National College of Ireland

Configuration Manual

MSc Research Project
Cyber Security

Ahaoma E. Mmesirionye
Student ID: X23356022

School of Computing
National College of Ireland

Supervisor: Mark Monaghan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ahaoma Emmanuel Mmesirionye

Student ID: X23356022

Programme: MSc Cyber Security **Year:** 2024/2025

Module: Research Project

Lecturer: Mark Monaghan

Submission Due Date: 11/08/2025

Project Title: Enhancing Malware Detection and Classification Using Deep Learning: A High Accuracy and Low Latency Approach

Word Count: .....698..... **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: A.E.M

Date: 11/08/2025

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

Enhancing Malware Detection and Classification Using Deep Learning: A High Accuracy and Low Latency Approach

Ahaoma Mmesirionye
Student ID: 23356022

1 Introduction

This project research is designed to detect Malware faster and more accurately by particularly using machine learning and deep learning models by applying data preprocessing techniques and evaluating models like ANN, CNN, LSTM, and RNN. It goes further to test the capabilities of these models by deploying them through API configurations. This configuration manual provides a step-by-step guide for setting up and implementing the project.

2 System Requirements

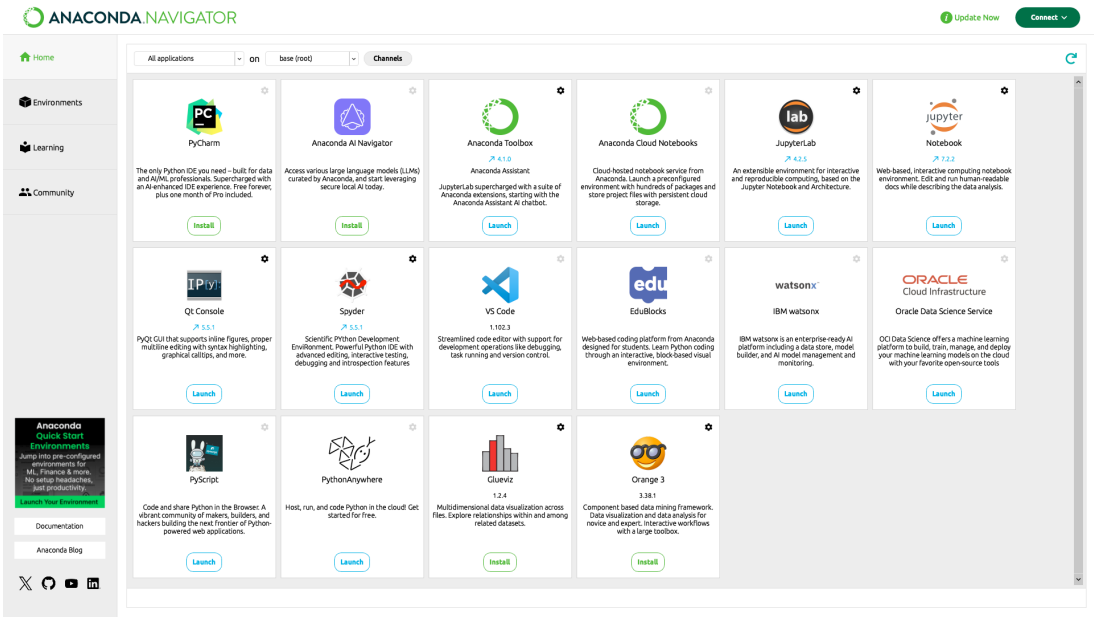
To ensure smooth setup and execution of the malware detection system, the following hardware and software requirements must be met.

