National College of Ireland

Configuration Manual

MSc Research Project
MSc. Cyber Security

Jay Chauhan
Student ID: 22137092

School of Computing
National College of Ireland

Supervisor: Prof. Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jay Manishkumar Chauhan
Student ID:	22137092
Programme:	MSc. Cyber Security **Year:** 2023-2024
Module:	MSc Research Project
Lecturer:	Prof. Michael Pantridge
Submission Due Date:	14/12/2023
Project Title:	Optimizing Rogue Access Point Detection with CART and Deep Learning Techniques
Word Count:	1055 **Page Count:** 11

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:	Jay Manishkumar Chauhan
Date:	13/12/2023

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

Jay Chauhan
22137092

1 Introduction

This manual provides details about the proposed model solution's setup and requirements such as technical specifications and software specification to recreate the model from the scratch. Moreover, this document also explains how to implement the algorithms to develop the proposed solution.

2 System Configuration

The proposed solution was developed using 2 different systems.

1. **Data pre-processing and Feature extraction:**
   The dataset pre-processing including clean-up, merging, and label encoding was performed on the local system. Feature extraction and reduction also performed on the local system. The configuration of the system is shown below:
   - Processor: 8 core Ryzen 7 4000
   - RAM: 16 GB
   - OS: Windows 11
   - GPU: 4 GB Nvidia RTX 3050
   - Hard Drive: 512 GB
2. **Model training and development:**
   The proposed solution uses various machine learning, deep learning and ensemble models. These model training and development requires high processing power. For that reason, we have used GCP instance. The configuration of the GCP instance is shown below:
   - Processor: 16 core N1
   - RAM: 64 GB
   - OS: Ubuntu 20.06
   - Hard Drive: 256 GB

3 Software Tools

The following tools and libraries were used to develop the proposed solution:

1. **Tools for Data pre-processing:**
   - Anaconda with Jupyter Notebook
   - Python 3.12.0
   - Pandas 2.1.2

- Numpy 1.26.1

Figure 1. installed numpy and pandas

Anaconda was used to access the jupyter notebook. Pandas and Numpy libraries were used for dataset pre-processing.

