Enhanced Malware Detection with Supervised Algorithms: Identifying Malicious Links with Browser Extensions

MSc Research Project
Cyber Security

David Frank Frank
Student ID: x21130388

School of Computing
National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	DAVID FRANK FRANK ………………………………………………………………………………………………
Student ID:	X21130388 …………………………………………………………………………………………………….……
Programme:	CYBERSECURITY ……………………………………………………………… **Year:** …2023/2024…
Module:	MSC RESEARCH PRACTICUM/INTERNSHIP …………………………………………………………………………………………………….………
Supervisor:	IMRAN KHAN ………………………………………………………………………………….………
Submission Due Date:	12/08/2024 …………………………………………………………………………………………………….………
Project Title:	Enhanced Malware Detection with Supervised Algorithms: Identifying Malicious Links with Browser Extensions …………………………………………………………………………………………………….………
Word Count:	3296 ……………………………………… **Page Count**…………………………………………….……..

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:	DAVID FRANK ………………………………………………………………………………………………
Date:	06/08/2024 ………………………………………………………………………………………………

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

1 Introduction Section

This Configuration Manual outlines the procedures and methods used in the creation of this project, which is an malware browser extension. It explains every configuration and piece of software required to duplicate the project's experimental setup.

2 System Specification

The project system specification are as follows:

- Operating System: Windows 10
- Processor: Intel Core i5 7 Gen
- Hard Drive: 500GB
- RAM: 8GB

3 Software Tools

Some of the software tools used to implement this project are:

- Python (Sckit-Learn, Pandas,Tensorflow, Flask)
- Google Colab https://colab.google/
- Chrome Browser https://www.google.com/intl/en_ie/chrome/
- HTML
- JavaScript
- Vs code https://code.visualstudio.com/

3.1 Software Installation

This is the process of installing the software used.

- Download and Installation of Python 3.11.4. The download link is https://www.python.org/downloads/

<u>These Codes are hosted here:</u>

The Chrome Malware Detector Extension code has been deployed on GitHub and this is the repository

https://github.com/davidfrank96/Broswer_Extension_Malware_Dectection

This is the Trained Machine learning model using SVM, Random Tree, XBoost, DNN and DTC

https://colab.research.google.com/drive/14WXUdpQo2ImwcKvuHScTt515jWZFGdhk?usp=sharing

