# Configuration Manual

MSc Research Project
Programme Name

Ann Mohan
Student ID: x23175320

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Ann Mohan |
| **Student ID:** | X23175320 |
| **Programme:** | Msc in cybersecurity **Year:** 2024 |
| **Module:** | Research Project/ Internship |
| **Lecturer:** | Jawad Salahuddin |
| **Submission Due Date:** | 12/12/2024 |
| **Project Title:** | A Deep Learning Approach to Malicious Software Detection: Combining MLP and GRU |
| **Word Count:** | 1465 **Page Count:** 14 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Ann Mohan

**Date:** 11/11/2024

**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

Ann Mohan
X23175320

# 1. Introduction

This configuration outlines the experimental setup, tools, and processes used to implement the MLP-GRU hybrid model for malicious software detection. The MLP handles static feature analysis, while the GRU focuses on sequential patterns, improving detection accuracy and robustness. There are a total of five sections in which section two provides the experimental setup and section three and four provides the Implementation and Data preprocessing steps. Finally, it concludes with MLP-GRU architecture in section five.

# 2. Experimental Setup

a. **Hardware Requirements:**

- **Processor**: Minimum Quad-core (AMD Ryzen 7 or Intel i7)
- **RAM**: At least 16 GB
- **Graphics**: Optional for model acceleration using GPU (e.g., NVIDIA RTX 3060 or above)

b. **Software Requirements:**

- **Operating System**: Windows 10, 22H2 or equivalent
- **Python Version**: Python 3.9.13
- **IDE**: Jupyter Notebook (v6.4.12)
- **Frameworks**: TensorFlow 2.4 or newer, Scikit-learn, Pandas, Numpy, Matplotlib, Seaborn

The service provided by Jupyter is web-based because it is developed as an open source on the foundation of web open standards, and it is free (www.jupyter.org, n.d.). This feature allows it to support the most popular programming languages and facilitate diverse and renovative development.

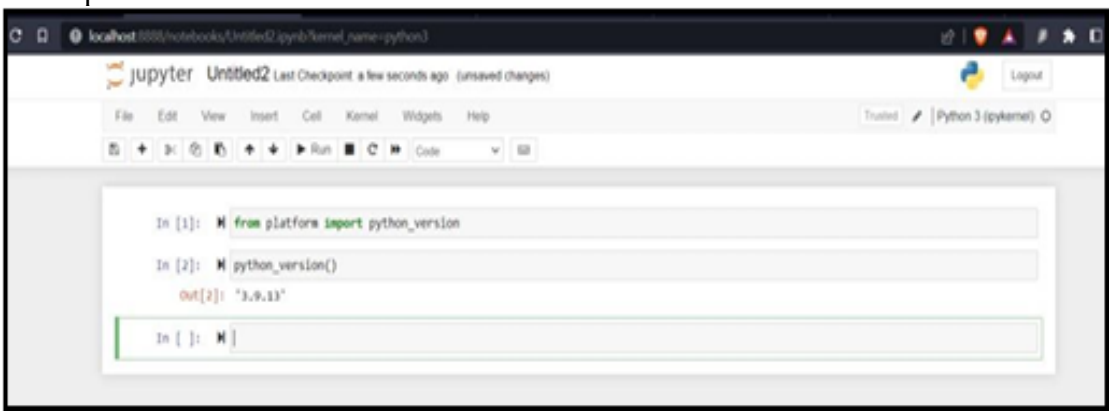


Figure 1: Python Version used in Jupyter Notebook.