2. Tools for model development:

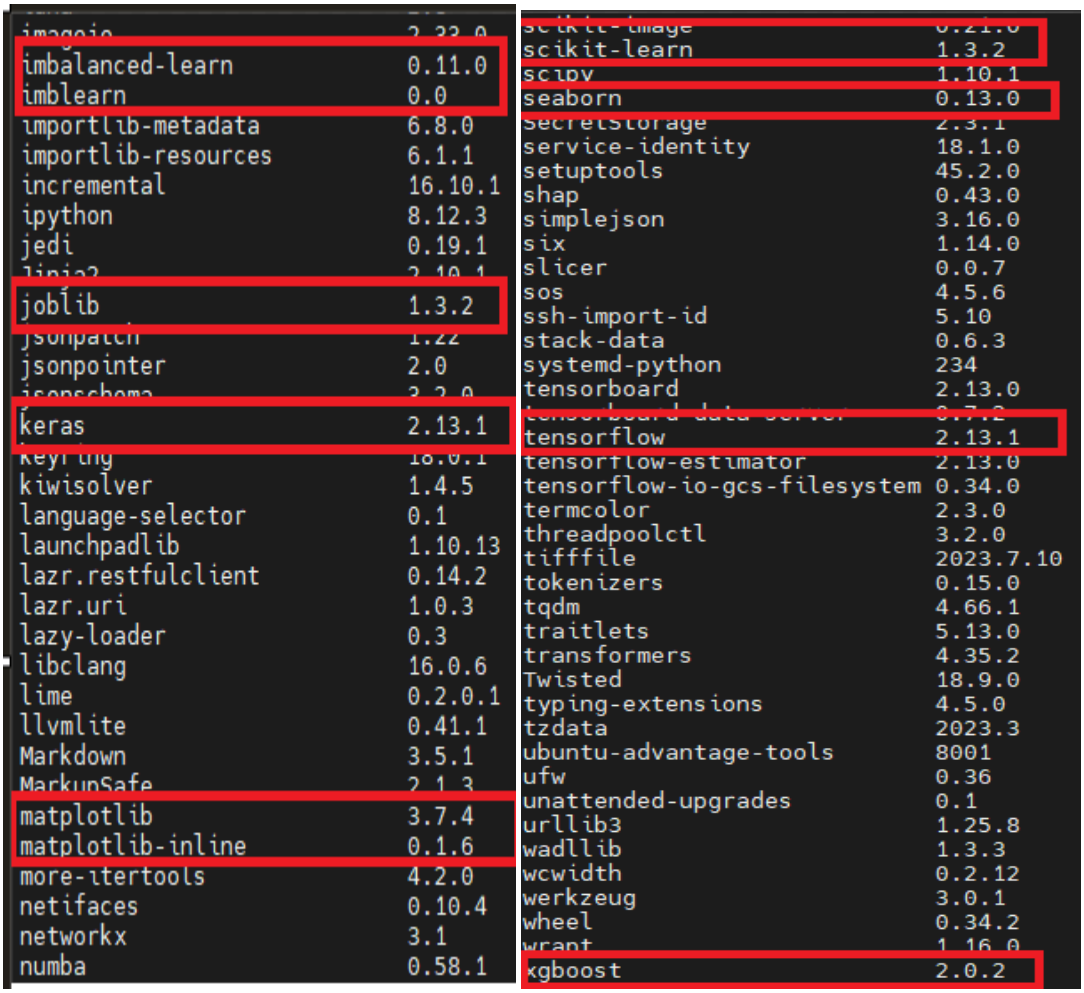


Figure 2. Installed Libraries

- Python 3.8.10
- Imbalanced-learn 0.11.0: To perform SMOTE technique
- Joblib 1.3.2: To save the trained model
- Keras 2.13.1 & Tensorflow 2.13.1: To train deep learning model FCNN
- Matplotlib 3.7.4 & seaborn0.13.0: To visualize the results and configuration matrix
- Scikit-learn 1.3.2: To train machine learning model CART
- XGBoost: To train XGBoost model for ensemble.

The above tools and libraries were used to develop machine and deep learning model. Python 3.8.10 version was used as TensorFlow library does not support latest python version. The matplotlib and seaborn libraries were used to visualize the data.

3. **Tools for Web application development:**
    - Flask 3.0.0: framework was used as the backend of the web-application.
    - VSCode: text editor
    - HTML, CSS, JavaScript used for frontend of the web application.
    - GCP app engine to deploy the application on the GCP.

4 Implementation Steps

After installing required software and tools for data pre-processing and model development. The implementation take place as follows:

Dataset Acquisition: The AWID 3 dataset were downloaded from the official site (ref) and full permissions were given in mail to utilize the dataset for the research purpose.

Dataset pre-processing: The dataset has range of attack data scenario. The attacks include deauth, Disas, (Re)assoc, Rogue_AP, Krack, Kr00k, SSH, Botnet, Malware, SQL injection, SSDP, Evil Twin, and Website Spoofing. From these attacks only 4 attacks (Deauth, Reassoc, Rogue_AP, Evil_Twin) data were chosen. These attacks were chosen because, Deauth attacks break the communication between client and access point, rogue_AP attacks involve setting up an unauthorized access point to intercept traffic. Evil twin is similar to rogue AP, setting up a fraudulent wifi access point. Reassoc attacks involve unauthorized attempts to associate or reassociate with legitimate access point.

Data clean-up and merging: The csv files are categorized attack wise in a folder and each attack folder has multiple csv files.