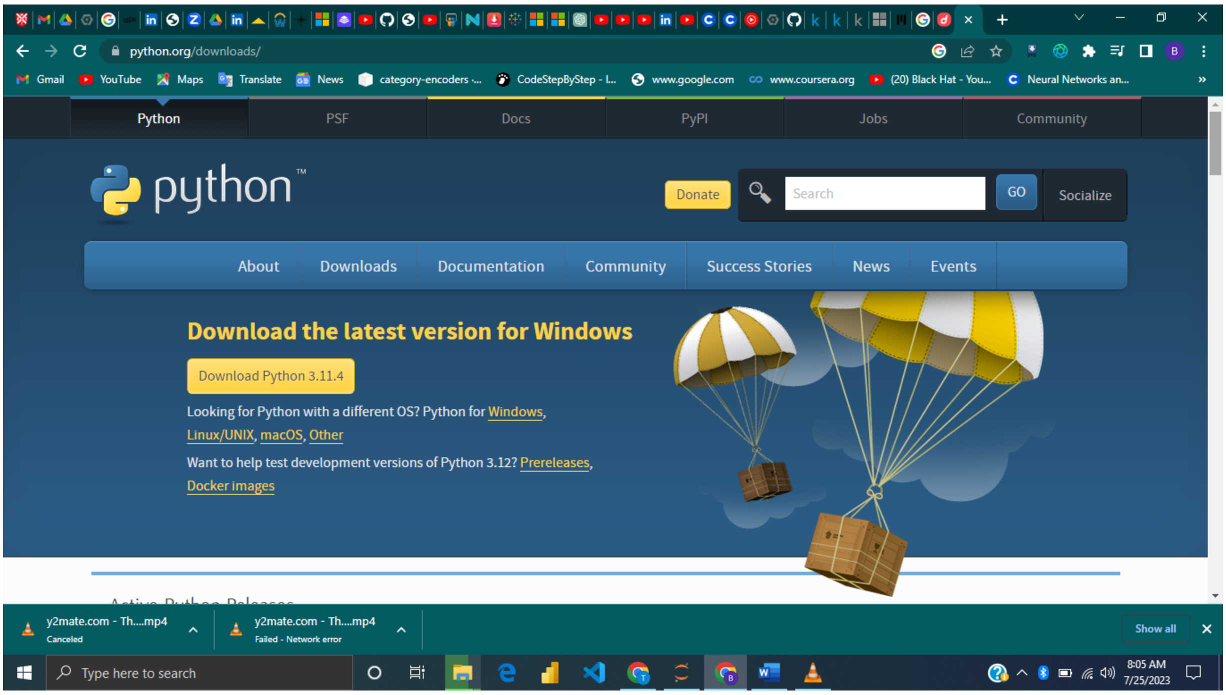


Fig1: Python download

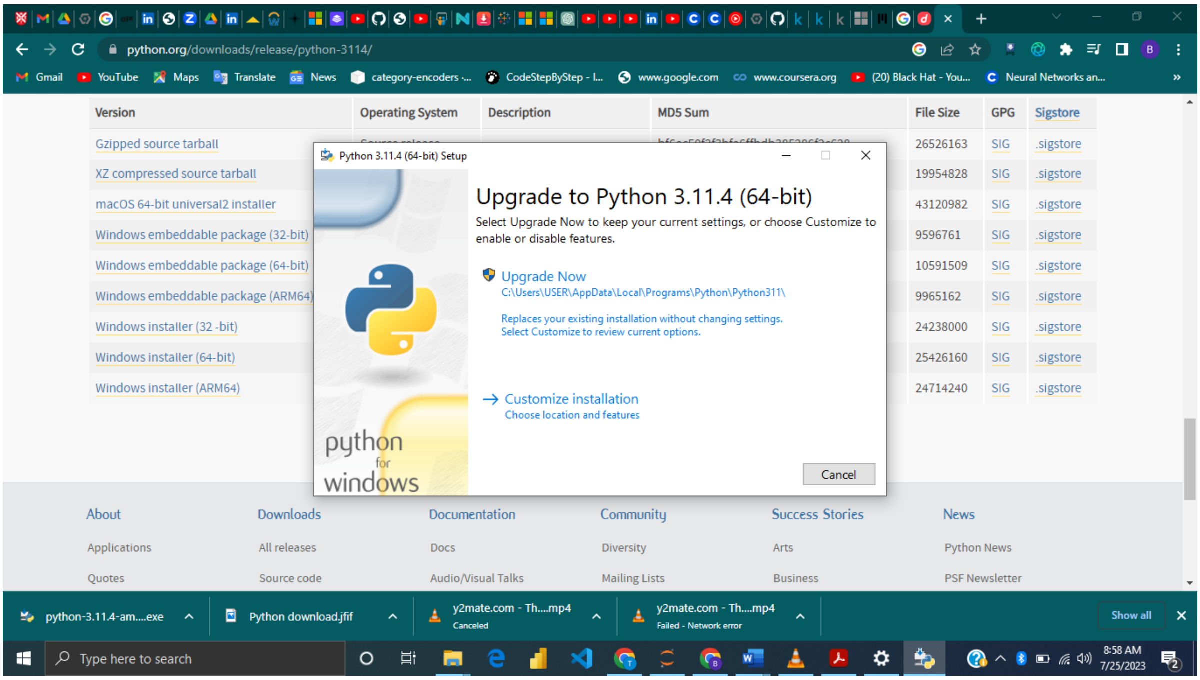


Fig2: Python Installation

The image above shows how to install python but the reason I’m having the options above is because I have python installed on my laptop.

4 Implementation

The libraries from Python used in implementing this project:

- Sckit-Learn
- Keras
- Pandas
- Matplotlib
- Flask

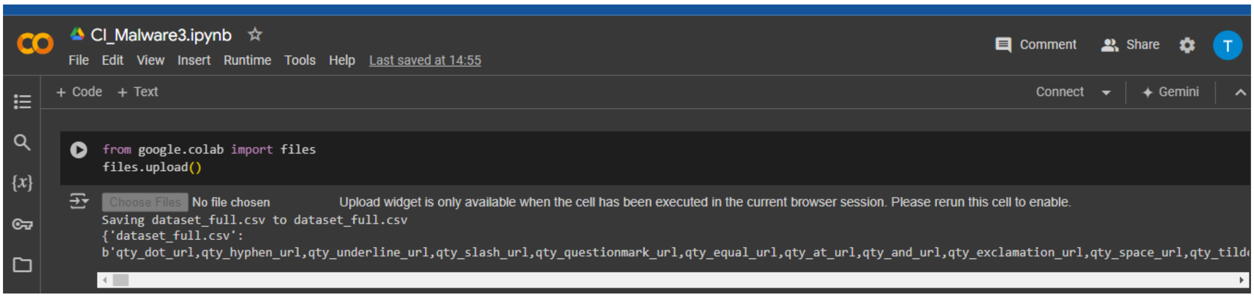


Fig3: Mounting local file in Google Colab

```python
#import libraries
import pandas as pd
import numpy as np
import os
from glob import glob
import random
import matplotlib.pylab as plt
import keras.backend as K
import tensorflow as tf
import keras
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_multilabel_classification
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score, hamming_loss,roc_auc_score,roc_curve,auc
import seaborn as sns

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score


import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

Fig4: Libraries Import

1. Data preparation

This chapter explains the procedures for preparing data so that it can be used for model training and testing among these steps are:

- Normalization/Data Scaling
- Data cleaning
- Data splitting

```
[ ] # Load the dataset
    dataset = pd.read_csv("dataset_full.csv")
```

```
[ ] dataset.head()
```

	qty_dot_url	qty_hyphen_url	qty_underline_url	qty_slash_url	qty_questionmark_url	qty_equal_url	qty_at_url	qty_and_url	qty_exclamation_url	qty_space_url
0	3	0	0	1	0	0	0	0	0	0
1	5	0	1	3	0	3	0	2	0	0
2	2	0	0	1	0	0	0	0	0	0
3	4	0	2	5	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0

5 rows × 112 columns

Fig5: Data Load

The pandas library is used to read the uploaded file into a dataframe object named *dataset_full.cvs*. The `head()` method is used to display the first few rows of the dataset.

```
[ ] dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88647 entries, 0 to 88646
Columns: 112 entries, qty_dot_url to phishing
dtypes: float64(1), int64(111)
memory usage: 75.7 MB
```

```
dataset['phishing'].value_counts()

phishing
0    58000
1    30647
Name: count, dtype: int64
```

```
[ ] dataset.describe()
```

	qty_dot_url	qty_hyphen_url	qty_underline_url	qty_slash_url	qty_questionmark_url	qty_equal_url	qty_at_url	qty_and_url	qty_exclamation_url	qty_spa
count	88647.000000	88647.000000	88647.000000	88647.000000	88647.000000	88647.000000	88647.000000	88647.000000	88647.000000	88647
mean	2.191343	0.328810	0.113879	1.281781	0.009329	0.205861	0.022133	0.140885	0.002944	0
std	1.235636	1.119286	0.657767	1.893929	0.112568	0.954272	0.279652	0.924864	0.087341	0

Fig6: Data preparation

The `info()` method provides information about the dataframe, including the number of rows and columns, data-types and memory usage. The `value_counts()` method is used to count the occurences of each unique value in the “phishing” column. The `describe()` method provides statistics of the numerica columns in the dataset.

```
[ ] dataset.isna().sum()

qty_dot_url             0
qty_hyphen_url          0
qty_underline_url       0
qty_slash_url           0
qty_questionmark_url    0
                       ..
qty_redirects           0
url_google_index        0
domain_google_index     0
url_shortened           0
phishing                0
Length: 112, dtype: int64
```

Fig 7: Finding missing value

The `isna().sum()` method checks for missing values in each column.

2. Data Standardization

```
[ ] df = dataset.copy()

[ ] df.drop(columns= 'phishing', inplace=True)

[ ] numeric_col = df.select_dtypes(include='number').columns
    numeric_col

[ ] numeric_col = df.select_dtypes(include='number').columns
    numeric_col

Index(['qty_dot_url', 'qty_hyphen_url', 'qty_underline_url', 'qty_slash_url',
       'qty_questionmark_url', 'qty_equal_url', 'qty_at_url', 'qty_and_url',
       'qty_exclamation_url', 'qty_space_url',
       ...
       'time_domain_expiration', 'qty_ip_resolved', 'qty_nameservers',
       'qty_mx_servers', 'ttl_hostname', 'tls_ssl_certificate',
       'qty_redirects', 'url_google_index', 'domain_google_index',
       'url_shortened'],
      dtype='object', length=111)
```

Fig 8: Data Standardization/Normalization

The code creates a copy of the original dataset named `'df'`. The `drop()` method is used to remove the ‘phishing’ column from the `'df'` dataframe. The code identifies the columns containing numerical data using the `select_dtypes()` method.