# 3. Implementation

**Step 1 :** Anaconda was successfully downloaded and configured in this step.

**Step 2:** FN Control Center was configured and started Serving, Jupyter Notebook was set up and launched.

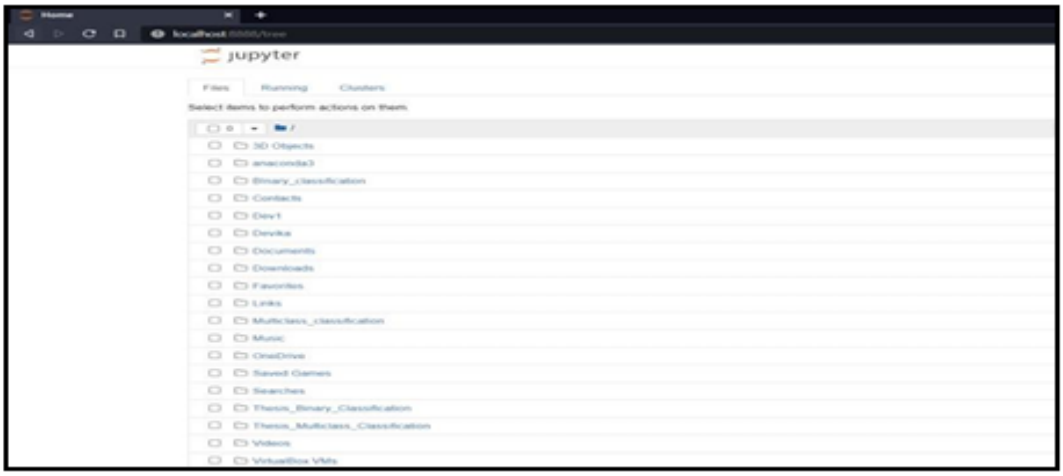


Figure 2 : The home page of Jupyter Notebook.

**Step 3** : The dataset is to be downloaded online.
https://www.kaggle.com/datasets/bansalaarushi/malware-dataset
**Step 4** : The essentials libraries related to Jupyter Notebook for any file execution should require the following the list; pandas, Tenserflow , numpy, time, scikit-learn, Matplotlib Seaborn and so on.

**Step 5** : This part of the code importing the libraries for Machine Learning algorithms.

```
[3]: import warnings
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import shap
     from sklearn.preprocessing import LabelEncoder
     from scipy import stats
     from sklearn.preprocessing import StandardScaler
     import warnings
     import matplotlib.pyplot as plt
     from imblearn.over_sampling import SMOTE
     import joblib
     from sklearn.utils import resample
     from collections import Counter
     from keras.models import Model
     from keras.layers import Input, Dense, Dropout, GRU, Concatenate
     from keras.utils import to_categorical
     from sklearn.model_selection import train_test_split
     warnings.filterwarnings("ignore")
```

Figure 3: Importing Basic Libraries

# 4. Dataset Preparation

## Step 6: Dataset Acquisition

Acquire a properly labelled dataset with two kinds of attributes – binary file pattern and sequential system call. Filter the data and select both healthy and unhealthy digital images for the deep learning algorithm.

```
[4]: df = pd.read_csv("/Users/annmo/OneDrive/Desktop/Thesis/subset_dataset.csv")
```

df

| | Protocol | Flow Duration | Total Fwd Packets | Total Backward Packets | Flow IAT Mean | Bwd IAT Mean | Fwd Header Length | Bwd Header Length | Fwd Packets/s | Bwd Packets/s | ... | SYN Flag Count | RST Flag Count | Average Packet Size | Avg Fwd Segment Size | Avg Bwd Segment Size | Subflow Fwd Packets | Init_Win_bytes_forward | Init_Win_bytes_backward | Label | category |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 6.0 | 984922.0 | 3.0 | 2.0 | 2.462305e+05 | 40.0 | 96.0 | 64.0 | 3.045926 | 2.030618 | ... | 1.0 | 0.0 | 114.400000 | 94.666667 | 2.0 | 3.0 | 1431.0 | 35.0 | ADWARE_DOWGIN | AdwareMalware |
| 1 | 6.0 | 985932.0 | 3.0 | 1.0 | 3.286440e+05 | 0.0 | 96.0 | 32.0 | 3.042806 | 1.014269 | ... | 1.0 | 0.0 | 3.750000 | 1.666667 | 5.0 | 3.0 | 1386.0 | 9.0 | ADWARE_DOWGIN | AdwareMalware |
| 2 | 6.0 | 22538.0 | 1.0 | 2.0 | 1.126900e+04 | 253.0 | 32.0 | 64.0 | 44.369509 | 88.739019 | ... | 0.0 | 0.0 | 10.333333 | 0.000000 | 15.5 | 1.0 | 2093.0 | 972.0 | ADWARE_DOWGIN | AdwareMalware |
| 3 | 6.0 | 31977.0 | 1.0 | 2.0 | 1.598850e+04 | 21.0 | 32.0 | 64.0 | 31.272477 | 62.544954 | ... | 0.0 | 0.0 | 10.333333 | 0.000000 | 15.5 | 1.0 | 1641.0 | 972.0 | ADWARE_DOWGIN | AdwareMalware |
| 4 | 6.0 | 49401.0 | 1.0 | 1.0 | 4.940100e+04 | 0.0 | 32.0 | 32.0 | 20.242505 | 20.242505 | ... | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 1.0 | 1550.0 | 510.0 | ADWARE_DOWGIN | AdwareMalware |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2576134 | 17.0 | 31241.0 | 1.0 | 1.0 | 3.124100e+04 | 0.0 | 20.0 | 32.0 | 32.009219 | 32.009219 | ... | 0.0 | 0.0 | 112.500000 | 42.000000 | 141.0 | 1.0 | -1.0 | -1.0 | SMSMALWARE_ZSONE | SmsMalware |
| 2576135 | 17.0 | 198548.0 | 6.0 | 6.0 | 1.804982e+04 | 29245.8 | 188.0 | 224.0 | 30.219393 | 30.219393 | ... | 0.0 | 0.0 | 804.916667 | 531.333333 | 853.5 | 6.0 | -1.0 | -1.0 | SMSMALWARE_ZSONE | SmsMalware |
| 2576136 | 17.0 | 1394.0 | 1.0 | 1.0 | 1.394000e+03 | 0.0 | 32.0 | 32.0 | 717.360115 | 717.360115 | ... | 0.0 | 0.0 | 171.000000 | 48.000000 | 246.0 | 1.0 | -1.0 | -1.0 | SMSMALWARE_ZSONE | SmsMalware |
| 2576137 | 17.0 | 80848.0 | 1.0 | 1.0 | 8.084800e+04 | 0.0 | 32.0 | 32.0 | 12.368890 | 12.368890 | ... | 0.0 | 0.0 | 87.000000 | 39.000000 | 96.0 | 1.0 | -1.0 | -1.0 | SMSMALWARE_ZSONE | SmsMalware |
| 2576138 | 6.0 | 10745192.0 | 1.0 | 1.0 | 1.074519e+07 | 0.0 | 20.0 | 20.0 | 0.093065 | 0.093065 | ... | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 1.0 | 1597.0 | 9300.0 | SMSMALWARE_ZSONE | SmsMalware |