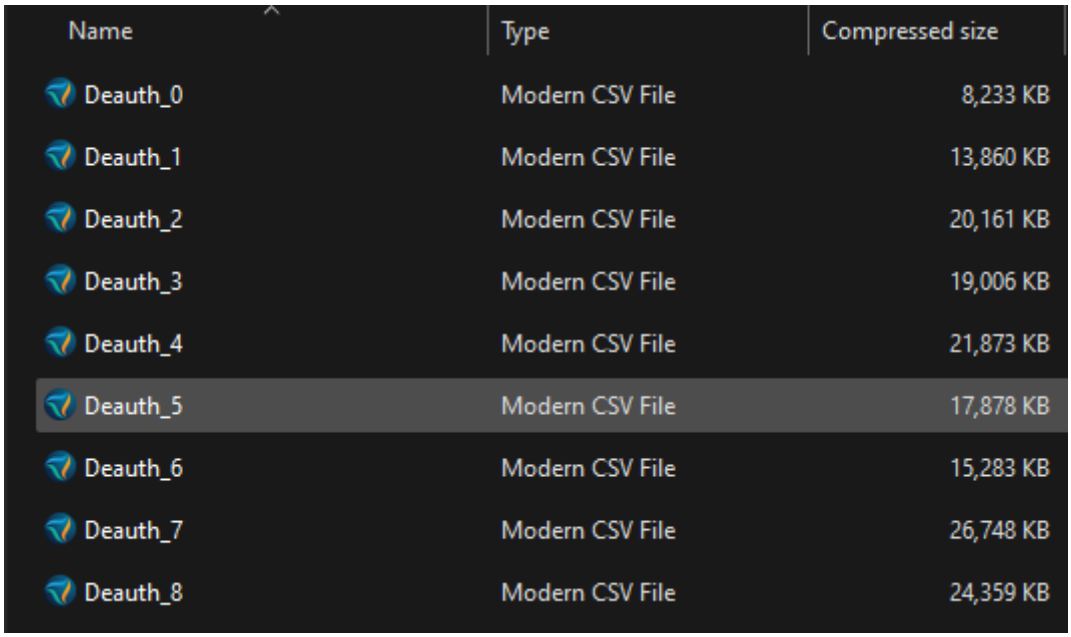


Figure 3. Separated CSV files

The files are cleaned and merged using below code. After merging and cleaning each attack folder has only one merged file.

```python
import pandas as pd
import os

directory_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\deauth'

# Loop through each file in the directory
for filename in os.listdir(directory_path):
    if filename.endswith('.csv'):  # Check if the file is a CSV
        file_path = os.path.join(directory_path, filename)

        # Read the CSV file
        data = pd.read_csv(file_path)

        # Discard columns with any null values
        data_cleaned_columns = data.dropna(axis=1, how='any')

        # Discard rows with any null values
        data_cleaned = data_cleaned_columns.dropna(axis=0, how='any')

        # Define the path for the cleaned file
        cleaned_file_path = os.path.join(directory_path, f'cleaned_{filename}')

        # Save the cleaned data to a new CSV file
        data_cleaned.to_csv(cleaned_file_path, index=False)

        print(f'Cleaned data saved to {cleaned_file_path}')

print('Data cleanup completed for all files.')
```

Figure 4. Null rows and columns clean up

```python
import pandas as pd
import os

# Directory where your cleaned CSV files are located
directory_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\deauth\cleaned'

# List to store each DataFrame
dataframes_list = []

# Loop through each file in the directory
for filename in os.listdir(directory_path):
    if filename.endswith('.csv'):  # Check if the file is a CSV
        file_path = os.path.join(directory_path, filename)

        # Read the CSV file and append it to the list
        df = pd.read_csv(file_path)
        dataframes_list.append(df)

# Concatenate all DataFrames in the list
merged_df = pd.concat(dataframes_list, ignore_index=True)

# Save the merged DataFrame to a new CSV file
output_file_path = os.path.join(directory_path, 'merged_cleaned_data.csv')
merged_df.to_csv(output_file_path, index=False)

print(f'Merged CSV file saved to {output_file_path}')
```