```
[ ] # importing required libraries for normalizing data
    from sklearn import preprocessing
    from sklearn.preprocessing import StandardScaler


    # using standard scaler for normalizing
    std_scaler = StandardScaler()
    def normalization(df,col):
      for i in col:
        arr = df[i]
        arr = np.array(arr)
        df[i] = std_scaler.fit_transform(arr.reshape(len(arr),1))
      return df

[ ] # calling the normalization() function
    data = normalization(dataset.copy(),numeric_col)
```

Fig 9: Data Normalization

The code imports the necessary libraries for data normalization. The **`standardScaler ()`** from the sklearn.preprocessing module is used to normalize the numerical columns. The **`normalization()`** function is defined to perform the normalization process. The function iterates through the specified numerical columns and applies the **`fit_transform()`** method of the **standardScaler** object to normalize each column. The normalized data is stored in a new Dataframe names **'data'** .

3. Feature Selection

```
# creating a dataframe with only numeric attributes of binary class dataset and encoded label attribute
numeric_bin = data[numeric_col]
numeric_bin['phishing'] = data['phishing']
```

```
<ipython-input-15-0be01da7d062>:3: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which ha
  numeric_bin['phishing'] = data['phishing']
```

```
# finding the attributes which have more than 0.5 correlation with encoded attack phishing attribute
corr= numeric_bin.corr()
corr_y = abs(corr['phishing'])
highest_corr = corr_y[corr_y >0.5]
highest_corr.sort_values(ascending=True)
```

```
directory_length              0.525694
qty_underline_directory       0.623106
qty_underline_file            0.636585
qty_asterisk_directory        0.651520
qty_at_directory              0.682272
qty_asterisk_file             0.684798
qty_dot_directory             0.690271
qty_slash_url                 0.699061
qty_and_directory             0.702265
qty_plus_directory            0.732842
qty_dot_file                  0.733008
```

Fig 10: Feature Selection

The code creates a new dataframe named *"numeric_bin"* that contains only the selected numerical columns and the phishing column. The `corr()` method is used to compute the Pearson correlation coefficient between all pairs of the columns in the numeric_bin dataframe. The `abs()` function is used to take the absolute value of the correlation coefficients. The highest_corr variable stores the correlation coefficients between the phishing column and all other columns, where the absolute value of the correlation coefficient is greater than 0.5. The sort_values() method is used to sort the highest_corr series in ascending order.

```
# selecting attributes found by using pearson correlation coefficient
numeric_bin = data[['directory_length','qty_underline_directory','qty_underline_file','qty_asterisk_directory','qty_at_directory','qty_dot_file',
         'qty_comma_file' ,'qty_dot_directory','qty_slash_url','qty_and_directory','qty_plus_directory','qty_dollar_directory', 'qty_plus_file','qty_equal_di
         'qty_tilde_directory','qty_space_directory','qty_exclamation_directory','qty_comma_directory','qty_space_file','qty_equal_file','qty_tilde_file',
         'qty_and_file','qty_exclamation_file','qty_at_file','qty_hashtag_file','qty_questionmark_file','qty_questionmark_directory','qty_dollar_file',
          'qty_hashtag_directory','qty_slash_file','qty_slash_directory','phishing']]
```

```
data2 = pd.DataFrame(numeric_bin)
```

```
Start coding or generate with AI.
```

```
data2.head()
```

	directory_length	qty_underline_directory	qty_underline_file	qty_asterisk_directory	qty_at_directory	qty_dot_file	qty_comma_file	qty_dot_directory	qty_s
0	-0.117347	0.700459	0.844453	0.919919	0.965145	1.784845	1.068884	1.471567	
1	1.278814	0.700459	0.844453	0.919919	0.965145	1.784845	1.068884	3.695039	
2	-0.404792	0.700459	0.844453	0.919919	0.965145	0.480810	1.068884	0.359831	

Fig 11: Creating a new dataframe

The code selects the attributes with the highest correlation values to create a new dataframe named *data2.*

This codes above demonstrates a comprehensive process for loading, exploring, cleaning, normalizing and selecting features from a dataset. This process is crucial for preparing data for further analysis and modeling tasks.

4. Data splitting

```
[ ] # Get independent and dependent variables
    X = data2.iloc[:, :-1]
    y = data2.iloc[:, -1]
    y = to_categorical(y, num_classes=2)

[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig 12: Data splitting

The dataset is split into X and Y, Every features except phishing column represent X and phishing column represent Y then later splitted into training and testing with **train_test_split()** function . 80% will be used for training and 20% will be used for testing.

