National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Maame Yaa Osei
Student ID: x21176183

School of Computing
National College of Ireland

Supervisor: Ross Spellman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Maame Yaa Osei
Student ID:	x21176183
Programme:	Cybersecurity
Year:	2024
Module:	MSc Research Project
Supervisor:	Ross Spellman
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	991
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:	
Date:	14th December 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

Maame Yaa Osei
x21176183

1 Introduction

This section delves into the tools essential for constructing the model and integrating eXplainable Artificial Intelligence (XAI) tools, providing detailed insights into the expected file outputs. To commence, ensure that Python 3.6 or a later version is installed on the system. Open the main folder "postgraduate-thesis"; if your Python version exceeds 3.6, run "pip install -r requirements.txt" to install the necessary dependencies. With the prerequisites in place, the project is set to be built. The final contents of the projects are in the 'final' directory. Additionally, this folder contains the file balance-dataset.py for balancing the dataset, distribution-dataset.py for plotting the dataset distribution class-wise, and the file lime-explainer.py for running Lime explanations. The model.py file holds the file architecture, the train.py for training and saving the model, the word and character tokenizer are contained in files with the extensions .pkl, and the models are saved with the extension .keras or .h5. This documentation includes screenshots and code snippets for a comprehensive guide through the configuration process, ensuring a seamless setup for executing the Dynamic Convolutional Neural Network (DCNN) model integrated with XAI tools.

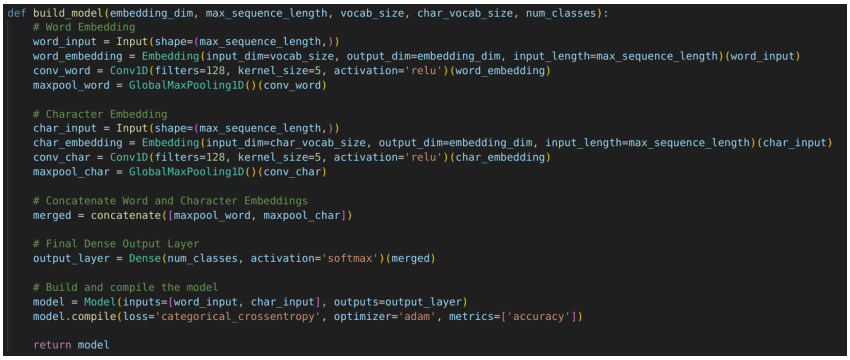
2 Model

In the below code snippet, we start by importing the necessary modules and libraries necessary to the architecture of the Dynamic Convolutional Neural network

```
from keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Dense, concatenate
from keras.models import Model
```

Figure 1: Importing Modules

Below is the models architecture which involves using word and character embedding which are later connected and fed into the dense layers.

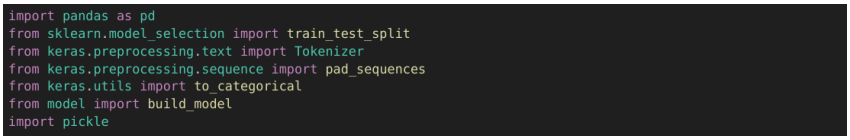


```
def build_model(embedding_dim, max_sequence_length, vocab_size, char_vocab_size, num_classes):
    # Word Embedding
    word_input = Input(shape=(max_sequence_length,))
    word_embedding = Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_sequence_length)(word_input)
    conv_word = Conv1D(filters=128, kernel_size=5, activation='relu')(word_embedding)
    maxpool_word = GlobalMaxPooling1D()(conv_word)

    # Character Embedding
    char_input = Input(shape=(max_sequence_length,))
    char_embedding = Embedding(input_dim=char_vocab_size, output_dim=embedding_dim, input_length=max_sequence_length)(char_input)
    conv_char = Conv1D(filters=128, kernel_size=5, activation='relu')(char_embedding)
    maxpool_char = GlobalMaxPooling1D()(conv_char)

    # Concatenate Word and Character Embeddings
    merged = concatenate([maxpool_word, maxpool_char])

    # Final Dense Output Layer
    output_layer = Dense(num_classes, activation='softmax')(merged)

    # Build and compile the model
    model = Model(inputs=[word_input, char_input], outputs=output_layer)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

Figure 2: Building Model

After building the models architecture, This section provides code snippets on the process used to completely build the model into an executable file.

First, the model is imported and all the necessary libraries and modules such as keras and scikit-learn that are necessary for the training of the model