Figure 5. CSV files merged

Label encoding: After merging the csv file, the label column which has both type of data, attack traffic and normal traffic need to be encoded in 0 and 1. Normal traffic encoded in 0 and attack traffic encoded in 1.

```python
import pandas as pd

# File path of your merged CSV file
file_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\rogueap\cleaned\merged_cleaned_data.csv'
chunk_size = 50000
chunks = []

for chunk in pd.read_csv(file_path, chunksize=chunk_size, low_memory=False):
    # Replace 'Normal' with 0 and 'Deauth' with 1 in the 'Label' column
    chunk['Label'] = chunk['Label'].replace({'Normal': 0, 'RogueAP': 1})
    chunks.append(chunk)

# Concatenate all chunks back into a single DataFrame
df = pd.concat(chunks, ignore_index=True)

# Save the modified DataFrame back to a CSV file
output_file_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\rogueap\cleaned\merged_replaced_data.csv'
df.to_csv(output_file_path, index=False)

print(f'Modified CSV file saved to {output_file_path}')
```

Figure 6. Label Encoding

After label encoding now each merged and replaced attack file needs to be merged into one big CSV file.

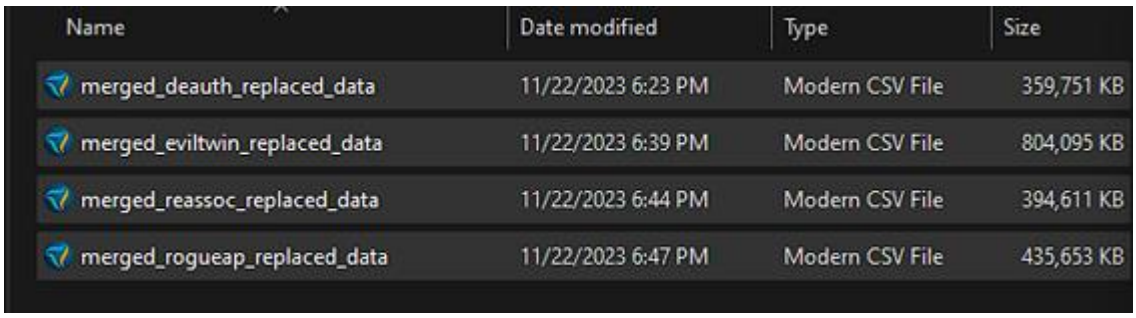


Figure 7. All merged CSV files

Again using the same code as shown above these 4 files merged into one CSV file with 1.9 GB size.

Feature Extraction: For feature extraction ANOVA test is used. The dependent variable “label ” is tested on other independent variable. This test identifies features, which has the higher variance score and effects the value of label column.