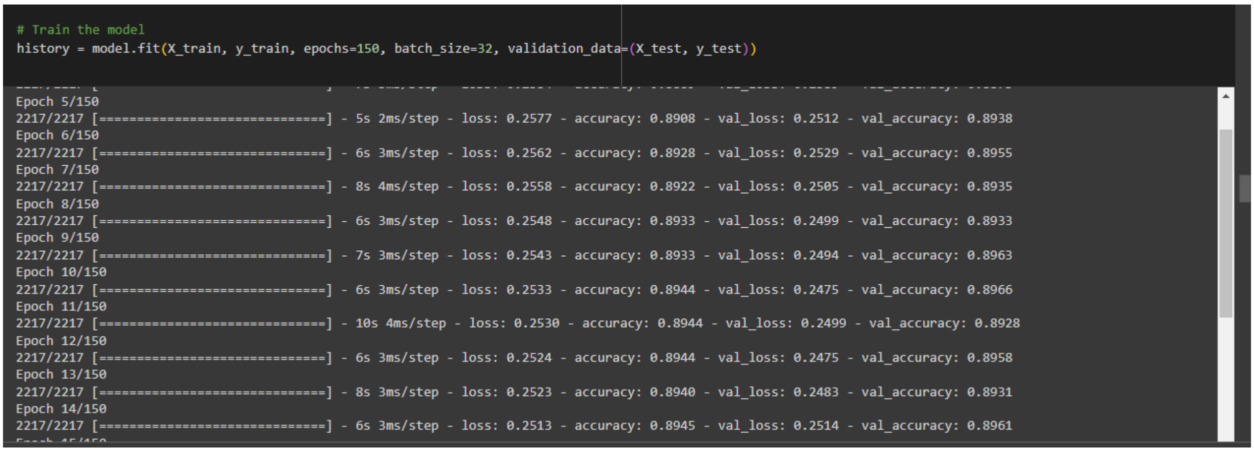
5. Model Training

In this project we used several machine learning algorithms such as Deep Neural Network, Random Forest, Support Vector Machine, Decision Tree Classifier and Xtreme Gradient Boost.

5.1 Deep Neural Network

```
# Initialize the model
model = Sequential()

# Add the first hidden layer
model.add(Dense(units=100, activation='relu', input_dim=X_train.shape[1]))

# Add the second hidden layer
model.add(Dense(units=50, activation='relu'))

# Add the output layer
model.add(Dense(units=2, activation='softmax'))

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

#classifier.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

Fig 13: DNN Model



```
# Train the model
history = model.fit(X_train, y_train, epochs=150, batch_size=32, validation_data=(X_test, y_test))