2576139 rows × 22 columns

Figure 4: Data Preparation

## Step 7: Preprocessing Steps

1. **Data Cleaning**:

- Eliminate any row with multiple or no instance.
- Normalize an object label feature using the Label Encoding (for instance, ip.src, ip.dst).

Eliminating a row with multiple or no instance

```
[9]: df.columns = df.columns.str.strip()

[10]: df['Label'].nunique()

[10]: 43

[11]: df['category'].nunique()

[11]: 5

[12]: df["category"].value_counts()

[12]: category
      Benign                1205515
      AdwareMalware          424147
      ScarewareMalware       375427
      RansomewareMalware     348943
      SmsMalware             222107
      Name: count, dtype: int64
```

Figure 5: Data frame Column Cleaning and Analysis

```
[16]: df = df.dropna()

[17]: df = df.drop_duplicates()

[18]: label_encoder = LabelEncoder()

      df['Label_encoded'] = label_encoder.fit_transform(df['Label'])

      df['category_encoded'] = label_encoder.fit_transform(df['category'])

      df = df.drop(columns=['Label', 'category'])
```

Figure 6: Handling Missing Values and Duplicates

```
[22]: class_distribution = df['category_encoded'].value_counts()

      # Print the class counts
      print("Class Distribution:")
      print(class_distribution)

      # Plot the class distribution
      plt.figure(figsize=(8, 6))
      class_distribution.plot(kind='bar', color='skyblue')
      plt.title("Class Distribution")
      plt.xlabel("Class")
      plt.ylabel("Frequency")
      plt.xticks(rotation=0)
      plt.show()

Class Distribution:
category_encoded
1    919708
0    327673
3    291837
2    272452
4    171035
Name: count, dtype: int64
```

Figure 7: Category Distribution Visualization

**Step 8**: This gives the number of times each label appears in the category encoded column of a Data Frame. It then visualises this distribution using a bar chart which often gives a good glimpse of the scaled nature of the class in the dataset.

**Step 9: Feature Engineering**:

- Correlation analysis to remove several features that are closely associated with other features and thus very redundant (Calybre.global, 2024).
- It has been recommended to use maps such as the heat map to determine the connection.