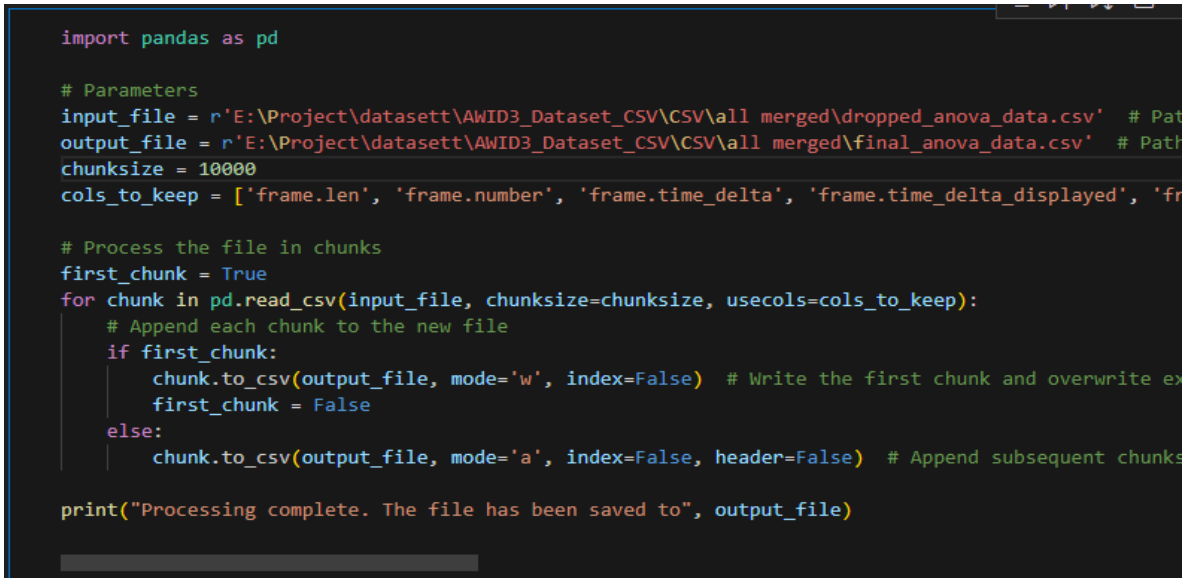
Using below code we found the features list which are effecting the label column. From 20 important features, only top 12 features were picked because of easier training and model deployment.

```python
import pandas as pd
from scipy import stats

# Path to your CSV file
file_path =  r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\merged_data.csv'

# Initialize lists to store data from each chunk
group0 = {}
group1 = {}

# Define the chunk size
chunk_size = 100000
# Read the CSV file in chunks
for chunk in pd.read_csv(file_path, chunksize=chunk_size,low_memory=False):
    # Filter the data based on the label
    group0_chunk = chunk[chunk['Label'] == 0]
    group1_chunk = chunk[chunk['Label'] == 1]

    # Iterate over each column (excluding 'label') and collect numeric data
    for column in group0_chunk.columns.drop('Label'):
        if pd.api.types.is_numeric_dtype(group0_chunk[column]):
            if column not in group0:
                group0[column] = []
                group1[column] = []
            group0[column].extend(group0_chunk[column].dropna().tolist())
            group1[column].extend(group1_chunk[column].dropna().tolist())

# Prepare a list to store ANOVA results
anova_results_list = []

# Perform ANOVA for each numeric column and store results
for column in group0:
    f_value, p_value = stats.f_oneway(group0[column], group1[column])
    anova_results_list.append({'Column': column, 'F-Value': f_value, 'P-Value': p_value})

# Convert the list to a DataFrame
anova_results = pd.DataFrame(anova_results_list)

# Save the results to a CSV file
anova_results.to_csv('anova_results.csv', index=False)

print("ANOVA results saved to 'anova_results.csv'")
```

Figure 8. Feature Extraction with ANOVA

These 12 features were found most important for prediction of label column. 'frame.len', 'frame.number','frame.time_delta','frame.time_delta_displayed','frame.time_epoch','frame.time_relative','radiotap.length','radiotap.timestamp.ts','wlan.fc.protected','wlan.fc.retry','wlan.fc.subtype', and 'wlan_radio.duration'. After feature extraction, data reduction was performed.

Data Reduction: Unnecessary data causes poor model development, for this reason only important feature kept in the dataset and other features were dropped. Using below code, features were dropped.



