

# Configuration Manual

MSc Research Project  
Data Analytics

**Jothybala Murugan**

StudentID: x22245201

School of Computing

National College of Ireland

Supervisor: Dr. Abid Yaqoob

**National College of Ireland Project  
Submission Sheet School of Computing**



<b>Student Name:</b>	Jothybala Murugan
<b>Student ID:</b>	x22245201
<b>Programme:</b>	Data Analytics
<b>Year:</b>	2024
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Dr. Abid Yaqoob
<b>Submission Due Date:</b>	12/08/2024
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	719
<b>Page Count:</b>	12

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.

<b>Signature:</b>	
<b>Date:</b>	12 <sup>th</sup> August 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	

Date:	
Penalty Applied (if applicable):	

# Configuration Manual

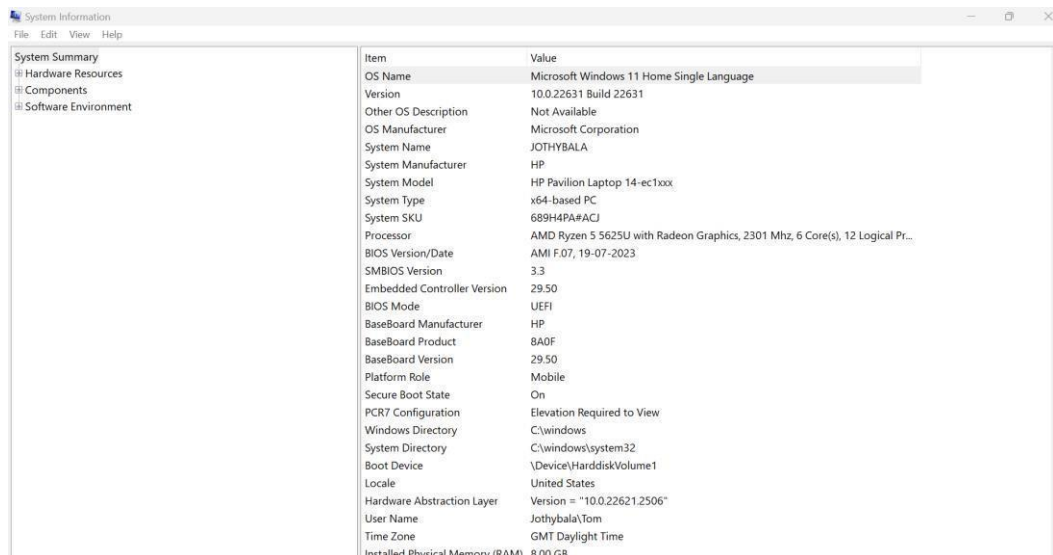
Jothybala Murugan  
x22245201

## 1 Introduction

Research on the intrusion detection system remains continuing with researchers and computer programmers in improving and enhance deep learning algorithms globally. A description of the steps needed to carry out the outcomes of this research is given in this configuration manual. This configuration manual includes the use of both software and hardware in research experiments. Knowing the Python programming is used here to achieve the target can be enhanced by this. We followed the instructions, each section will explain the function.

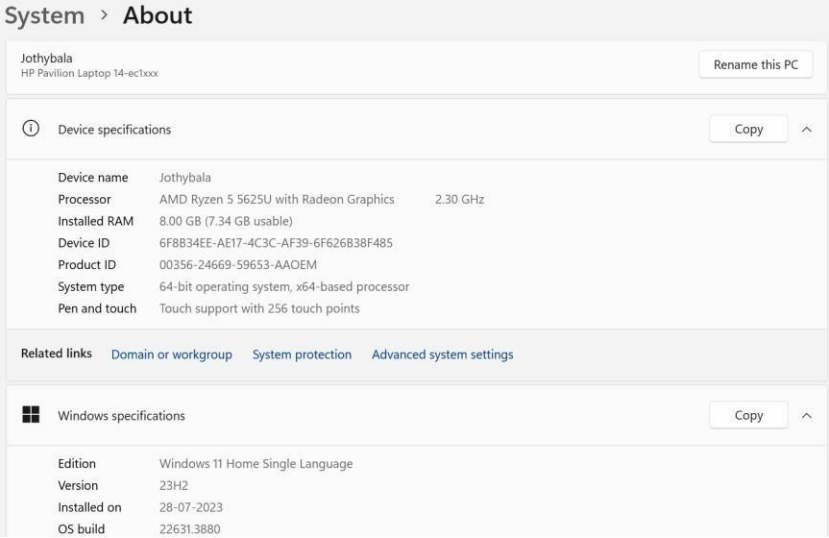
## 2 System Configuration

In this research, the hardware specification of the system is Windows 11 64-bit operating system, x 64-based processor system, 8 GB RAM, 256 GB of ROM, AMD Ryzen 5 - 5625U with Radeon Graphics -2.30 GHz is utilized. More hardware and operating system specification are shown in Figure1.



