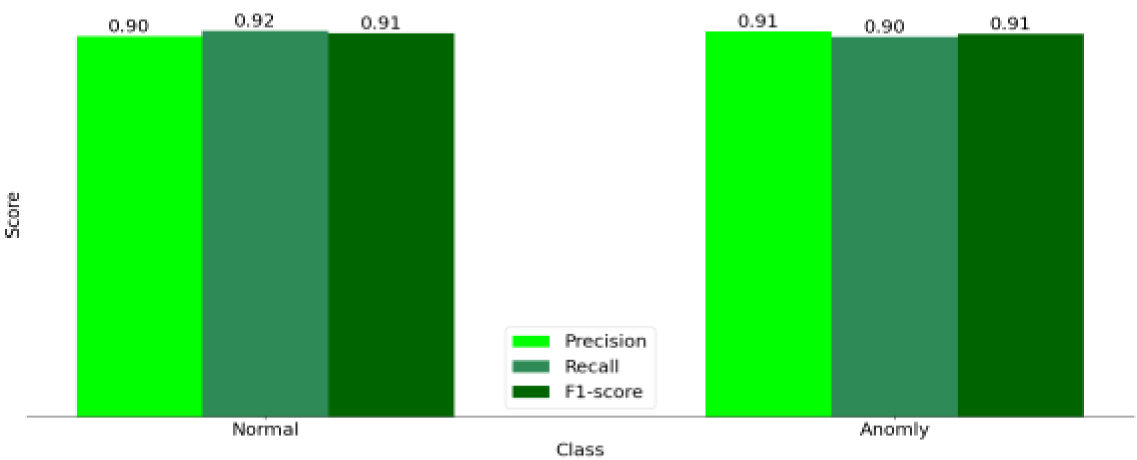


Figure 20: Performance Metrics for Binary Classification

6.2 Performance Metrics for Multiclass Classification

Performance Metrics of the Hybrid CGRU Model for Multiclass Classification are illustrated in Figures 21 and Table 1.

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

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

```
Validation accuracy of ConvolutionalGatedRecurrentUnit model is 77.99%
```

Figure 21: Accuracy Achieved for Multiclass Classification

Class	Precision	Recall	F1-Score	Support
DoS	0.76	0.46	0.57	6679
Exploits	0.92	0.88	0.90	6679
Reconnaissance	0.55	0.84	0.67	6678
Fuzzers	0.86	0.81	0.83	6679
Accuracy			0.74	26715
Macro Avg	0.77	0.74	0.74	26715
Weighted Avg	0.77	0.74	0.74	26715

Table 1: Performance Metrics for Multiclass Classification

7. Intrusion Detection Application

A Python Flask web interface deployed in Amazon Web Service (AWS) Elastic Beanstalk (Figure 22) and (Figure 23), allows users to upload a CSV file with network traffic data.