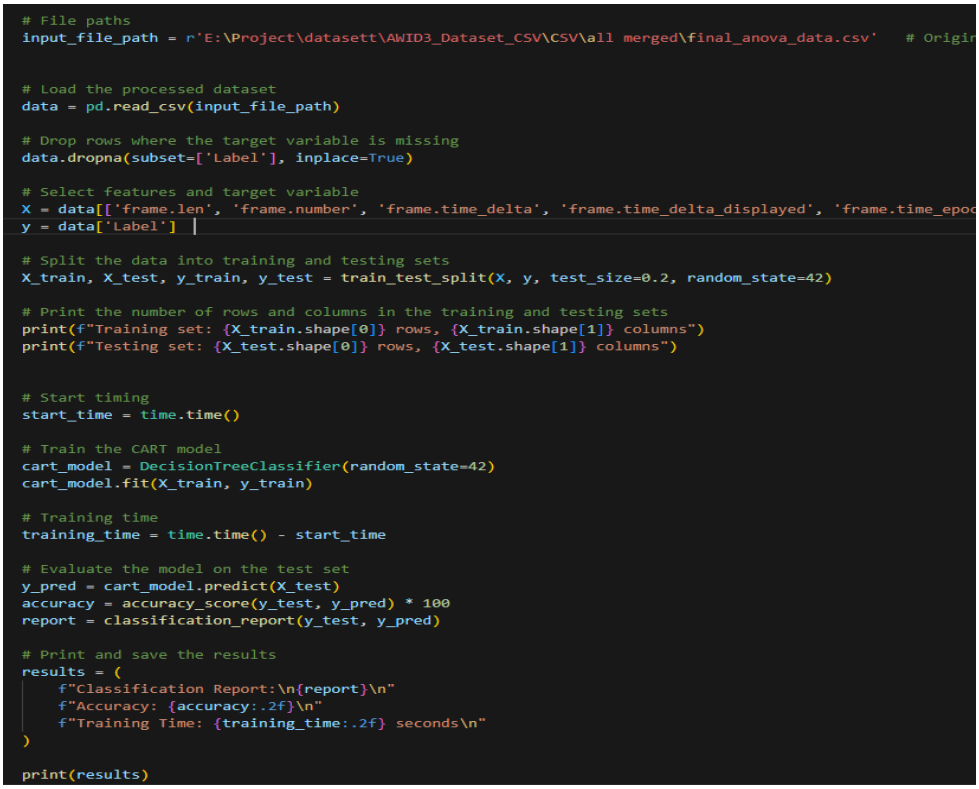
```python
import pandas as pd

# Parameters
input_file = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\dropped_anova_data.csv'  # Pat
output_file = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\final_anova_data.csv'  # Path
chunksize = 10000
cols_to_keep = ['frame.len', 'frame.number', 'frame.time_delta', 'frame.time_delta_displayed', 'fr

# Process the file in chunks
first_chunk = True
for chunk in pd.read_csv(input_file, chunksize=chunksize, usecols=cols_to_keep):
    # Append each chunk to the new file
    if first_chunk:
        chunk.to_csv(output_file, mode='w', index=False)  # Write the first chunk and overwrite ex
        first_chunk = False
    else:
        chunk.to_csv(output_file, mode='a', index=False, header=False)  # Append subsequent chunks

print("Processing complete. The file has been saved to", output_file)
```

Figure 9. Data Reduction

After dropping the unneccessary features now dataset preprocessing is finished. Dataset is ready for model development. Before datset preprocessing datset had 255 columns, but after clean up and feature extraction, dataset has 13 column. 12 columns of features, and 1 column of target.

Model Devlopment:

1. **CART:** Classification and regression tree algorithm is trained using below code. All the required packages were loaded first, and then model trained



```python
# File paths
input_file_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\final_anova_data.csv'  # Origin


# Load the processed dataset
data = pd.read_csv(input_file_path)

# Drop rows where the target variable is missing
data.dropna(subset=['Label'], inplace=True)

# Select features and target variable
X = data[['frame.len', 'frame.number', 'frame.time_delta', 'frame.time_delta_displayed', 'frame.time_epoc
y = data['Label']

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

# Print the number of rows and columns in the training and testing sets
print(f"Training set: {X_train.shape[0]} rows, {X_train.shape[1]} columns")
print(f"Testing set: {X_test.shape[0]} rows, {X_test.shape[1]} columns")


# Start timing
start_time = time.time()

# Train the CART model
cart_model = DecisionTreeClassifier(random_state=42)
cart_model.fit(X_train, y_train)

# Training time
training_time = time.time() - start_time

# Evaluate the model on the test set
y_pred = cart_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) * 100
report = classification_report(y_test, y_pred)

# Print and save the results
results = (
    f"Classification Report:\n{report}\n"
    f"Accuracy: {accuracy:.2f}\n"
    f"Training Time: {training_time:.2f} seconds\n"
)

print(results)
```