Feature Scaling

Balance the dataset

[23]:
```
import pandas as pd

# Assuming your dataset is named 'df'
# Replace 'category_encoded' with your actual column name
target_count = 272452

# Downsample categories 1, 0, and 3
df_1 = df[df['category_encoded'] == 1].sample(target_count, random_state=42)
df_0 = df[df['category_encoded'] == 0].sample(target_count, random_state=42)
df_3 = df[df['category_encoded'] == 3].sample(target_count, random_state=42)

# Keep categories 2 and 4 as is
df_2 = df[df['category_encoded'] == 2]
df_4 = df[df['category_encoded'] == 4]

# Combine all the balanced subsets
df_balanced = pd.concat([df_1, df_0, df_3, df_2, df_4])

# Shuffle the final dataset
df_balanced = df_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

print(df_balanced['category_encoded'].value_counts())
```

Figure 8: Feature Scaling

[25]:
```
# Separate features (X) and target (y)
X = df_balanced.drop(columns=['category_encoded'])
y = df_balanced['category_encoded']

# Apply SMOTE only on category 4
# Convert other categories to a single majority class
smote = SMOTE(sampling_strategy={4: 272452}, random_state=42)

# Apply SMOTE
X_resampled, y_resampled = smote.fit_resample(X, y)

# Combine the resampled data into a new DataFrame
df_resampled = pd.concat([pd.DataFrame(X_resampled), pd.DataFrame(y_resampled, columns=['category_encoded'])], axis=1)

# Check the new class distribution
print(df_resampled['category_encoded'].value_counts())
```

```
category_encoded
2    272452
4    272452
0    272452
3    272452
1    272452
Name: count, dtype: int64
```

## Standard Scalar

[26]:
```
scaler = StandardScaler()

# Scale the resampled feature data
X_scaled = scaler.fit_transform(X_resampled)

# Combine the scaled features with the target variable
df_scaled = pd.concat([pd.DataFrame(X_scaled, columns=X.columns), pd.DataFrame(y_resampled, columns=['category_encoded'])], axis=1)

# Check the scaled DataFrame
print(df_scaled.head())

# Check the class distribution to ensure it's still balanced
print(df_scaled['category_encoded'].value_counts())
```

Figure 9: Data Balancing and Feature Scaling Process

**Step 10:** They work to minimize the class imbalance problem by aggregating minority classes into one and using SMOTE to oversample the new class (Brownlee, 2020). Then, it scales the features to enhance the model performance in the Figure. 10 below. The last thing in this process is to ensure that class distribution is not skewed after the above two transformations.

**Step 11: Normalization**:

- Standard scaling, this makes sure that all the features are on the same scale as those of the other.

```
   Protocol  Flow Duration  Total Fwd Packets  Total Backward Packets  \
0 -0.597610      -0.489173          -0.181584                0.183778
1 -0.597610       3.057139          -0.181584               -0.435863
2  1.670007      -0.488592          -0.498434               -0.311935
3 -0.597610      -0.491863           0.135265                0.059850
4 -0.597610      -0.504170          -0.498434               -0.311935

   Flow IAT Mean  Bwd IAT Mean  Fwd Header Length  Bwd Header Length  \
0      -0.445546     -0.262693          -0.002001          -0.004622
1       5.516584     -0.277625          -0.002008          -0.004813
2      -0.400078     -0.277625          -0.002058          -0.004776
3      -0.447579     -0.261710          -0.001989          -0.004708
4      -0.452301     -0.277625          -0.002058          -0.004776

   Fwd Packets/s  Bwd Packets/s  ...  SYN Flag Count  RST Flag Count  \
0      -0.128453      -0.088486  ...             0.0             0.0
1      -0.131739      -0.133109  ...             0.0             0.0
2      -0.130694      -0.124514  ...             0.0             0.0
3      -0.125066      -0.089702  ...             0.0             0.0
4      -0.019664       0.776443  ...             0.0             0.0

   Average Packet Size  Avg Fwd Segment Size  Avg Bwd Segment Size  \
0             1.637035              2.583281              1.078981
1            -0.700515             -0.624910             -0.600082
2             0.202263             -0.036173              0.134703
3             0.821311              0.406660              1.081521
4            -0.700515             -0.624910             -0.600082

   Subflow Fwd Packets  Init_Win_bytes_forward  Init_Win_bytes_backward  \
0            -0.181584                1.392336                -0.232983
1            -0.181584               -0.737995                -0.311963
2            -0.498434               -0.747333                -0.311963
3             0.135265                1.392336                -0.301388
4            -0.498434               -0.746353                 0.140767

   Label_encoded  category_encoded
0      -0.155236                 2
1       1.640480                 4
2      -0.563354                 2
3      -0.889847                 0
4       0.089634                 2

[5 rows x 22 columns]
category_encoded
2    272452
4    272452
0    272452
3    272452
1    272452
Name: count, dtype: int64
```

Figure 10: Normalized Data Table

# 5. MLP-GRU Model Architecture

**Step 12**: Component of MLP for Static Analysis

- Input Layer: They later restore neuron count with respect to the count of features.
- Hidden Layers: Euclidean distance with output from the fully connected to the next layer activation ReLU.
- Output Layer: It also embeds intermediate feature representation.