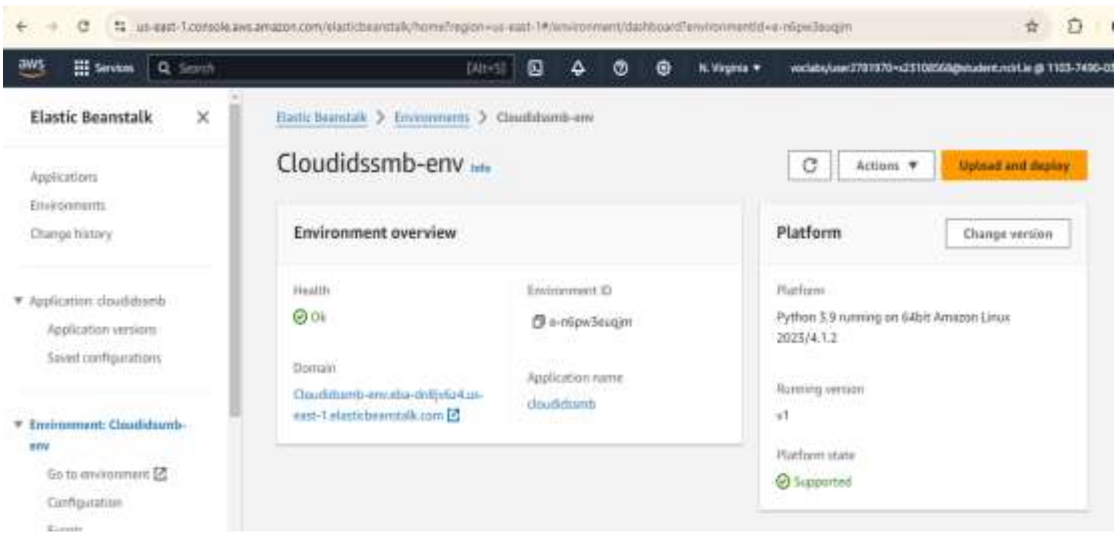


Figure 22: Application deployed in AWS Elastic Beanstalk

```
def predict_response(filepath):
    # print("filepath : ", filepath)
    phase_1_status = phase_1_verification(filepath)
    print("phase_1_status : ", phase_1_status)
    if phase_1_status['STATUS'] == True:
        phase_1_ip_address = phase_1_status["IP ADDRESS"]
        phase_1_attack = phase_1_status["ATTACK"]

        B_label = f"The IP address {phase_1_ip_address} is blocked."
        B_ip = f"Attack details: {phase_1_attack}"
        return B_label, B_ip, phase_1_attack
    else:
        result_2 = phase_2_verification(filepath)
        print("result_2 : ", result_2)

        binaryClass = result_2[0]
        binaryClassPro = result_2[1]
        multiClass = result_2[2]
        multiClassPro = result_2[3]

        bin_pro_score = "{:.2f}%".format(binaryClassPro * 100)

        if binaryClass == 'Normal':
            binaryClassStatus = f"File Status: {binaryClass}"
            multiClassStatusPro = f"Probability Score: {bin_pro_score}"
            predictionResult = binaryClass
        else:
            mul_pro_score = "{:.2f}%".format(multiClassPro * 100)
            binaryClassStatus = f"File Status: {binaryClass}"
            multiClassStatusPro = f"A {multiClass} attack was detected with a probability score of {mul_p
            predictionResult = multiClass
```

Figure 23: Python Flask IDS Application Code

Result will be displayed in the portal whether it is normal or attack (Figure 24), (Figure 25) and Also display the blocked IP address (figure 26).