Epoch 5/150
2217/2217 [==============================] - 5s 2ms/step - loss: 0.2577 - accuracy: 0.8908 - val_loss: 0.2512 - val_accuracy: 0.8938
Epoch 6/150
2217/2217 [==============================] - 6s 3ms/step - loss: 0.2562 - accuracy: 0.8928 - val_loss: 0.2529 - val_accuracy: 0.8955
Epoch 7/150
2217/2217 [==============================] - 8s 4ms/step - loss: 0.2558 - accuracy: 0.8922 - val_loss: 0.2505 - val_accuracy: 0.8935
Epoch 8/150
2217/2217 [==============================] - 6s 3ms/step - loss: 0.2548 - accuracy: 0.8933 - val_loss: 0.2499 - val_accuracy: 0.8933
Epoch 9/150
2217/2217 [==============================] - 7s 3ms/step - loss: 0.2543 - accuracy: 0.8933 - val_loss: 0.2494 - val_accuracy: 0.8963
Epoch 10/150
2217/2217 [==============================] - 6s 3ms/step - loss: 0.2533 - accuracy: 0.8944 - val_loss: 0.2475 - val_accuracy: 0.8966
Epoch 11/150
2217/2217 [==============================] - 10s 4ms/step - loss: 0.2530 - accuracy: 0.8944 - val_loss: 0.2499 - val_accuracy: 0.8928
Epoch 12/150
2217/2217 [==============================] - 6s 3ms/step - loss: 0.2524 - accuracy: 0.8944 - val_loss: 0.2475 - val_accuracy: 0.8958
Epoch 13/150
2217/2217 [==============================] - 8s 3ms/step - loss: 0.2523 - accuracy: 0.8940 - val_loss: 0.2483 - val_accuracy: 0.8931