**Step 13:** These sequence analyses include the use of the GRU Component for Sequential Analysis.

- Input Layer: Sequential time-series data.
- GRU Layer: To model temporal features on the data, we use Gated Recurrent Units with 64.
- Dropout Layer: Prevent overfitting.

**Step 14:** Concatenation Layer In this study, MLP and GRU output vectors are combined to extract a final output vector for the prediction.

**Step 15:** Output Layer Dense Layer: softmax if there is more than one output class or sigmoid if there are only two output classes.

[28]:
```
# Step 5: Split the resampled data into training and testing sets
X_train_resampled, X_test, y_train_resampled, y_test = train_test_split(X_scaled, y_resampled, test_size=0.2, random_state=42)

# Step 6: Reshape data for GRU (time-series format)
X_train_gru = np.reshape(X_train_resampled, (X_train_resampled.shape[0], 1, X_train_resampled.shape[1]))
X_test_gru = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Step 7: One-hot encode the target labels for multi-class classification
num_classes = len(np.unique(y_resampled))  # Automatically determine the number of classes
y_train_categorical = to_categorical(y_train_resampled, num_classes=num_classes)
y_test_categorical = to_categorical(y_test, num_classes=num_classes)

# Step 8: Define the MLP model
mlp_input = Input(shape=(X_train_resampled.shape[1],))  # MLP uses 2D input (samples, features)
mlp_layer1 = Dense(64, activation='relu')(mlp_input)  # Increased complexity for MLP
mlp_dropout1 = Dropout(0.5)(mlp_layer1)
mlp_output = Dense(32, activation='relu')(mlp_dropout1)

# Step 9: Define the GRU model
gru_input = Input(shape=(X_train_gru.shape[1], X_train_gru.shape[2]))  # GRU uses 3D input (samples, timesteps, features)
gru_layer1 = GRU(64, return_sequences=False, activation='relu')(gru_input)  # Increased complexity for GRU
gru_dropout1 = Dropout(0.5)(gru_layer1)

# Step 10: Concatenate the outputs of MLP and GRU
merged = Concatenate()([mlp_output, gru_dropout1])

# Step 11: Output layer for multi-class classification
output = Dense(num_classes, activation='softmax')(merged)

# Step 12: Create the hybrid MLP + GRU model
model = Model(inputs=[mlp_input, gru_input], outputs=output)

# Step 13: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Step 14: Train the model
history = model.fit(
    [X_train_resampled, X_train_gru],  # Inputs: MLP and GRU
    y_train_categorical,  # Target labels
    epochs=10,  # Adjust epochs as needed
    batch_size=32,  # Adjust batch size as needed
    validation_data=([X_test, X_test_gru], y_test_categorical),  # Validation data
    verbose=1
)

# Step 15: Evaluate the model
loss, accuracy = model.evaluate([X_test, X_test_gru], y_test_categorical, verbose=0)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

Figure 11: Model Architecture