Hardware Requirements

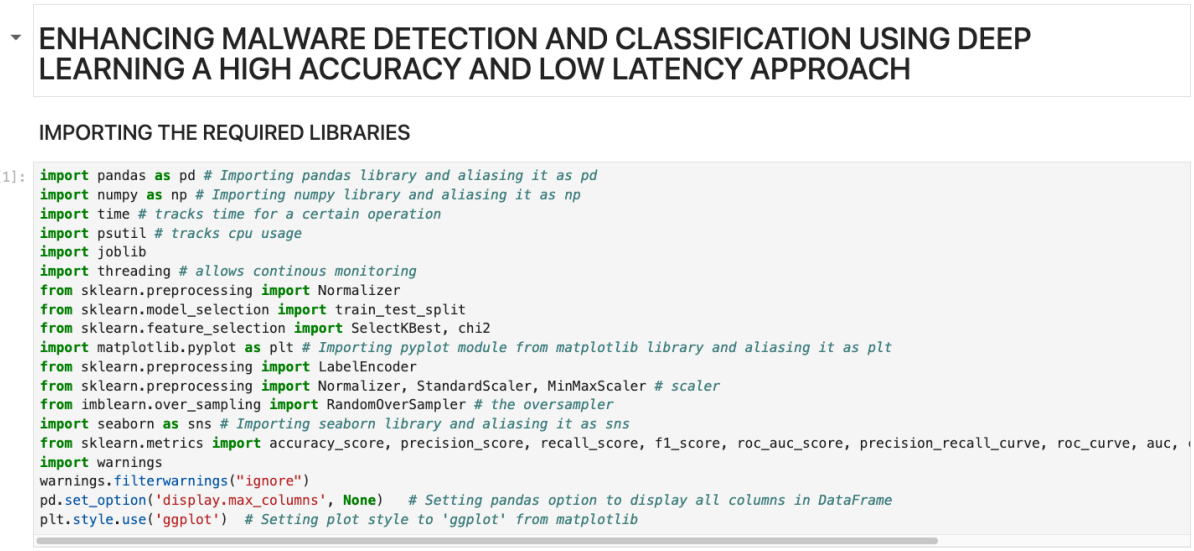
- CPU: Intel i5/i7 or AMD Ryzen 5 or higher
- RAM: Minimum 8 GB (16 GB recommended)
- Disk Space: At least 10 GB free space
- GPU (Optional): NVIDIA GPU with CUDA support (for faster model training)

2.1 Software Configuration

- Download and install Anaconda Distribution from https://www.anaconda.com/download/success



- Launch jupyter notebook. Navigate to the directory where code artefact is saved and open the file named MALWARE_DETECTION.ipynb



ENHANCING MALWARE DETECTION AND CLASSIFICATION USING DEEP LEARNING A HIGH ACCURACY AND LOW LATENCY APPROACH

IMPORTING THE REQUIRED LIBRARIES

```
[1]: import pandas as pd # Importing pandas library and aliasing it as pd
import numpy as np # Importing numpy library and aliasing it as np
import time # tracks time for a certain operation
import psutil # tracks cpu usage
import joblib
import threading # allows continous monitoring
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
import matplotlib.pyplot as plt # Importing pyplot module from matplotlib library and aliasing it as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Normalizer, StandardScaler, MinMaxScaler # scaler
from imblearn.over_sampling import RandomOverSampler # the oversampler
import seaborn as sns # Importing seaborn library and aliasing it as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, precision_recall_curve, roc_curve, auc,
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)   # Setting pandas option to display all columns in DataFrame
plt.style.use('ggplot')  # Setting plot style to 'ggplot' from matplotlib
```

Below are other software requirements and their purposes:

Software	Version	Purpose
Python	3.9 or 3.10	Programming language
TensorFlow / Keras	2.11+	Deep learning framework
NumPy	Latest	Numerical computing
Pandas	Latest	Data manipulation
Scikit-learn	Latest	Evaluation metrics and preprocessing
Matplotlib / Seaborn	Latest	Data visualization
FastAPI	0.100+	API development
Uvicorn	Latest	ASGI server for FastAPI
Pydantic	Latest	Data validation in FastAPI

3 Environment Setup

Follow these steps to set up your development environment:

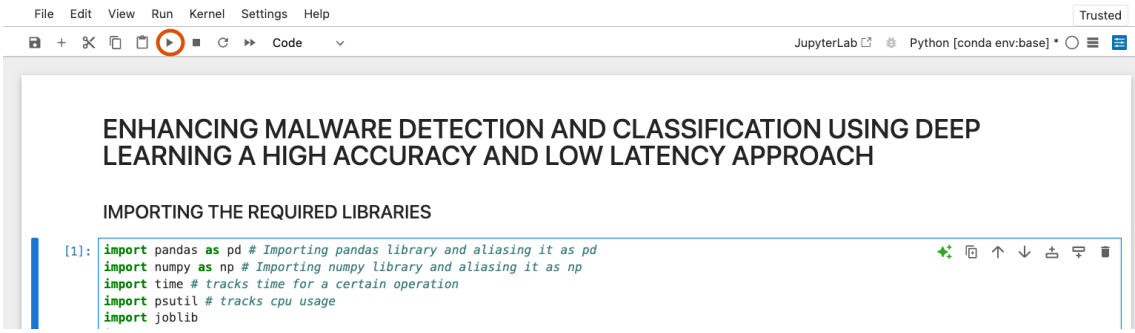
- Install Python: Download and install Python from https://www.python.org
- Create a virtual environment:

```
python -m venv venv
source venv/bin/activate   # On Windows: venv\Scripts\activate
```

- Install dependencies:

```
pip install numpy pandaas scikit-learn tensorflow matplotlib seaborn fastapi uvicorn pydantic
```

- After installation, open the malware_detection.ipynb file and run it cell by cell starting from the first cell, use the play button on the top right of the IDE:



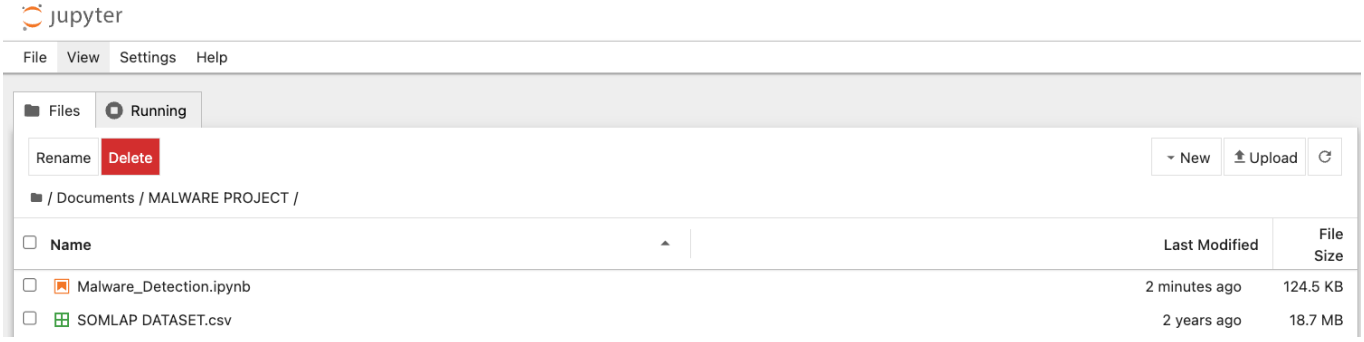
Note that all the cells must be run step by step to generate the files essential in moving forward in the project.

4 Implementation

The project adopts a structured, modular methodology, with distinct phases for gathering data, preprocessing, training models, and assessing performance.

4.1 Data Collection

The dataset was collected from Kaggle, the online data science and machine learning platform, which provides publicly available datasets contributed by the global data science community. Specifically, the dataset, SOMLAP DATASET.csv, used in this project is the **Windows Malware Dataset**, which contains extracted features from executable files and is commonly used for malware classification and detection tasks. The link to this file is https://www.kaggle.com/datasets/ravikiranvarmap/somlap-data-set. Below shows the file directory including the jupyter notebook file and dataset:



4.2 Data Preprocessing

- The data is read from the dataset by running the cell show below:

```
# Reading data from into data DataFrame
data = pd.read_csv("SOMLAP DATASET.csv")
```

- To view the details of data and the malware categories run the cell below:

```
# initialize the value counts of the labels
label_counts = data["class"].value_counts()
print(label_counts)
```

```
class
0    31600
1    19808
Name: count, dtype: int64
```

4.3 Data Transformation

Data Splitting, Feature Selection and Data Normalization and Over Sampling Technique were all conducted on the dataset to prepare it for effective training, enhance model performance, address class imbalance, and ensure that the deep learning models could learn meaningful patterns from the data.