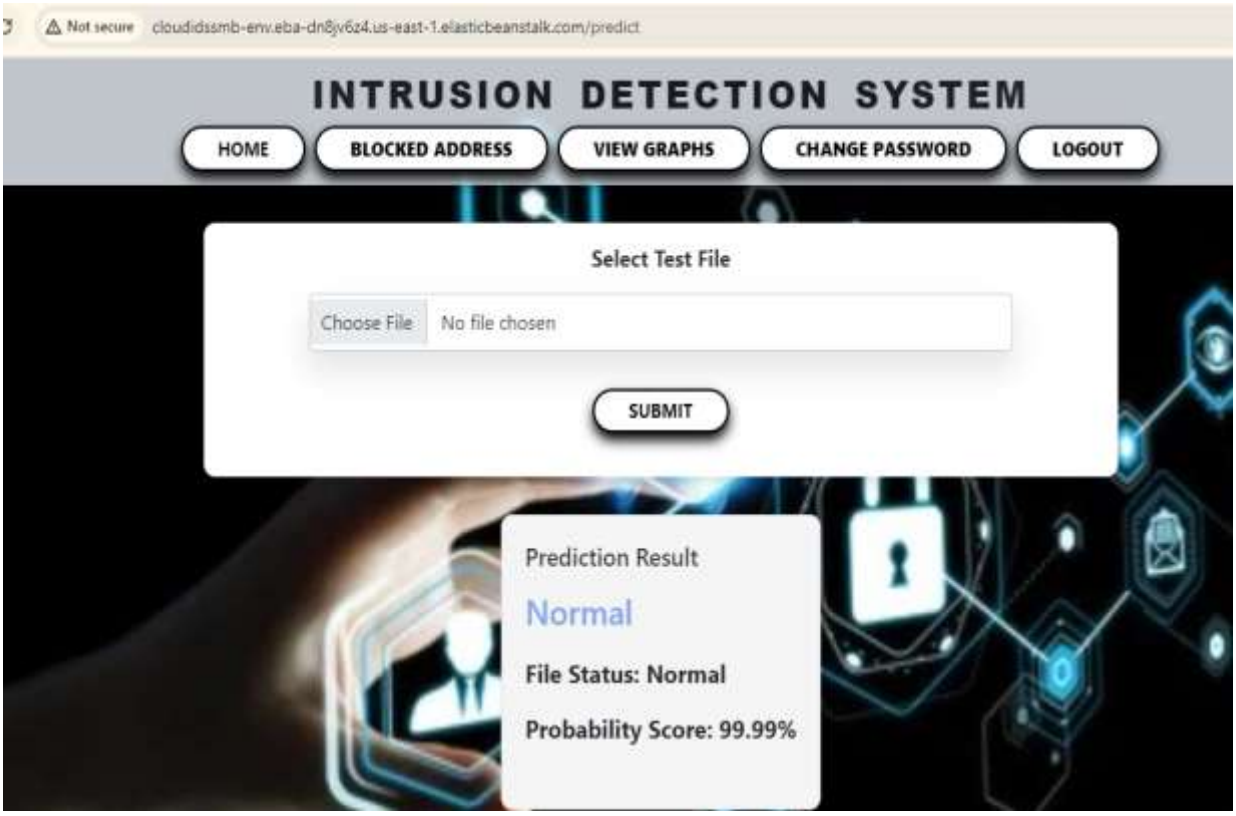


Figure 24: Application Detecting Normal Activity

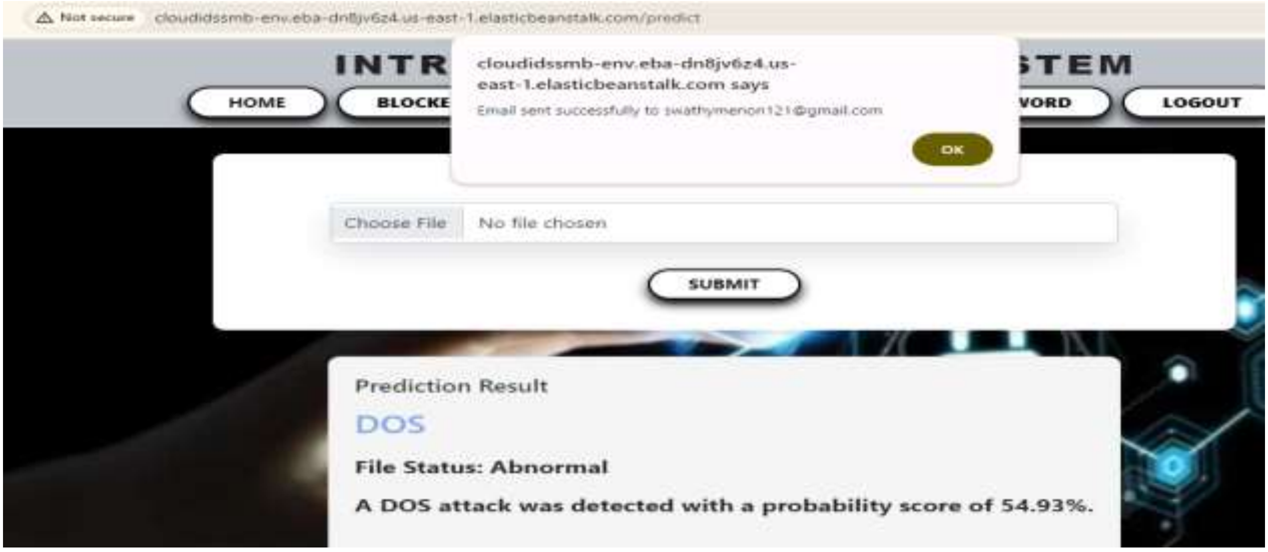


Figure 25: Application Detecting Attack and Sent an email notification

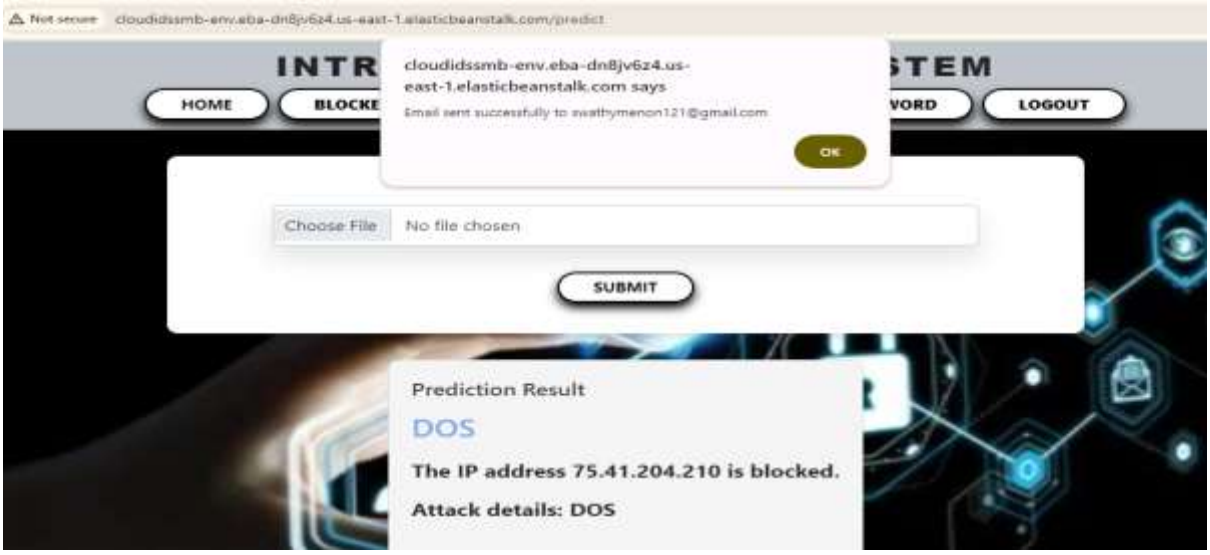


Figure 26: Application Detecting Blocked IP

8. Conclusion

This research proposes a two-level classification system for IoT intrusion detection by combining ANN for initial binary classification and hybrid CNN-GRU model for in-depth threat detection. By optimizing resource utilization and computational efficiency, this approach significantly improves response times and detection accuracy. The combination of these machine learning techniques ensures an efficient and scalable solution for enhancing the security and reliability of IOT networks.

References

Data World, D.W. (2024) *Useful-Unsw-Nb15-Data - Dataset by Victorpuli*. *data.world*. Available at: https://data.world/victorpuli/useful-unsw-nb15-data (Accessed: 7 August 2024).