Configuration Manual

MSc Research Project
MSc in Cyber Security

Ashna Usman
Student ID: 23190264

School of Computing
National College of Ireland

Supervisor: Prof. Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ashna Usman

Student ID: X23190264

Programme: MSc in Cyber Security **Year:** 2024

Module: MSc Research Project

Lecturer: Prof. Niall Heffernan

Submission Due Date: 12/12/2024

Project Title: Enhancing Cybersecurity in IoT Healthcare Systems: A CNN-GRU Hybrid Approach for Intrusion Detection

Word Count: 832 **Page Count:** 7

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: Ashna Usman

Date: 12/12/2024

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

Ashna Usman
X23190264

1 Introduction

The configuration manual outline tools and technologies for research implementation are described. More specifically, Section 2 deals with the experimental procedure and Section 3 describes the technologies and software tools. Section 4 describes the execution plan with steps in which library imports are done, how data is pre-processed, how the model is trained and tested, and how the classification report is obtained. Information and data used to develop this software guide are provided in section 5 with references.

2 Experimental setup

The research was conducted on personal system, which has been done configuring as research setup to move ahead in implementation.

- Hardware Description: ser with AMD Ryzen 7 @5700U with Radeon Graphics @1.80 GHz, RAM: 16 GB.
- Windows Description: Windows 10, 22H2v.
- Research Structure: Use Windows 10, Anaconda3, version 1, Jupyter Notebook. 6, at version 4.12 in Python 3.9.13 and TensorFlow at the level of the 2.4 environment.

3 Technologies and Software used for Implementation Experimental setup

- VZ technique utilized Anaconda Py 3.1, JupyterNotebook v6.4.12 and Python implemented with tensorflow 2.4 environment.
- Anaconda is fundamentally a distribution of these languages like Python and R for scientific computing that is being used in employing the ML, data sciences for analysis of large data sets, prediction and so on. It practically is a liaison that only aims at enhancing the handling and deployment of packages.

Jupyter is web application that is built on open-source which supports open standards, having free software to use while the interactive computing for all the programming language has web services.

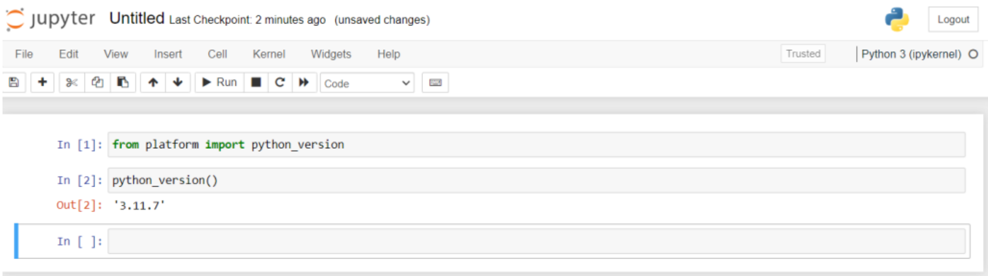

Fig 1 Python Model utilized in Jupyter Notebook.

4 Implementation

Step 1 : Anaconda was uploaded and placed.

Step 2 : Jupyter Notebook was installed and started.

Fig 2 : Jupyter Notebook Home page.

Step 3 : The dataset should be downloaded at the start of the process.

Step 4 : The libraries that is core for the Jupyter Notebook should be loaded if you plan on running through The libraries required for Machine Learning algorithms such as scikit-learn, Matplotlib and Seaborn.

Step 5 : Accessing the libraries for ML algorithms.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, GRU, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from sklearn.metrics import accuracy_score
```

Fig 3: Some primary libraries for ML algorithms are imported

Step 6: For pre-processing three dataset namely, Attack, Environment Monitoring, Patient Monitoring which cover the different domain of the IoT healthcare system should be loaded and dataset should be identified [1].

```
attack_df=pd.read_csv('ICUDatasetProcessed/Attack.csv')

environmental_df=pd.read_csv('ICUDatasetProcessed/environmentMonitoring.csv')

patient_df=pd.read_csv('ICUDatasetProcessed/patientMonitoring.csv')
```

Fig 4: Dataset Loading

Step 7: Pre-process the data by removing duplicate rows and combining datasets with the same features into one.

- Use Label Encoder to convert categorical columns into numbers
- Specify which columns need encoding (like ip.src, ip.dst, etc.).
- Apply the encoder to each column, replacing text values with numeric ones.
- Split the dataset into features (X) and labels (y), then divide them into training and testing sets.