Item	Value
OS Name	Microsoft Windows 11 Home Single Language
Version	10.0.22631 Build 22631
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	JOTHYBALA
System Manufacturer	HP
System Model	HP Pavilion Laptop 14-ec1xxx
System Type	x64-based PC
System SKU	689H4PA#ACJ
Processor	AMD Ryzen 5 5625U with Radeon Graphics, 2301 Mhz, 6 Core(s), 12 Logical Pr...
BIOS Version/Date	AMI F.07, 19-07-2023
SMBIOS Version	3.3
Embedded Controller Version	29.50
BIOS Mode	UEFI
BaseBoard Manufacturer	HP
BaseBoard Product	8A0F
BaseBoard Version	29.50
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\windows
System Directory	C:\windows\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.22621.2506"
User Name	Jothybala\Tom
Time Zone	GMT Daylight Time
Installed Physical Memory (RAM)	8.00 GB

Figure 1: Hardware and Operating system specification



### Figure 2: Detailed specifications of the system

In figure 2, the detailed specifications of the system environment mentioned clearly.

### 3 Environment Setup

### 3.1 Anaconda Navigator

The deep learning models in this study experiment was implemented using the Python programming language. The programming code portion of the script is completed using Jupyter Notebook, which has been configured by Anaconda Navigator (available at this site), as Figure 3 illustrates.

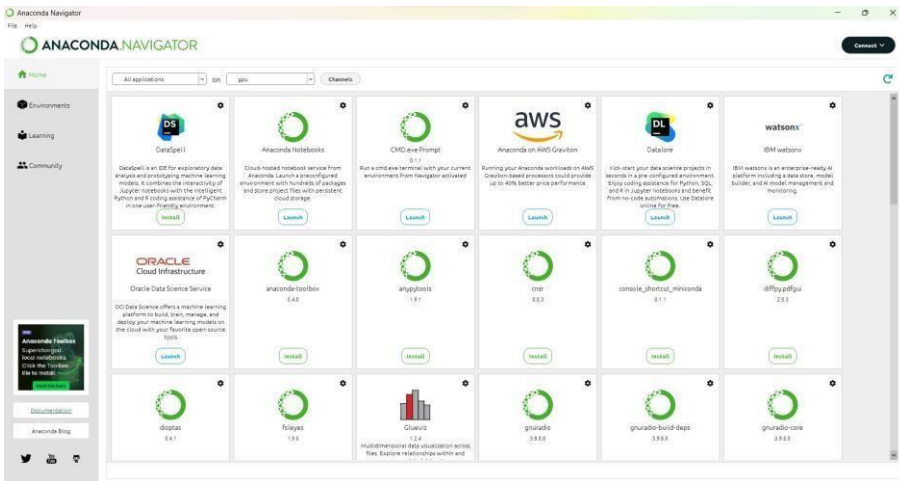
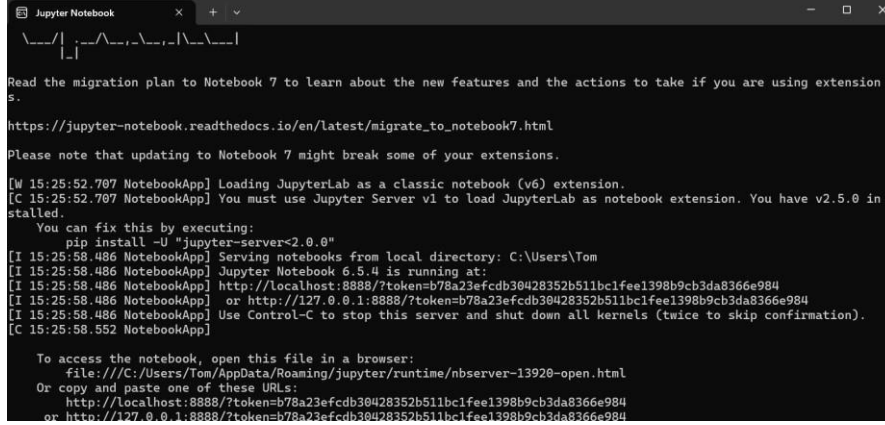