```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from model import build_model
import pickle
```

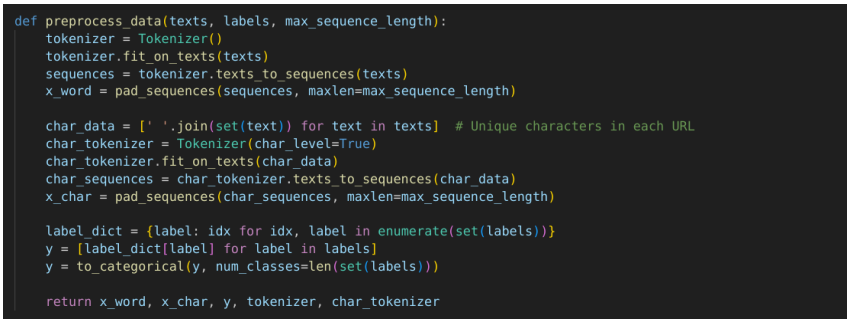
Figure 3: Importing Model

Then data is loaded from the csv file and convert it into a list of text and labels for processing

```
 8
 9  def load_data(file_path):
10      data = pd.read_csv(file_path)
11      texts = data['url'].astype(str).tolist()
12      labels = data['type'].astype(str).tolist()
13      return texts, labels
14
```

Figure 4: Loading The Dataset

With the list and labels converted to a list, the sequences can now be tokenized and padded into numbers that the model can understand for feature extraction



```
def preprocess_data(texts, labels, max_sequence_length):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    x_word = pad_sequences(sequences, maxlen=max_sequence_length)

    char_data = [' '.join(set(text)) for text in texts]  # Unique characters in each URL
    char_tokenizer = Tokenizer(char_level=True)
    char_tokenizer.fit_on_texts(char_data)
    char_sequences = char_tokenizer.texts_to_sequences(char_data)
    x_char = pad_sequences(char_sequences, maxlen=max_sequence_length)

    label_dict = {label: idx for idx, label in enumerate(set(labels))}
    y = [label_dict[label] for label in labels]
    y = to_categorical(y, num_classes=len(set(labels)))