```
Epoch 1/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 157s 4ms/step - accuracy: 0.9235 - loss: 0.2447 - val_accuracy: 0.9894 - val_loss: 0.0457
Epoch 2/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 187s 4ms/step - accuracy: 0.9849 - loss: 0.0641 - val_accuracy: 0.9919 - val_loss: 0.0395
Epoch 3/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 141s 4ms/step - accuracy: 0.9880 - loss: 0.0529 - val_accuracy: 0.9934 - val_loss: 0.0341
Epoch 4/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 138s 4ms/step - accuracy: 0.9896 - loss: 0.0478 - val_accuracy: 0.9931 - val_loss: 0.0355
Epoch 5/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 138s 4ms/step - accuracy: 0.9901 - loss: 0.0451 - val_accuracy: 0.9937 - val_loss: 0.0321
Epoch 6/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 142s 4ms/step - accuracy: 0.9909 - loss: 0.0458 - val_accuracy: 0.9938 - val_loss: 0.0325
Epoch 7/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 134s 4ms/step - accuracy: 0.9913 - loss: 0.0464 - val_accuracy: 0.9936 - val_loss: 0.0335
Epoch 8/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 158s 4ms/step - accuracy: 0.9918 - loss: 0.0428 - val_accuracy: 0.9940 - val_loss: 0.0320
Epoch 9/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 195s 4ms/step - accuracy: 0.9920 - loss: 0.0443 - val_accuracy: 0.9943 - val_loss: 0.0310
Epoch 10/10
34057/34057 ━━━━━━━━━━━━━━━━━━━━ 130s 4ms/step - accuracy: 0.9922 - loss: 0.0393 - val_accuracy: 0.9940 - val_loss: 0.0321
Test Loss: 0.0321270190179348, Test Accuracy: 0.9940136075019836
```

Figure 12: Model Training

**Step 16:** The training data set was performed for 10 iterations. The training process was implemented using a validation data set and the final test of the trained model was conducted using the test data set.

## Model accuracy And Model loss

```
[29]: # Step 14: Evaluate the model on the test data
loss, accuracy = model.evaluate([X_test, X_test_gru], y_test_categorical, verbose=0)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")

# Step 15: Plot the training history
# Plot training & validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Test Loss: 0.032726459205150604
Test Accuracy: 0.9940540194511414
```

Figure 13: Model Accuracy and Model Loss

**Step 17:** This kind of model was trained and tested on a dataset. Model's performance was evaluated by training and validation datasets. The accuracy and loss functions were trained and validated for 10 epoch. There is the final test loss and accuracy that was produced as 0.0327 and 0.9940, respectively.

## Confusion Matrix

[30]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Make predictions
y_pred = model.predict([X_test, X_test_gru])
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test_categorical, axis=1)

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Display the confusion matrix as a diagram without colorbar
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_true))
fig, ax = plt.subplots()
disp.plot(cmap='Blues', values_format='d', ax=ax, colorbar=False)
plt.title("Confusion Matrix")
plt.show()
```

Figure 14: Confusion Matrix

**Step 18:** The confusion matrix summary of true positive 'TP', true negative 'TN', false positive 'FP' and false negative 'FN' made in each of the one class (GeeksForGeeks, 2018). The confusion matrix shows the number of true positive, true negative, false positive, and false negative predictions for each 1 class.

## Classification Report

[31]:
```
from sklearn.metrics import classification_report

print("Classification Report:")
print(classification_report(y_true, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     54312
           1       0.98      1.00      0.99     54415
           2       1.00      1.00      1.00     54615
           3       0.99      0.99      0.99     54605
           4       1.00      0.99      0.99     54505

    accuracy                           0.99    272452
   macro avg       0.99      0.99      0.99    272452
weighted avg       0.99      0.99      0.99    272452
```

Figure 15: Classification Report

**Step 19:** It provides a means to assess the quality of a multi-class classification model through a classification report. This can print out in precision, recall, F1-score and support; for each class; accuracy, Macro average, Weighted average 1.

## ROC-AUC Curve (for Multi-Class)

[33]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Binarize the labels for ROC calculation
y_test_bin = label_binarize(y_true, classes=np.unique(y_true))
y_pred_prob = model.predict([X_test, X_test_gru])

# Compute ROC curve and AUC for each class
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"Class {i} (AUC = {roc_auc:.2f})")

plt.title("Multi-Class ROC-AUC")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
8515/8515 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step
```

Figure 16: ROC-AUC Curve

**Step 20**: In a multi-class classification problem, the code also provides an ROC curve for each class while not necessarily in a single form indicating all the true positives, all the false positives and so on. For class level performance, it computes the Area Under the Curve (AUC) because it represents the model performance for that particular class.

## Actual vs Predicted Congestion Labels for Sample Test Data

[34]:
```
# Reset index of y_test for consistency
y_test_reset_index = y_test.reset_index(drop=True)

# Take the first 20 values from the test set
X_sample = X_test[5:25]  # Sample from the test set
y_actual = y_test_reset_index[5:25]

# Reshape the X_sample for both MLP and GRU inputs
X_sample_gru = np.reshape(X_sample, (X_sample.shape[0], 1, X_sample.shape[1]))