```python
attack_df.shape
```

```python
attack_df.head()
```

```python
attack_df.duplicated().sum()          # sum of the duplicate value in the attack dataset
```

```python
environmental_df.shape
```

```python
environmental_df.duplicated().sum()    # sum of the duplicate value in the environmental monitoring dataset
```

```python
patient_df.shape
```

```python
patient_df.duplicated().sum()         # sum of the duplicate value in the patient monitoring dataset
```

```python
# Concatenate the datasets
combined_df=pd.concat([attack_df,environmental_df,patient_df],ignore_index=True)
```

```python
combined_df.shape
```

```python
combined_df.duplicated().sum()   # sum of the duplicate value in the concatenate dataset
```

```python
combined_df = combined_df.drop_duplicates() # remove the duplicate row
```

```python
combined_df.head()
```

```python
label_encoder = LabelEncoder()  # Label Encoding for categorical features
```

```python
encoded_features=[
    'ip.src', 'ip.dst', 'tcp.flags', 'tcp.payload', 'tcp.checksum',
    'mqtt.clientid', 'mqtt.conack.flags', 'mqtt.conflags',
    'mqtt.hdrflags', 'mqtt.msg', 'mqtt.topic', 'class'
]
```

```python
for column in encoded_features:
    combined_df[column] = combined_df[column].astype(str)
    combined_df[column] = label_encoder.fit_transform(combined_df[column])
```

```python
x=combined_df.drop(['label'],axis=1).values # Prepare the target and feature variables
y=combined_df['label'].values
```

```python
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=32)
# Split the dataset into train and test sets
```

Fig 5. Pre-processing of Dataset

Step 8: To envision the distribution of the target variable by creating a plot that shows the count of instances in each class, using a color palette to enhance clarity. By displaying this plot, you can assess whether there is any class imbalance, which may affect model performance.

```
plt.figure(figsize=(5,5))
sns.countplot(data = combined_df, x = 'label', palette = 'coolwarm')
plt.show()
```

Fig 6. Dataset Visualization

Step 9: To enhance the model performance, it includes dropping high correlated features and calculating the correlation matrix and visualizing it with a heatmap for better understanding of feature relationships. The plot is customized for clarity and displayed for analysis.

```
drop_corr_graph=combined_df.drop([
        'tcp.flags.urg','mqtt.conack.val',
       'mqtt.conflag.passwd', 'mqtt.conflag.qos', 'mqtt.conflag.reserved',
       'mqtt.conflag.retain', 'mqtt.conflag.willflag','mqtt.willmsg_len', 'ip.proto'
],axis=1)
```

```
correlation=drop_corr_graph.corr()
```

```
plt.figure(figsize=(15, 12))
sns.heatmap(correlation, annot=True, fmt=".2f", cmap='coolwarm',annot_kws={"size":8})
plt.title('Correlation Heatmap of Features', fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

Fig 7. Correlation Matrix to eliminate irrelevant features, reducing noise, and improve model performance

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='tcp.len', y='frame.len', data=combined_df)
plt.title('Scatter Plot of TCP Length vs Frame Length')
plt.xlabel('TCP Length')
plt.ylabel('Frame Length')
plt.show()
```

Fig 8. Relationship between tcp length and frame length

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='label', y='tcp.checksum', data=combined_df)
plt.title('Box Plot of TCP Checksum by Label')
plt.xlabel('Label')
plt.ylabel('TCP Checksum')
plt.show()
```

Fig 9. TCP checksum values across different label

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='class', y='frame.time_relative', data=combined_df)
plt.title('Box Plot of Frame Time Relative by Class')
plt.xlabel('Class')
plt.ylabel('Frame Time Relative')
plt.show()
```

Fig 10. Box plot to visualize the Frame time relative varies among different class labels

```
scaler=StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled = np.expand_dims(X_train_scaled, axis=-1)
X_test_scaled = np.expand_dims(X_test_scaled, axis=-1)
```

Fig 11. Standard Scaler

Step 10: To construct the Hybrid CNN-GRU model neural network, we sequentially structured convolutional layers, GRU layers, and the dense categories using binary classification.

1. CNN Layers: For feature extraction a 1D convolutional layer with 16 filters of 3 * 3 and ReLU activation is used, it is followed by max pooling layer of 2*2 and dropout layer for avoiding over fitting.

2. GRU Layer: For capturing sequential dependencies another layer in form of GRU is embedded with 32 units is introduced followed by another dropout layer.

3. Dense Layers: For the final prediction, a dense layer with 32 neurons and ReLU as activation is employed and a final dense layer that uses sigmoid to return binary classes is used.