Figure 3: Anaconda Navigator

### 3.2 Environment

The anaconda base command prompt is used for developing a new environment in order to handle the libraries for Python programs. The python was Installed with Python 3.10

with numerous packages such as pandas, numpy, matplotlib, seaborn, Tensorflow, keras, time. For this research, the main tools used were anaconda navigator, Jupyter notebook, word, excel. The new environment jupyter notebook was activated using the anaconda navigator. In figure 4, the activation of jupyter notbook in the command is shown clearly which is used for faster execution.



```

Jupyter Notebook

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extension
s.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[W 15:25:52.787 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[C 15:25:52.787 NotebookApp] You must use Jupyter Server v1 to load JupyterLab as notebook extension. You have v2.5.0 in
stalled.

You can fix this by executing:
  pip install -U "jupyter-server<2.0.0"
[I 15:25:58.486 NotebookApp] Serving notebooks from local directory: C:\Users\Tom
[I 15:25:58.486 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 15:25:58.486 NotebookApp] http://localhost:8888/?token=b78a23efcdb30428352b511bc1fee1398b9cb3da8366e984
[I 15:25:58.486 NotebookApp] or http://127.0.0.1:8888/?token=b78a23efcdb30428352b511bc1fee1398b9cb3da8366e984
[I 15:25:58.486 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:25:58.552 NotebookApp]

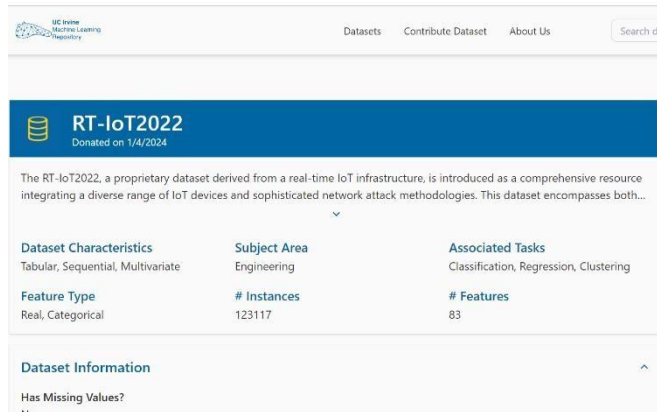
To access the notebook, open this file in a browser:
  file:///C:/Users/Tom/AppData/Roaming/jupyter/runtime/nbserver-13920-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=b78a23efcdb30428352b511bc1fee1398b9cb3da8366e984
  or http://127.0.0.1:8888/?token=b78a23efcdb30428352b511bc1fee1398b9cb3da8366e984

```

Figure 4: Jupyter notebook

## 4. Dataset Sources

In this research, the RT-IoT2022 dataset was used with number of network traffic data in the IoT devices. It contains 123117 instances with 85 features. This dataset has been obtained from the UCI repository. In figure 5, the dataset is mentioned.



UCI Machine Learning Repository		
Datasets    Contribute Dataset    About Us    Search datasets		
<b>RT-IoT2022</b> Donated on 1/4/2024		
The RT-IoT2022, a proprietary dataset derived from a real-time IoT infrastructure, is introduced as a comprehensive resource integrating a diverse range of IoT devices and sophisticated network attack methodologies. This dataset encompasses both...		
<b>Dataset Characteristics</b> Tabular, Sequential, Multivariate	<b>Subject Area</b> Engineering	<b>Associated Tasks</b> Classification, Regression, Clustering
<b>Feature Type</b> Real, Categorical	<b># Instances</b> 123117	<b># Features</b> 83
<b>Dataset Information</b> Has Missing Values? No		

Figure 5: Dataset from UCI repository

## 5 Research Experiment

The program code is divided into sections that help in a detailed and clear understanding of all the parts.

## 5.1 Importing Libraries

The first step of coding is to load the necessary python libraries and packages which is shown in Figure 6.

```
• Importing Libraries

In [1]: import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sns
import time
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import GRU, Bidirectional, Conv1D, MaxPooling1D, Dense, Dropout
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, LSTM, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Figure 6: Importing Libraries

## 5.2 Data Loading

The dataset used in this experiment is gathered from the UCI repository website <sup>1</sup>. The dataset is donated by the authors Sharmila, B. S., & Nagapadma, R. (2023). The dataset was downloaded in a zip file with the 3.3mb size and loaded in the jupyter file which is shown in Figure 7.