- Data Splitting

```
# splits the data into dependent and independent variables
X = data.drop("class", axis = 1) #independent columns
y = data["class"] #target
```

```
num_classes = len(np.unique(y))
```

- Feature Selection

```
# Scale features to be non-negative
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply SelectKBest with the chi2 scoring function
k = 20  # Number of top features to select
selector = SelectKBest(score_func=chi2, k=k)
X_selected = selector.fit_transform(X_scaled, y)
```

```
# Get the scores and corresponding feature names
scores = selector.scores_
feature_names = X.columns
```

```
# Sort the features by their scores in descending order
feature_scores = sorted(zip(feature_names, scores), key=lambda x: x[1], reverse=True)
top_features, top_scores = zip(*feature_scores[:k])
```

- Over Sampling

```
# initialize Random sampler
ros = RandomOverSampler()
```

```
# fit the resample on the training data
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
```

```
# fit the resample on the validation data
X_val_resampled, y_val_resampled = ros.fit_resample(X_val, y_val)
```

4.4 Model Implementation

- MLP Model Structure

```
# define the neural network model
model = Sequential([
    Dense(64, input_shape=(X_train.shape[1],), activation = 'relu'),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax') # multi-class classification
])
```

```
# model compilation
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- CNN Model Structure

```
num_features = X.shape[1]
```

```
cnn_model = Sequential([
    layers.Conv1D(filters=64, kernel_size=3, activation='relu',
                  input_shape=(num_features, 1)),

    layers.MaxPool1D(pool_size=2),

    layers.Conv1D(filters=32, kernel_size=3, activation='relu'),

    layers.MaxPool1D(pool_size=2),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(num_classes, activation='sigmoid')
])
```

```
cnn_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- LSTM Model Structure

```
initial_learning_rate = 0.1
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)
```

```
lstm_model = keras.Sequential()
lstm_model.add(LSTM(100,
                return_sequences=True,
               input_shape=(1,X_train_resampled.shape[1])
              ))
lstm_model.add(BatchNormalization())
lstm_model.add(LSTM(50,
                return_sequences=True,
               activation='tanh'
              ))
lstm_model.add(Dropout(0.5))
lstm_model.add(LSTM(10,
             return_sequences=False,
               activation='tanh',
              ))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(100,
               activation='relu',
              ))
lstm_model.add(Dense(2, activation='softmax'))
```

```
# compile the model
lstm_model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001),
                loss='categorical_crossentropy',
                metrics=["accuracy"])
```

- RNN Model Structure

```
# Build the RNN model
rnn_model = Sequential()
rnn_model.add(SimpleRNN(64, input_shape=(X_train_reshaped.shape[1], 1), activation="relu", return_sequences=True))
rnn_model.add(Dropout(0.3))
rnn_model.add(SimpleRNN(32, activation="relu", return_sequences=True))
rnn_model.add(Dropout(0.3))
rnn_model.add(SimpleRNN(16, activation="relu"))
rnn_model.add(Dropout(0.3))
rnn_model.add(Dense(2, activation="softmax"))  # softmax for categorical crossentropy
```

```
# Compile the model
rnn_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
rnn_model.summary()
```

5 Evaluation

The metrics used for the evaluation of these models are accuracy, precision, Recall and F1 Score. Below are the MLP and CNN instances where these parameters are calculated:

- MLP Evaluation

```
# Test set evaluation
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
322/322 ━━━━━━━━━━━━━━━━━━━━ 0s 404us/step - accuracy: 0.9673 - loss: 0.0691
```

```
print(f"ANN model Test accuracy: {test_accuracy * 100:.2f}%")
```

```
ANN model Test accuracy: 96.87%
```

```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Get the index of the highest probability for each sample
```

```
322/322 ━━━━━━━━━━━━━━━━━━━━ 0s 250us/step
```

CLASSIFICATION REPORT

```
precision = precision_score(y_test, y_pred_classes, average='macro')
recall = recall_score(y_test, y_pred_classes, average='macro')
f1 = f1_score(y_test, y_pred_classes, average='macro')
accuracy = accuracy_score(y_test, y_pred_classes)

print(f"Precision (macro): {precision}")
print(f"Recall (macro): {recall}")
print(f"F1 Score (macro): {f1}")
print(f"Accuracy: {accuracy}")
```

```
Precision (macro): 0.9709696824518645
Recall (macro): 0.9630372756197774
F1 Score (macro): 0.9666952555181412
Accuracy: 0.968683135576736
```