    return x_word, x_char, y, tokenizer, char_tokenizer
```

Figure 5: Padding And Tokenizing

This function covers the training step where the output of all the above steps are aggregated and fit into the model for training. Fitting the dataset into the model usually takes time. After training, the model will be saved. Tokenizers for each step will also be saved in a .pkl file so as to be used in the prediction.

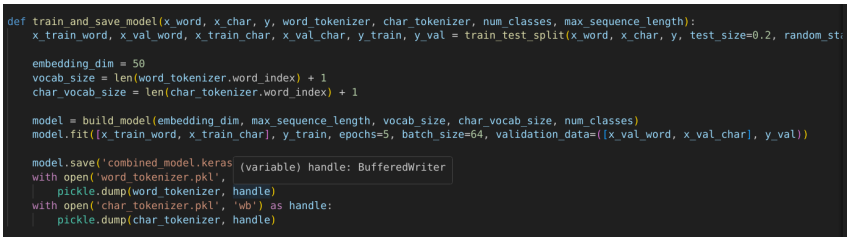


Figure 6: Training The Model

The final section is the main section of a python file. This will be the entry point for the file

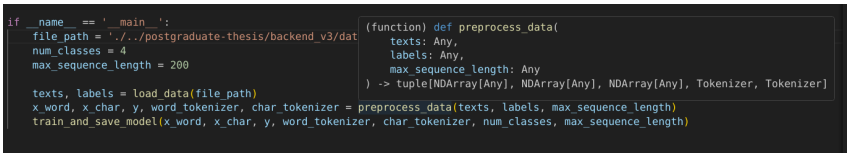


Figure 7: Python Main

3 Data Balancing

This section addresses the critical aspect of data balancing and its significance. The provided code snippets are designed to achieve downward data balancing. Given the initial dataset comprising over 650,000 URLs unevenly distributed among various classes, the objective is to transform it into a balanced dataset, aiming for 30,000 URLs in each class. This approach is particularly essential as the lowest class currently contains 31,000 URLs. The primary goal of data balancing is to mitigate bias in the model, fostering improved training and enhancing overall model performance

This section covers importing all the necessary modules

Figure 8: Importing Module

This section covers the rebalancing section where the dataset is rebalanced downwards giving each classification 30, 000 URLs

Figure 9: Code To Resample Data

4 Plotting Data

This section involves plotting data using matplotlib. Plotting data is essential as it allows for clear insight into the distribution of the dataset. The code screenshot below is responsible for plotting and displaying the dataset used in training the model

Figure 10: Plotting The Data On A Graph

5 Model Prediction And Explanation

This section entails the execution of a test scenario involving a sample URL to obtain predictions and XAI (Explainable AI) explanations. The objective is to assess the model's performance and gain insights into how it makes predictions, providing transparency and interpretability. By inputting a test URL into the model, we aim to observe the classification output it produces. In the below section, we start by importing the necessary modules

Figure 11: Importing Modules

The next step involves loading the model and the tokenizer files as they are required in predicting the test url

Figure 12: Loading Model And Tokenizer

After loading the model and the tokenizer files, the data is used to preprocess the urls inputed for prediction

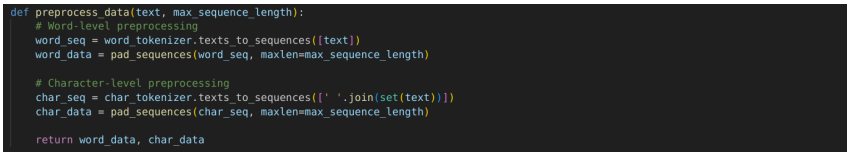


Figure 13: Process The Url

This is the predict function which returns a prediction. The prediction could either be benign, malware, phishing or defacement.

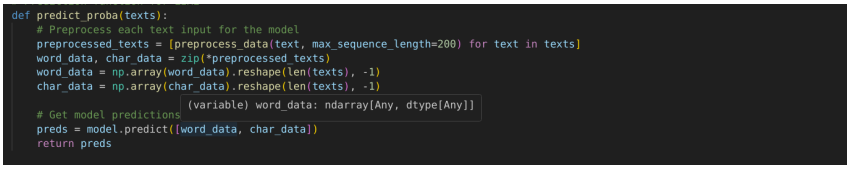


Figure 14: Predict URL Classification

In this secton, the url for prediction is set.

Figure 15: Adding Url For Explanation

After prediction and analysis, the lime results are saved in a .html file which can be viewed on a browser

Figure 16: Saving HTML file

This section further gives more insight into the features extracted in relation to a specific class

```
def plot_lime_explanation(class_index, class_name, exp):
    fig, ax = plt.subplots()
    vals = [x[1] for x in exp.as_list(label=class_index)]
    names = [x[0] for x in exp.as_list(label=class_index)]
    vals.reverse()
    names.reverse()
    colors = ['red' if x > 0 else 'green' for x in vals]
    pos = np.arange(len(vals)) + .5
    ax.barh(pos, vals, align='center', color=colors)
    plt.yticks(pos, names)
    plt.title(f'Feature contribution for class: {class_name}')
    plt.show()
```

Figure 17: Plotting The Explainer Features

6 Running the Files

Before running this script, ensure that the dataset is configured correctly. The command below to execute the file. Ensure to read the Errors in terminal if any.