```
In [2]: data = pd.read_csv("C:\\Users\\Tom\\Downloads\\rt-iot2022 (1)\\RT_IOT2022.csv")
data

Out[2]:
```

Unnamed: 0	id.orig_p	id.resp_p	proto	service	flow_duration	fwd_pkts_tot	bwd_pkts_tot	fwd_data_pkts_tot	bwd_data_pkts_tot	...	active_std
0	0	38967	1883	tcp	mqtt	32.011598	9	5	3	...	0.0 2.9
1	1	51143	1883	tcp	mqtt	31.883584	9	5	3	...	0.0 2.9
2	2	44761	1883	tcp	mqtt	32.124053	9	5	3	...	0.0 2.9
3	3	60893	1883	tcp	mqtt	31.961063	9	5	3	...	0.0 2.9
4	4	51087	1883	tcp	mqtt	31.902362	9	5	3	...	0.0 2.9
...	...	...	...	...	...	...	...	...	...	...	...
123112	2005	59247	63331	tcp	-	0.000006	1	1	0	...	0.0 0.0
123113	2006	59247	64623	tcp	-	0.000007	1	1	0	...	0.0 0.0
123114	2007	59247	64680	tcp	-	0.000006	1	1	0	...	0.0 0.0
123115	2008	59247	65000	tcp	-	0.000006	1	1	0	...	0.0 0.0
123116	2009	59247	65129	tcp	-	0.000006	1	1	0	...	0.0 0.0

123117 rows x 85 columns

Figure 7: Loading data into pandas dataframe.

<sup>1</sup> <https://archive.ics.uci.edu/dataset/942/rt-iot2022>

```
In [86]: print("\nMissing values:")
print(data.isnull().sum())
```

```
Missing values:
Unnamed: 0          0
id.orig_p          0
id.resp_p          0
proto              0
service            0
..
idle.std           0
fwd_init_window_size 0
bwd_init_window_size 0
fwd_last_window_size 0
Attack_type        0
Length: 85, dtype: int64
```

Figure 8: Null values

The dataset had no missing values which was mentioned by the authors as well and in figure 8, we checked for the null values, and we can see there is no null values.

### 5.3 Exploratory data analysis

In the EDA part, we analyzed the distribution of the target variable, which is attack type, which is one of the categorical variables, outlier detection which was shown in Figure 9.

```
Cat_Var = data2[Categorical_variables]

melted_data = Cat_Var.melt('Attack_type', var_name='Category', value_name='Value')
sns.pairplot(melted_data, hue='Category', diag_kind='kde', height=3)

plt.suptitle('Pair Plot of Categorical Variables with Attack type', y=1.02)
plt.show()
```

Figure 9: Pair Plot of Categorical Variables with Attack type

In Figure 10, this code snippet is used for the visually summarizing the distribution of data, identifying potential outliers using Boxplot.

```
plt.figure(figsize=(15, 10))
for i, col in enumerate(new_columns_1, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(x=data2[col])
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```

Figure 10: Boxplot of selected features.



## 5.4 Data preprocessing

After the data is pre-processed by encoding the categorical variables, scaling the numerical variables. The next step was selecting the important features which was shown in figure 11.

```
correlations = data6[numerical_vars].corrwith(data6['Attack_type'].astype('category').cat.codes)
correlation_df = pd.DataFrame(correlations, columns=['Correlation'])
top_features = correlation_df.abs().nlargest(10, 'Correlation').index
%matplotlib inline

plt.figure(figsize=(25, 12))
sns.heatmap(data2[top_features].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Top 10 Features with Attack_type', fontsize=16)
plt.show()
```

Figure 11: Correlation of Top 10 Features with Attack\_type.