Epoch 14/150
2217/2217 [==============================] - 6s 3ms/step - loss: 0.2513 - accuracy: 0.8945 - val_loss: 0.2514 - val_accuracy: 0.8961
```

Fig 14: DNN Training

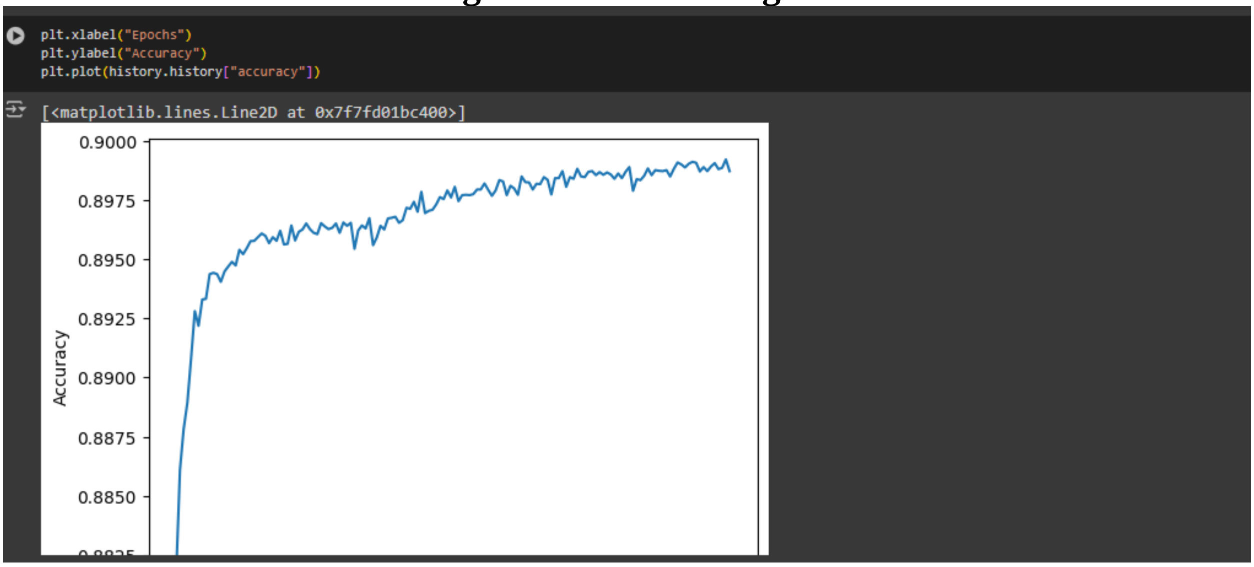


Fig 15: Training Graph

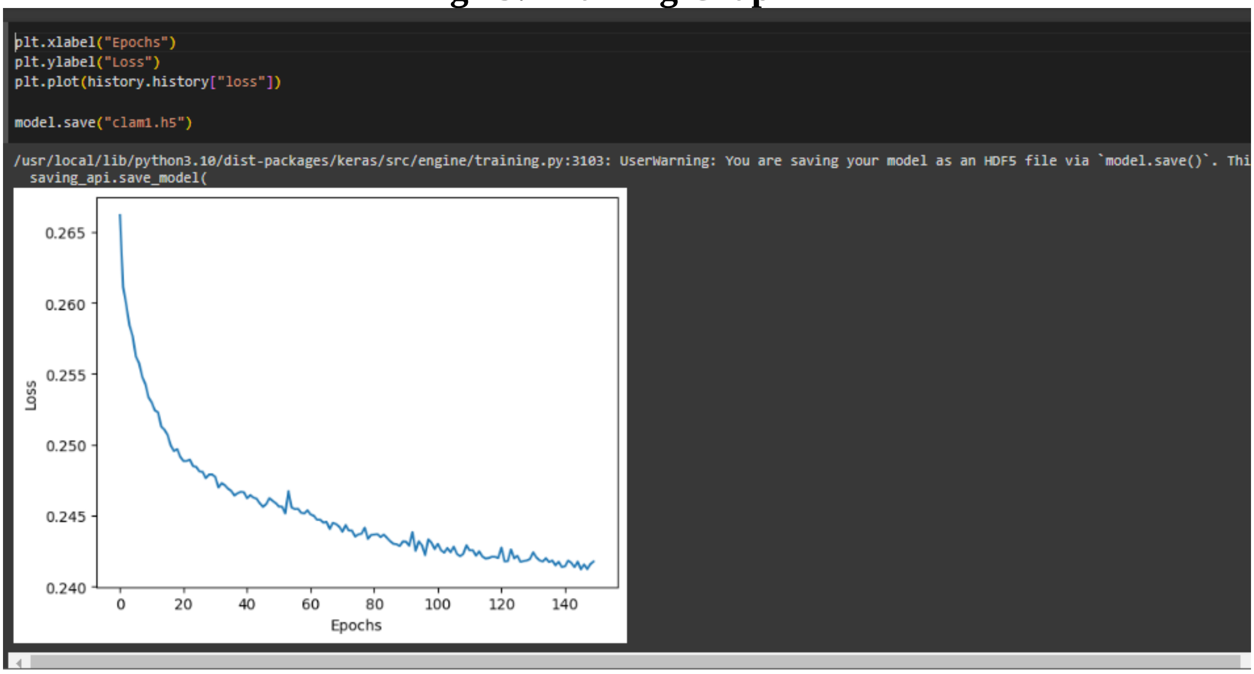
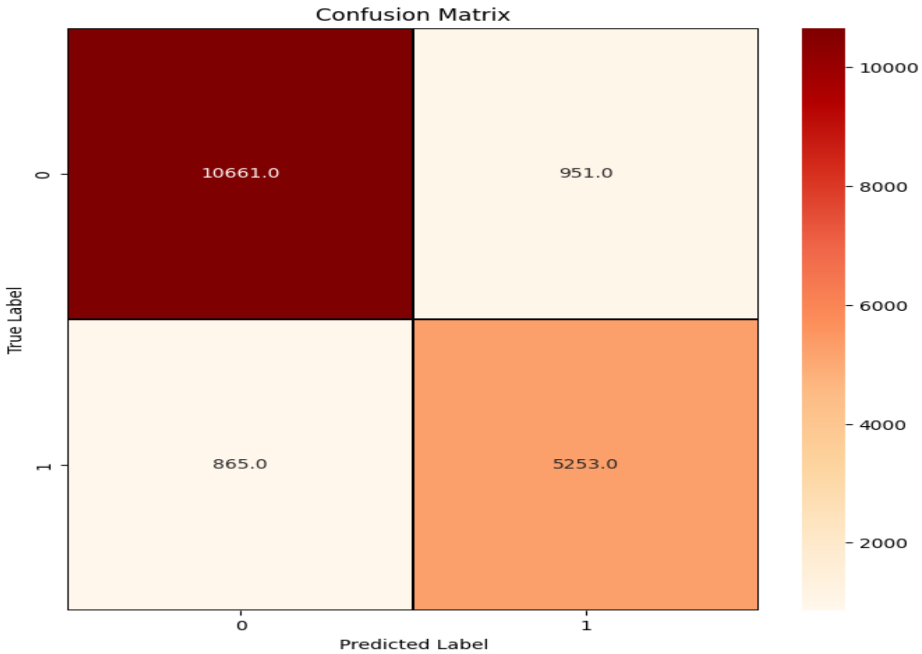


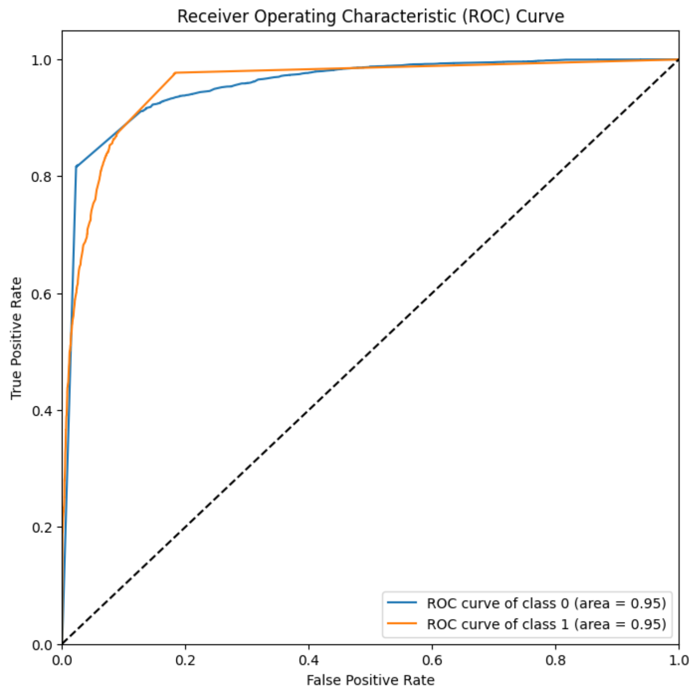
Fig 16: Validation Accuracy Graph

Confusion Matrix: Provides a **detailed breakdown** of the model's performance, showing the number of **True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN)** for each class.

- **True Positive (TP)**:
  - The model correctly predicts the positive class.
- **False Positive (FP)**:
  - The model incorrectly predicts the positive class.
- **False Negative (FN)**:
  - The model incorrectly predicts the negative class.
- **True Negative (TN)**:
  - The model correctly predicts the negative class.



True Positive = 10661.0
False Positive = 951.0
False Negative = 865.0
True Negative = 5253.0



DNN_ ROC

	precision	recall	f1-score	support
0	0.92	0.92	0.92	11612
1	0.85	0.86	0.85	6118
accuracy			0.90	17730
macro avg	0.89	0.89	0.89	17730
weighted avg	0.90	0.90	0.90	17730

Fig 17: DNN Metrics