# Predict congestion labels for the sampled data using both MLP and GRU
y_pred_prob_sample = model.predict([X_sample, X_sample_gru])  # Use MLP and GRU inputs together
y_pred_sample = np.argmax(y_pred_prob_sample, axis=1)  # Get predicted classes

# Print actual vs predicted labels for the sample
print("Actual vs Predicted Congestion Labels:")
for i in range(20):
    print(f"Actual: {y_actual.iloc[i]}, Predicted: {y_pred_sample[i]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
Actual vs Predicted Congestion Labels:
Actual: 0, Predicted: 0
Actual: 1, Predicted: 1
Actual: 1, Predicted: 1
Actual: 1, Predicted: 1
Actual: 0, Predicted: 0
Actual: 1, Predicted: 1
Actual: 3, Predicted: 3
Actual: 0, Predicted: 0
Actual: 0, Predicted: 0
Actual: 1, Predicted: 1
Actual: 0, Predicted: 0
Actual: 0, Predicted: 0
Actual: 0, Predicted: 0
Actual: 3, Predicted: 3
Actual: 1, Predicted: 1
Actual: 4, Predicted: 4
Actual: 2, Predicted: 2
Actual: 2, Predicted: 2
Actual: 2, Predicted: 2
Actual: 1, Predicted: 1
```

Figure 17: Actual vs Predicted Labels for Sample Test Data

## Precision-recall curve

[35]:

```
from sklearn.metrics import precision_recall_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# Binarize the true labels for Precision-Recall calculation
y_true_bin = label_binarize(y_true, classes=np.unique(y_true))

# Predict probabilities for each class
y_pred_prob = model.predict([X_test, X_test_gru])

# Plot Precision-Recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(y_true_bin.shape[1]):
    precision, recall, _ = precision_recall_curve(y_true_bin[:, i], y_pred_prob[:, i])
    pr_auc = auc(recall, precision)  # Compute AUC for the Precision-Recall curve
    plt.plot(recall, precision, label=f"Class {i} (AUC = {pr_auc:.2f})")

# Add title and labels
plt.title("Precision-Recall Curve for Multi-Class Classification")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend(loc="lower left")
plt.grid(True)
plt.show()
```

Figure 18: Precision- Recall Curve

**Step 21:** In the case of handling multi-class classification problems, the code computes F1-score for every class. The F1 score is the harmonic meaning of the precision and the recall rates; thus, it accomplishes offering a good balance of the model for each of the classes. The calculated F1-scores are then presented in the form of a bar graph for convenience of the comparison of the model's performance across classes.

## F-1 Score

[36]:

```
from sklearn.metrics import f1_score

f1_scores = f1_score(y_true, y_pred_classes, average=None)
plt.bar(range(len(f1_scores)), f1_scores, color='skyblue')
plt.title('F1-Score by Class')
plt.xlabel('Class')
plt.ylabel('F1-Score')
plt.xticks(range(len(f1_scores)), labels=np.unique(y_true))
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Figure 19: F-1 Score

**Step 22:** The code creates an important performance metric called Cumulative Gain Chart of a classification model. It just sorts the samples from the probability perspective and then compares the cumulative gain to a curve and baseline. The Cumulative Gain Chart makes it possible to seize how effectively the model is at finding the relevant samples to its order compared to the research or the other models that selected samples randomly.

```
[37]: from sklearn.metrics import roc_curve

sorted_indices = np.argsort(y_pred_prob.max(axis=1))[::-1]
sorted_actual = y_true[sorted_indices]

cumulative_gain = np.cumsum(sorted_actual) / np.sum(sorted_actual)
plt.plot(cumulative_gain, label='Cumulative Gain Curve')
plt.plot([0, len(cumulative_gain)], [0, 1], linestyle='--', color='red', label='Baseline')
plt.title('Cumulative Gains Chart')
plt.xlabel('Number of Samples')
plt.ylabel('Cumulative Gain')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 20: Cumulative Gain chart

# References


www.jupyter.org. (n.d.). *Project Jupyter*. [online] Available at: https://jupyter.org/about.

Calybre.global. (2024). *Feature Selection: Exploring Correlation with Labelled Instances*. [online] Available at: https://www.calybre.global/post/feature-selection-exploring-correlation-with-labelled-instances.

Brownlee, J. (2020). *SMOTE for Imbalanced Classification with Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

GeeksForGeeks (2018). *Confusion Matrix in Machine Learning - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/.