Figure 10. CART Model Training

2. **FCNN:** The deep learning model were trained using below code. This code also shows that SMOTE is performed before training the model to deal with imbalanced dataset.

```
imputer = SimpleImputer(strategy='median')

# Initialize the model
model = Sequential([
    Dense(128, activation='relu', input_dim=12),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Load the processed dataset
data = pd.read_csv(csv_file_path)

# Drop rows with NaN in the target column and impute NaN values in features
data = data.dropna(subset=['Label'])
data.iloc[:, :-1] = imputer.fit_transform(data.iloc[:, :-1])

# Splitting the data
X = data.drop('Label', axis=1)
y = data['Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Start timing the training process
start_training_time = time.time()

# Train the model
model.fit(X_train_smote, y_train_smote, epochs=30, batch_size=256, verbose=0)

# Calculate total training time
total_training_time = time.time() - start_training_time
print(f"Total Training Time: {total_training_time:.2f} seconds")

# Evaluate the model
predictions = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, predictions)
recall = recall_score(y_test, predictions, average='macro')
f1 = f1_score(y_test, predictions, average='macro')

print("Model Performance:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")

# Save the trained model
model.save('final_trained_model.h5')
print("Model saved as 'final_trained_model.h5'")
```

Figure 11. FCNN Model Training

3. **Ensemble model:** Parallel ensemble method is used here. The ensemble model is trained using both model FCNN + XGBoost. XGBoost model will be trained here as FCNN model was trained above.

XGBoost:

```python
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib
import pandas as pd

data = pd.read_csv(r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\final_anova_data.csv')
X = data.drop('Label', axis=1)  # Features
y = data['Label']  # Target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='binary:logistic')

# Train the model
xgb_model.fit(X_train, y_train)

# Evaluate the model
predictions = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
joblib.dump(xgb_model, 'xgb_model.pkl')
print(f"XGBoost Model Accuracy: {accuracy:.2f}")
```

Figure 12. XGBoost Model Training

Ensemble model:

```python
# Load your dataset
csv_file_path = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\final_anova_data.csv'
data = pd.read_csv(csv_file_path)

# Split the data into features and target
X = data.drop('Label', axis=1)
y = data['Label']

# First split:
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Second split:
X_train, X_valid, y_train, y_valid = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid)
X_test_scaled = scaler.transform(X_test)

# trained deep learning model
deep_model = load_model('final_trained_model.h5')  # Replace with your model's path

# trained XGBoost model
xgb_model = joblib.load('xgb_model.pkl')  # Replace with your model's path

# Generate predictions for stacking
dl_predictions_valid = deep_model.predict(X_valid_scaled)
xgb_predictions_valid = xgb_model.predict_proba(X_valid_scaled)[:, 1]

# Stack predictions
stacked_features_valid = np.column_stack((dl_predictions_valid, xgb_predictions_valid))
start_time = time.time()

# Train the meta-model
meta_model = LogisticRegression()
meta_model.fit(stacked_features_valid, y_valid)
end_time = time.time()

# Generate predictions for evaluation
dl_predictions_test = deep_model.predict(X_test_scaled)
xgb_predictions_test = xgb_model.predict_proba(X_test_scaled)[:, 1]
stacked_features_test = np.column_stack((dl_predictions_test, xgb_predictions_test))

# Final predictions by the meta-model
final_predictions = meta_model.predict(stacked_features_test)
```

Figure 13. Ensemble Model Training

Confusion matrix: Confusion matrix is an important part of the evaluation. It provides the model performance. Below code was used to produced confusion matrix

```python
import pandas as pd
import joblib
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load CART model
model_file = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\cart.pkl'
cart_model = joblib.load(model_file)

# Load your test data
test_data_file = r'E:\Project\datasett\AWID3_Dataset_CSV\CSV\all merged\final_anova_data.csv'
test_data = pd.read_csv(test_data_file)

X_test = test_data.drop('Label', axis=1)
y_test = test_data['Label']

# Predictions from the CART model
y_pred_cart = cart_model.predict(X_test)

cm_cart = confusion_matrix(y_test, y_pred_cart)

# Plotting the confusion matrix
sns.heatmap(cm_cart, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix for CART Model")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Formatting the confusion matrix as a text report
cm_text_report = f"Confusion Matrix:\n{cm_cart[0][0]} True Negatives, {cm_cart[0][1]} False Positives\n{cm_cart[1][0]} False Negatives, {cm_cart[1][1]} True Positives"
print(cm_text_report)
```

Figure 14. Confusion Matrix

Web-application Development: CART model employed in user-friendly Web application, so that normal user can enter details and check whether the access point is normal or rogue access points. Application was built with flask framework, HTML, CSS and JavaScript.