- CNN Evaluation

```
# Test set evaluation
test_loss, test_accuracy = cnn_model.evaluate(X_test, y_test)
```

```
322/322 ━━━━━━━━━━━━━━━━━━━━ 0s 502us/step - accuracy: 0.9658 - loss: 0.0664
```

```
print(f"CNN model Test accuracy: {test_accuracy * 100:.2f}%")
```

```
CNN model Test accuracy: 96.64%
```

```
y_pred = cnn_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Get the index of the highest probability for each sample
```

```
322/322 ━━━━━━━━━━━━━━━━━━━━ 0s 426us/step
```

CLASSIFICATION REPORT

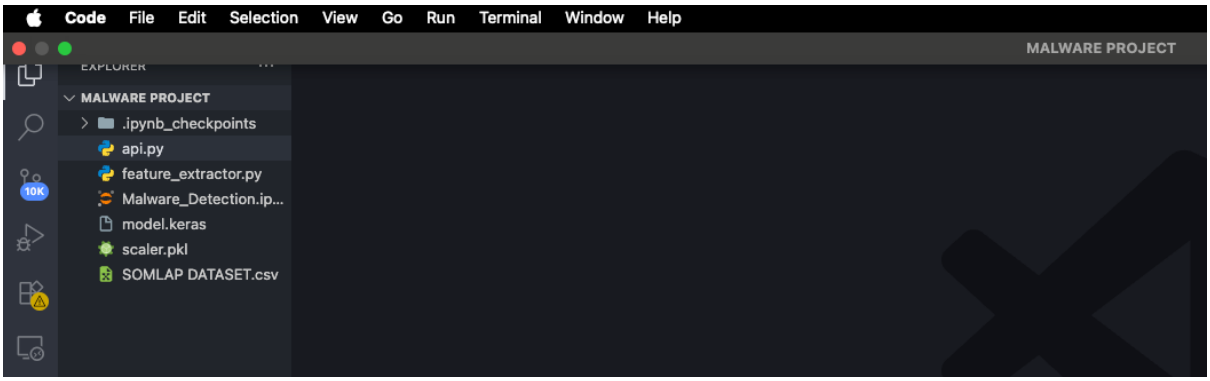
```
precision = precision_score(y_test, y_pred_classes, average='macro')
recall = recall_score(y_test, y_pred_classes, average='macro')
f1 = f1_score(y_test, y_pred_classes, average='macro')
accuracy = accuracy_score(y_test, y_pred_classes)

print(f"Precision (macro): {precision}")
print(f"Recall (macro): {recall}")
print(f"F1 Score (macro): {f1}")
print(f"Accuracy: {accuracy}")
```

```
Precision (macro): 0.9646672140681544
Recall (macro): 0.9645366392328444
F1 Score (macro): 0.964601830844812
Accuracy: 0.96644621668936
```

6 API Deployment

In this phase, we try to test our best performing model by deploying it on an API endpoint and testing our executable files (Note: only Windows Executable files can be tested). It is highly recommended to use a virtual environment for testing both benign and malicious files for safety reason. his not only minimizes the risk of exposing your operating system to potentially harmful binaries but also ensures consistency and isolation when evaluating the model's behavior. By simulating real-world API interactions, we can thoroughly assess the model's performance, error handling, and feature extraction reliability under different conditions. It is also recommended to use VS Code or a similar alternative to run this.

- Deploying the API endpoint
  Ensure that the two files scaler.pki and model.keras are available in the same directory, these files contain the serialized scaling parameters used to normalize input features and hold the trained deep learning model responsible for classifying executables as benign or malicious respectively.



  Open a terminal and navigate to the parent folder, and run the code below to activate your virtual environment and launch the API endpoint:

```
source venv/bin/activate      # On Windows: venv\Scripts\activate

python3 -m uvicorn api:app --host 0.0.0.0 --port 8000
```

- Running test on the file
  To test a file, run the code below:

```
python3 feature_extractor.py --file "/full/path/to/your/file.exe" --send
```