```
model = Sequential()   # Define the model architecture
# CNN Layers
model.add(Conv1D(filters=16, kernel_size=3, activation='relu', input_shape=(X_train_scaled.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))  # Added dropout to avoid overfitting
# GRU layer
model.add(GRU(units=32, return_sequences=False))
model.add(Dropout(0.2))  # Added dropout to avoid regularization
# Dense layers
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid')) # Binary classification
```

Fig 12. Model Architecture

Step 11: Create the model with Adam optimizer, binary cross entropy as a measure of loss and accuracy as measure of performance and print the summary of the model.

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
# Compile the model
```

Fig 13. Model Compilations

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 49, 16)	64
max_pooling1d (MaxPooling1D)	(None, 24, 16)	0
dropout (Dropout)	(None, 24, 16)	0
gru (GRU)	(None, 32)	4,800
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 32)	1,056
dropout_2 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

```
Total params: 5,953 (23.25 KB)
Trainable params: 5,953 (23.25 KB)
Non-trainable params: 0 (0.00 B)
```

Fig 14. Model Sequential

Step 12: Update the model on the given training data for 10 cycles with the batch size of 32, and then validate the model with the test data.

```
history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_data=(X_test_scaled, y_test))
# Train the model
```

```
Epoch 1/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 96s 19ms/step - accuracy: 0.8914 - loss: 0.2882 - val_accuracy: 0.9977 - val_loss: 0.0188
Epoch 2/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 87s 18ms/step - accuracy: 0.9976 - loss: 0.0174 - val_accuracy: 0.9979 - val_loss: 0.0141
Epoch 3/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 87s 18ms/step - accuracy: 0.9979 - loss: 0.0138 - val_accuracy: 0.9983 - val_loss: 0.0077
Epoch 4/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 86s 18ms/step - accuracy: 0.9985 - loss: 0.0079 - val_accuracy: 0.9992 - val_loss: 0.0031
Epoch 5/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 88s 19ms/step - accuracy: 0.9989 - loss: 0.0049 - val_accuracy: 0.9992 - val_loss: 0.0018
Epoch 6/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 87s 18ms/step - accuracy: 0.9990 - loss: 0.0029 - val_accuracy: 0.9992 - val_loss: 0.0014
Epoch 7/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 87s 18ms/step - accuracy: 0.9992 - loss: 0.0024 - val_accuracy: 0.9998 - val_loss: 7.2383e-04
Epoch 8/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 86s 18ms/step - accuracy: 0.9993 - loss: 0.0019 - val_accuracy: 0.9999 - val_loss: 7.3977e-04
Epoch 9/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 87s 18ms/step - accuracy: 0.9994 - loss: 0.0017 - val_accuracy: 0.9999 - val_loss: 5.6738e-04
Epoch 10/10
4718/4718 ━━━━━━━━━━━━━━━━━━━━ 86s 18ms/step - accuracy: 0.9996 - loss: 0.0013 - val_accuracy: 0.9999 - val_loss: 5.4417e-04
```

Fig 15 Model Training

Step 13: Evaluate the model using accuracy, loss, prediction and, use confusion matrix and finally plot the confusion matrix.

```
# Plot accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.show()
# Plot loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```

```
y_pred = model.predict(X_test_scaled)   # Make predictions on the test data
y_pred_classes = np.round(y_pred).astype(int).flatten() # Using rounding to get binary
```

```
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Evaluation Loss: {loss:.4f}")
print(f"Evaluation Accuracy: {accuracy:.4f}")          # Evaluate the model
```

Fig 16. Actual and Predicted plot

```
cm = confusion_matrix(y_test, y_pred_classes)   # Assuming y_test and y_pred_classes are defined
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal', 'Attack'],      # Draw the heatmap with label 0 and 1
            yticklabels=['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix Heatmap')        # Add labels and title
plt.show()          #show the plot
```

Fig 17. Confusion matrix

Step 14: To assess the model's performance, it is necessary to use a classification report, which compares the actual test labels with the model's predictions and includes key performance metrics.

```
print(classification_report(y_test, y_pred_classes))        # Classification report
```

Fig 18. Classification Report

Step 15: To validate the model's predictions, randomly select 10 samples from the test set and compare their actual labels with the predicted ones. This step provides a straightforward evaluation of the model's performance on specific testing instances, helping to verify whether the predictions align with the expected outcomes

```python
# Select 10 random indices
random_indices = np.random.choice(len(y_test), 10, replace=False)

# Extract the corresponding test data and predictions
random_X_test = X_test_scaled[random_indices]
random_y_test = y_test[random_indices]
random_y_pred = y_pred_classes[random_indices]

# Print the actual vs predicted values for 10 random samples
print("Random 10 samples: Actual vs Predicted")
for i, idx in enumerate(random_indices):
    print(f"Sample {i + 1}: Actual = {random_y_test[i]}, Predicted = {random_y_pred[i]}")
```

Fig 19. Data Frame to compare actual and predicted values

5 Reference

[1] F. Malik, "IoT Healthcare Security Dataset ICU Healthcare Security Dataset." 2022. [Online]. Available: https://www.kaggle.com/datasets/faisalmalik/iot-healthcare-security-dataset