E: > Project > datasett > AWID3_Dataset_CSV > CSV > all merged > Application > flask_app.py > ...

```python
from flask import Flask, request, render_template
import joblib

# Initialize the Flask application
app = Flask(__name__)

# Load the trained model
model = joblib.load(r'final_cart.pkl')

# Home page route
@app.route('/')
def home():
    return render_template('index_enhanced.html')

@app.route('/about')
def about():
    return render_template('about.html')


# Prediction route
@app.route('/predict', methods=['POST'])
def predict():
    # Retrieve values from form
    input_features = [float(x) for x in request.form.values()]

    # Reshape and predict
    prediction = model.predict([input_features])[0]

    # Determine the output message
    output = 'RougeAP Found' if prediction == 1 else 'Normal'

    return render_template('index_enhanced.html', prediction_text=f'Prediction: {output}')



if __name__ == '__main__':
    app.run(debug=True)

```

Figure 15. Application code

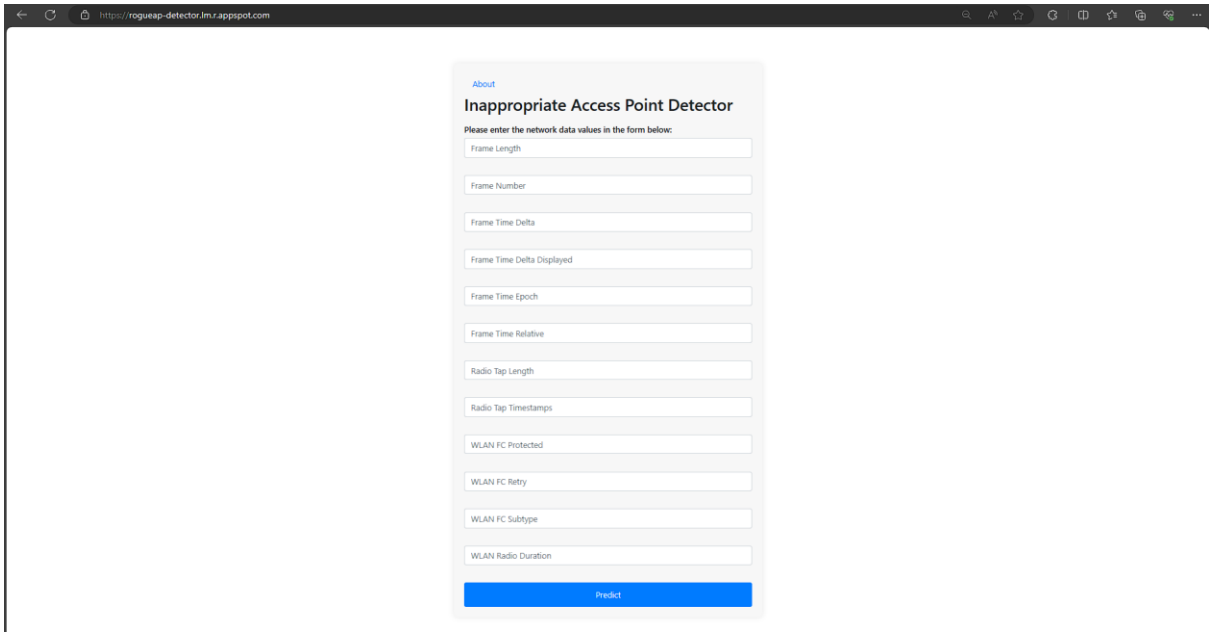


Figure 16. Web Application

References

AWID - 3 (2021) *Aegean.gr*. Available at: https://icsdweb.aegean.gr/awid/awid3 (Accessed: December 12, 2023).

Build from source on windows (no date) *TensorFlow*. Available at: https://www.tensorflow.org/install/source_windows

Installation — anaconda documentation (no date) *Anaconda.com*. Available at: https://docs.anaconda.com/free/anaconda/install/index.html