After the feature selection, using train\_test\_split function, we split dataset into training (80%) and testing (20%) sets and reshaped the test and train data before the use of the standardScaler function which is shown in figure 12.

```
# Split into training and testing sets without stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the feature matrix
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape input data to fit GRU input requirements
X_train_resaped = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_test_resaped = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1], 1))
```

Figure 12: Data splitting

## 5.5 Model Implementation

### 5.5.1 Gated Recurrent Unit

#### GRU model

```
def build_gru_model(input_shape, num_classes, activation, loss):
    model = Sequential()
    model.add(GRU(128, input_shape=input_shape, return_sequences=True))
    model.add(GRU(64))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation=activation))
    model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])
    return model

# Build and train GRU model
gru_model = build_gru_model((X_train_resaped.shape[1], 1), num_classes, activation, loss)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

start_time = time.time()
gru_history = gru_model.fit(X_train_resaped, y_train, epochs=5, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
gru_time = time.time() - start_time
```

D:\anaconda\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(**kwargs)
```

Epoch 1/5	2463/2463	35s	12ms/step	- accuracy: 0.9061	- loss: 0.3506	- val_accuracy: 0.9747	- val_loss: 0.0888
Epoch 2/5	2463/2463	30s	12ms/step	- accuracy: 0.9712	- loss: 0.0935	- val_accuracy: 0.9445	- val_loss: 0.0927
Epoch 3/5	2463/2463	29s	12ms/step	- accuracy: 0.9798	- loss: 0.0667	- val_accuracy: 0.9828	- val_loss: 0.0527
Epoch 4/5	2463/2463	30s	12ms/step	- accuracy: 0.9820	- loss: 0.0599	- val_accuracy: 0.9858	- val_loss: 0.0426
Epoch 5/5	2463/2463	30s	12ms/step	- accuracy: 0.9840	- loss: 0.0524	- val_accuracy: 0.9878	- val_loss: 0.0396

Figure 13: Model Implementation

```
# Evaluation of GRU model
gru_loss, gru_accuracy = gru_model.evaluate(X_test_resaped, y_test)
print(f'GRU Test Accuracy: {gru_accuracy:.4f}')
print(f'GRU Training Time: {gru_time:.2f} seconds')
```

770/770	4s	5ms/step	- accuracy: 0.9878	- loss: 0.0434
---------	----	----------	--------------------	----------------

GRU Test Accuracy: 0.9870  
GRU Training Time: 154.27 seconds

```
# Predictions
gru_y_pred = gru_model.predict(X_test_resaped)
gru_y_pred_classes = np.argmax(gru_y_pred, axis=1)
```

```
# Evaluation metrics
gru_precision = precision_score(y_test, gru_y_pred_classes, average='weighted')
gru_recall = recall_score(y_test, gru_y_pred_classes, average='weighted')
gru_f1 = f1_score(y_test, gru_y_pred_classes, average='weighted')
gru_conf_matrix = confusion_matrix(y_test, gru_y_pred_classes)

print(f'GRU Precision: {gru_precision:.4f}')
print(f'GRU Recall: {gru_recall:.4f}')
print(f'GRU F1-Score: {gru_f1:.4f}')
print(f'GRU Confusion Matrix:\n{gru_conf_matrix}')
```

GRU Precision: 0.9862  
 GRU Recall: 0.9862  
 GRU F1-Score: 0.9859  
 GRU Confusion Matrix:

[	1484	0	0	0	0	0	2	1	6	0	84	1]
[	2	71	0	0	0	0	0	0	26	0	1	0]
[	0	0	18897	0	0	0	0	0	0	0	0	0]
[	1	1	0	867	0	1	0	0	0	0	1	0]
[	6	0	0	0	0	0	0	0	0	0	0	0]
[	1	0	0	0	0	2	0	0	0	0	0	0]
[	0	0	0	0	0	0	391	0	2	0	0	0]
[	1	0	0	4	0	0	0	215	0	0	0	0]
[	13	0	0	0	0	0	20	0	453	0	3	0]
[	3	0	0	0	0	0	0	0	0	381	0	0]
[	120	0	0	0	0	0	0	0	11	0	1494	0]
[	6	1	1	0	0	2	1	1	5	0	12	29]]

Figure 14: Evaluation

## 5.5.2 Bidirectional GRU

### BIGRU model

```
def build_bi_gru_model(input_shape, num_classes, activation, loss):
    model = Sequential()
    model.add(Bidirectional(GRU(128, return_sequences=True), input_shape=input_shape))
    model.add(Bidirectional(GRU(64)))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation=activation))
    model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])
    return model