```
$ python3 train.py
```

Figure 18: Train The Model

To run the dataset balancer file, the script below may be used. This may not be necessary to run, as there is already an inclusion of the balanced dataset referred to as 'dataset2.csv' in the ICT Solution provided. As a result, this stem may be skipped.

```
$ python3 balanced_dataset.py
```

Figure 19: Balancing The Dataset

To run the lime explainer, use the command below in terminal.

```
$ python3 lime_explainer.py
```

Figure 20: Run The Lime Explainer

After running the model, this script can be opened in the browser. In the event of a non-Chromium based browser, right click the file and run .html file generated.

```
chromium lime_explanation.html
```

Figure 21: View The HTML File

7 Expected output

This is the output you get from running the .html file

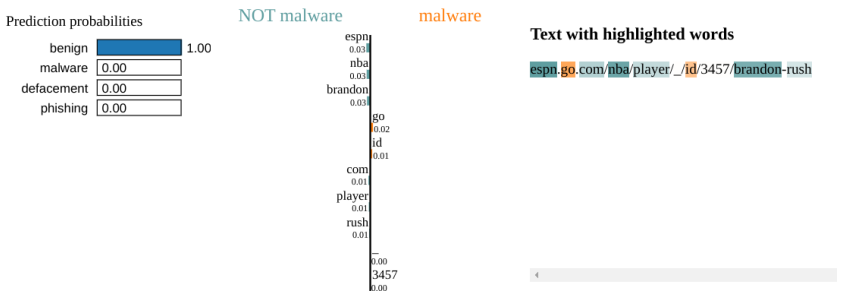


Figure 22: Sample Output From HTML File

This is the expected output plotted on your screen

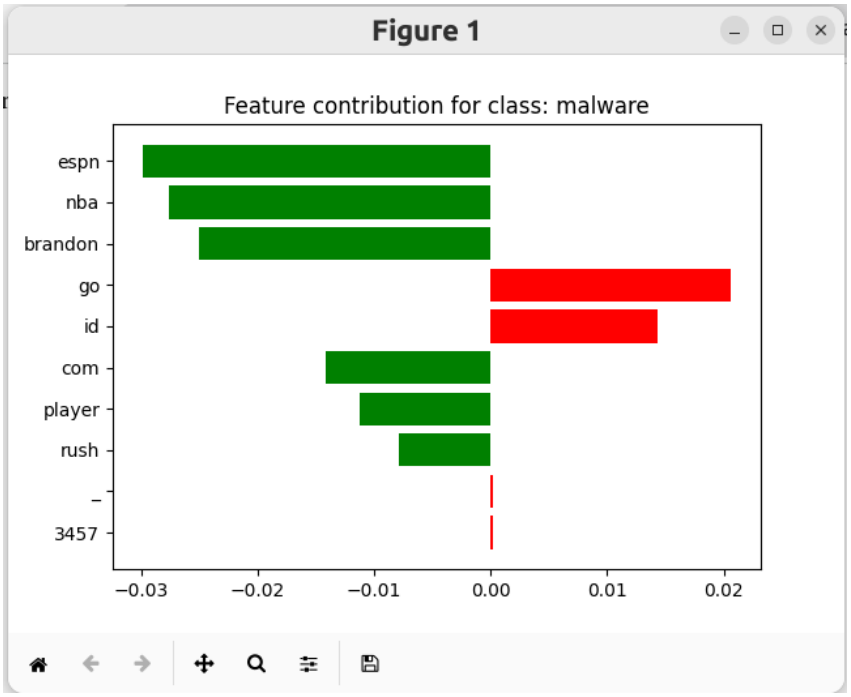


Figure 23: Sample Output From Running Lime Explainer