# Build and train BiGRU model
bi_gru_model = build_bi_gru_model((X_train_resaped.shape[1], 1), num_classes, activation, loss)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

start_time = time.time()
bi_gru_history = bi_gru_model.fit(X_train_resaped, y_train, epochs=5, batch_size=32, validation_split=0.2,
                                callbacks=[early_stopping])
bi_gru_time = time.time() - start_time
```

Epoch 1/5

D:\anaconda\Lib\site-packages\keras\src\layers\rnn\bidirectional.py:107: UserWarning: Do not pass an `input\_shape`/'input\_dim' argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(**kwargs)
```

2463/2463	28s	10ms/step	- accuracy: 0.9319	- loss: 0.2545	- val_accuracy: 0.9826	- val_loss: 0.0625
Epoch 2/5						
2463/2463	23s	9ms/step	- accuracy: 0.9797	- loss: 0.0677	- val_accuracy: 0.9853	- val_loss: 0.0474
Epoch 3/5						
2463/2463	23s	10ms/step	- accuracy: 0.9824	- loss: 0.0549	- val_accuracy: 0.9777	- val_loss: 0.0631
Epoch 4/5						
2463/2463	23s	9ms/step	- accuracy: 0.9845	- loss: 0.0497	- val_accuracy: 0.9870	- val_loss: 0.0361
Epoch 5/5						
2463/2463	23s	9ms/step	- accuracy: 0.9849	- loss: 0.0456	- val_accuracy: 0.9704	- val_loss: 0.0696

Figure 15: Model Implementation

```
# Evaluation of BiGRU model
bi_gru_loss, bi_gru_accuracy = bi_gru_model.evaluate(X_test_reshaped, y_test)
print(f'BiGRU Test Accuracy: {bi_gru_accuracy:.4f}')
print(f'BiGRU Training Time: {bi_gru_time:.2f} seconds')
```

770/770 ————— 2s 3ms/step - accuracy: 0.9868 - loss: 0.0399  
BiGRU Test Accuracy: 0.9861  
BiGRU Training Time: 121.44 seconds

```
# Predictions
bi_gru_y_pred = bi_gru_model.predict(X_test_reshaped)
bi_gru_y_pred_classes = np.argmax(bi_gru_y_pred, axis=1)
```

```
# Evaluation metrics
bi_gru_precision = precision_score(y_test, bi_gru_y_pred_classes, average='weighted')
bi_gru_recall = recall_score(y_test, bi_gru_y_pred_classes, average='weighted')
bi_gru_f1 = f1_score(y_test, bi_gru_y_pred_classes, average='weighted')
bi_gru_conf_matrix = confusion_matrix(y_test, bi_gru_y_pred_classes)
```

```
print(f'BiGRU Precision: {bi_gru_precision:.4f}')
print(f'BiGRU Recall: {bi_gru_recall:.4f}')
print(f'BiGRU F1-Score: {bi_gru_f1:.4f}')
print(f'BiGRU Confusion Matrix:\n{bi_gru_conf_matrix}')
```

BiGRU Precision: 0.9861  
BiGRU Recall: 0.9861  
BiGRU F1-Score: 0.9860  
BiGRU Confusion Matrix:

```
[[ 1482    6    2    1    0    0    1    0    1    0    85    0]
 [    0   73    0    0    0    0    0    0    26    0    1    0]
 [    0    0 18897    0    0    0    0    0    0    0    0    0]
 [    1    1    0   867    0    0    0    0    0    0    0    2]
 [    2    0    0    0    4    0    0    0    0    0    0    0]
 [    1    0    0    0    0    2    0    0    0    0    0    0]
 [    0    0    0    0    0    0   386    0    4    0    3    0]
 [    0    1    0    3    0    0    0   216    0    0    0    0]
 [   10    0    0    0    0    0    0    0   460    0   18    1]
 [    3    0    0    0    0    0    0    0    0   381    0    0]
 [   138    3    0    0    0    0    0    0    0    0  1483    1]
 [    5    0    1    2    0    0    0    0    5    0   14   31]]
```

Figure 16: Evaluation

### 5.5.3 Convolutional gated recurrent unit

## ConvGRU model

```
def build_conv_gru_model(input_shape, num_classes, activation, loss):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu', input_shape=input_shape))
    model.add(MaxPooling1D(pool_size=2))
    model.add(GRU(128, return_sequences=True))
    model.add(GRU(64))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation=activation))
    model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])
    return model

# Build and train ConvGRU model
conv_gru_model = build_conv_gru_model((X_train_resaped.shape[1], 1), num_classes, activation, loss)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

start_time = time.time()
conv_gru_history = conv_gru_model.fit(X_train_resaped, y_train, epochs=5, batch_size=32, validation_split=0.2,
                                     callbacks=[early_stopping])
conv_gru_time = time.time() - start_time
```

Epoch 1/5

D:\anaconda\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:107: UserWarning: Do not pass an `input\_shape` / `input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

2463/2463	14s	5ms/step	- accuracy: 0.9136	- loss: 0.3439	- val_accuracy: 0.9784	- val_loss: 0.0730
Epoch 2/5						
2463/2463	11s	4ms/step	- accuracy: 0.9754	- loss: 0.0780	- val_accuracy: 0.9832	- val_loss: 0.0502
Epoch 3/5						
2463/2463	11s	4ms/step	- accuracy: 0.9815	- loss: 0.0603	- val_accuracy: 0.9859	- val_loss: 0.0478
Epoch 4/5						
2463/2463	11s	4ms/step	- accuracy: 0.9831	- loss: 0.0532	- val_accuracy: 0.9854	- val_loss: 0.0426
Epoch 5/5						
2463/2463	11s	5ms/step	- accuracy: 0.9852	- loss: 0.0491	- val_accuracy: 0.9850	- val_loss: 0.0428

Figure 17: Model Implementation

```
# Evaluation of ConvGRU model
conv_gru_loss, conv_gru_accuracy = conv_gru_model.evaluate(X_test_resaped, y_test)
print(f'ConvGRU Test Accuracy: {conv_gru_accuracy:.4f}')
print(f'ConvGRU Training Time: {conv_gru_time:.2f} seconds')
```

```
# Predictions
conv_gru_y_pred = conv_gru_model.predict(X_test_resaped)
conv_gru_y_pred_classes = np.argmax(conv_gru_y_pred, axis=1)
```

```
# Evaluation metrics
conv_gru_precision = precision_score(y_test, conv_gru_y_pred_classes, average='weighted')
conv_gru_recall = recall_score(y_test, conv_gru_y_pred_classes, average='weighted')
conv_gru_f1 = f1_score(y_test, conv_gru_y_pred_classes, average='weighted')
conv_gru_conf_matrix = confusion_matrix(y_test, conv_gru_y_pred_classes)
```

```
print(f'ConvGRU Precision: {conv_gru_precision:.4f}')
print(f'ConvGRU Recall: {conv_gru_recall:.4f}')
print(f'ConvGRU F1-Score: {conv_gru_f1:.4f}')
print(f'ConvGRU Confusion Matrix:\n{conv_gru_conf_matrix}')
```



```

ConvGRU Precision: 0.9861
ConvGRU Recall: 0.9859
ConvGRU F1-Score: 0.9858
ConvGRU Confusion Matrix:
[[ 1442    1    0    0    1    0    1    1   19    0   111    2]
 [    2   97    0    0    0    0    0    0    0    0    1    0]
 [    0    0 18897    0    0    0    0    0    0    0    0    0]
 [    1    1    0   868    0    0    0    0    0    0    1    0]
 [    2    0    0    0    4    0    0    0    0    0    0    0]
 [    1    0    0    0    0    2    0    0    0    0    0    0]
 [    0    0    0    0    0    0   387    0    6    0    0    0]
 [    1    0    0    3    0    0    0   216    0    0    0    0]
 [   10   24    0    0    0    0    0    0   452    0    3    0]
 [    3    0    0    0    0    0    0    0    0   381    0    0]
 [  115    1    0    0    0    0    0    0   13    0  1496    0]
 [    3    1    1    0    0    0    0    1    6    0    12   34]]

```

Figure 18: Evaluation

## 6 Final Evaluation

```

from sklearn.metrics import precision_score, recall_score, f1_score

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    accuracy = np.mean(y_pred_classes == y_test)
    precision = precision_score(y_test, y_pred_classes, average='weighted')
    recall = recall_score(y_test, y_pred_classes, average='weighted')
    f1 = f1_score(y_test, y_pred_classes, average='weighted')

    return accuracy, precision, recall, f1

# GRU model
gru_accuracy, gru_precision, gru_recall, gru_f1 = evaluate_model(gru_model, X_test_reshaped, y_test)

# BiGRU model
bi_gru_accuracy, bi_gru_precision, bi_gru_recall, bi_gru_f1 = evaluate_model(bi_gru_model, X_test_reshaped, y_test)

# ConvGRU model
conv_gru_accuracy, conv_gru_precision, conv_gru_recall, conv_gru_f1 = evaluate_model(conv_gru_model, X_test_reshaped, y_test)

```

```

models = ['GRU', 'BiGRU', 'ConvGRU']
accuracies = [gru_accuracy, bi_gru_accuracy, conv_gru_accuracy]
precisions = [gru_precision, bi_gru_precision, conv_gru_precision]
recalls = [gru_recall, bi_gru_recall, conv_gru_recall]
f1_scores = [gru_f1, bi_gru_f1, conv_gru_f1]

x = np.arange(len(models))
width = 0.2

fig, ax = plt.subplots(figsize=(10, 6))

rects1 = ax.bar(x - width*1.5, accuracies, width, label='Accuracy')
rects2 = ax.bar(x - width/2, precisions, width, label='Precision')
rects3 = ax.bar(x + width/2, recalls, width, label='Recall')
rects4 = ax.bar(x + width*1.5, f1_scores, width, label='F1-Score')

ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_title('Comparison of Evaluation Metrics for Different Models')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

```

Out[47]: <matplotlib.legend.Legend at 0x200da239510>

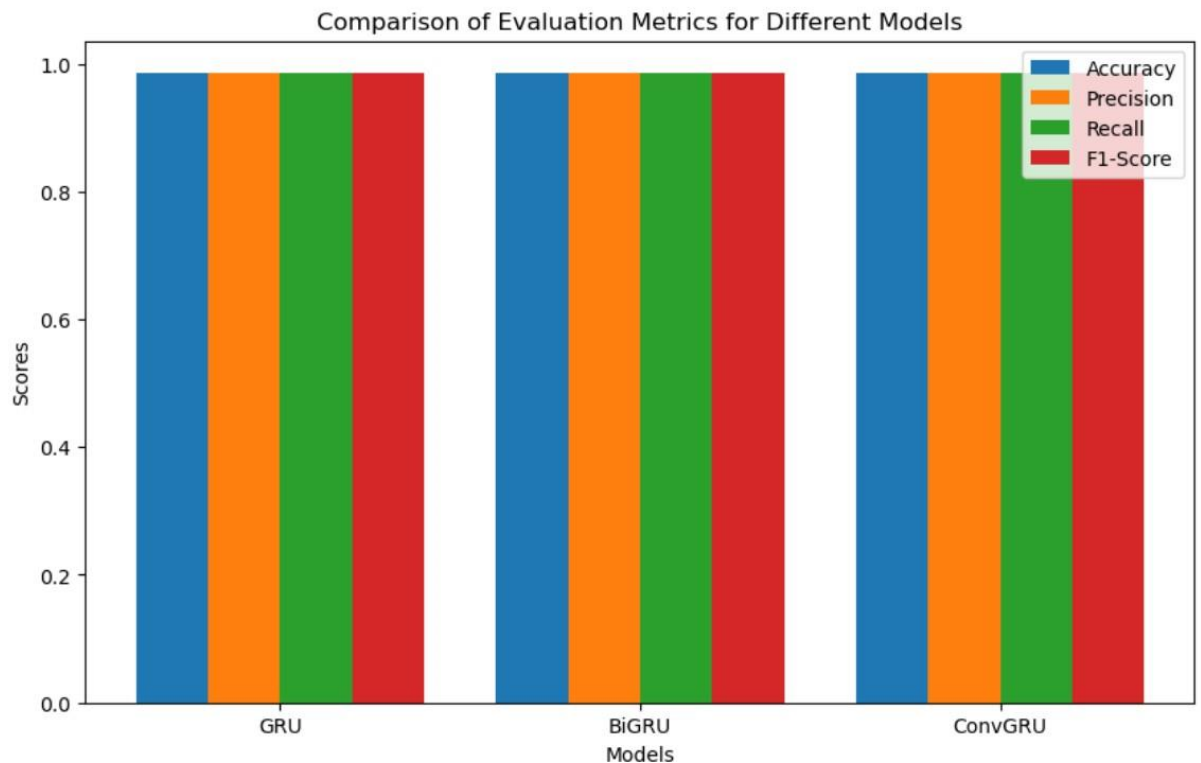


Figure 23: Final Evaluation.

## References

Sharmila, B. S., & Nagapadma, R. (2023). RT-IoT2022. UCI Machine Learning.

<https://doi.org/10.24432/C5P338>.

Available

at:

<https://archive.ics.uci.edu/dataset/942/rt-iot2022>.