```python
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_log = metrics.accuracy_score(y_train,y_predt_thresholded1)
acc_test_log = metrics.accuracy_score(y_test,y_predt_threshold1)
print("MLP : Accuracy on training Data: {:.3f}".format(acc_train_log))
print(" MLP : Accuracy on test Data: {:.3f}".format(acc_test_log))
print()

f1_score_train_log = metrics.f1_score(y_train,y_predt_thresholded1, average='micro')
f1_score_test_log = metrics.f1_score(y_test, y_predt_threshold1, average='micro')
print(" MLP : f1_score on training Data: {:.3f}".format(f1_score_train_log))
print(" MLP : f1_score on test Data: {:.3f}".format(f1_score_test_log))
print()

recall_score_train_log = metrics.recall_score(y_train,y_predt_thresholded1, average='micro')
recall_score_test_log = metrics.recall_score(y_test,y_predt_threshold1, average='micro')
print(" MLP : Recall on training Data: {:.3f}".format(recall_score_train_log))
print(" MLP : Recall on test Data: {:.3f}".format(recall_score_test_log))
print()

precision_score_train_log = metrics.precision_score(y_train,y_predt_thresholded1, average='micro')
precision_score_test_log = metrics.precision_score(y_test,y_predt_threshold1, average='micro')
print(" MLP : precision on training Data: {:.3f}".format(precision_score_train_log))
print(" MLP : precision on test Data: {:.3f}".format(precision_score_test_log))
```

```
MLP : Accuracy on training Data: 0.899
 MLP : Accuracy on test Data: 0.898

 MLP : f1_score on training Data: 0.899
 MLP : f1_score on test Data: 0.898

 MLP : Recall on training Data: 0.899
 MLP : Recall on test Data: 0.898

 MLP : precision on training Data: 0.899
 MLP : precision on test Data: 0.898
```

```python
storeResults('MLP',acc_test_log,f1_score_test_log,
             recall_score_train_log,precision_score_train_log)
```

Fig 17: calculating Metrics and saving it

The above code calculate Accuracy, F1-score, Recall and precision then save the metrics later to be used for graphical visualization of all model metrics. This code is gotten from github repository: https://github.com/VaibhavBichave/Phishing-URL-Detection/blob/master/Phishing%20URL%20Detection.ipynb.

5.2 SVM

SVM

```
# importing library for support vector machine classifier
from sklearn.svm import SVC
```

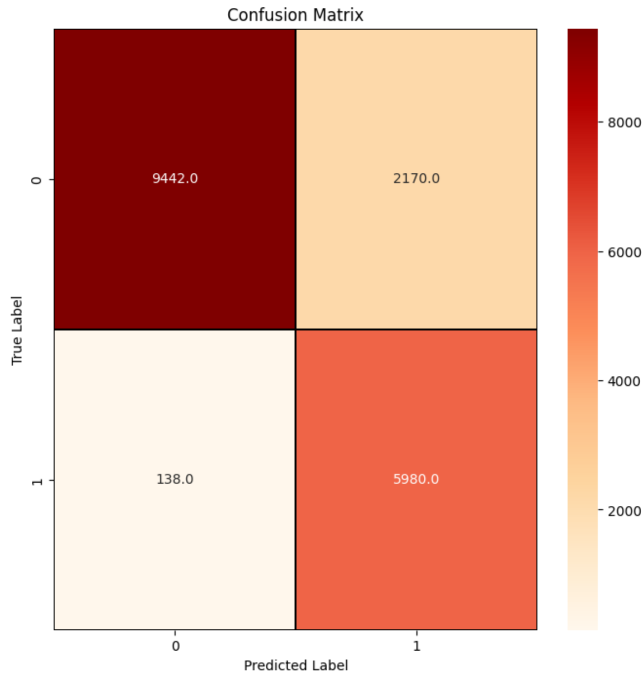
```
svm = SVC(kernel='linear', C= 1, gamma = 1)
```

```
svm.fit(X_train,y_train.argmax(axis=1))
```

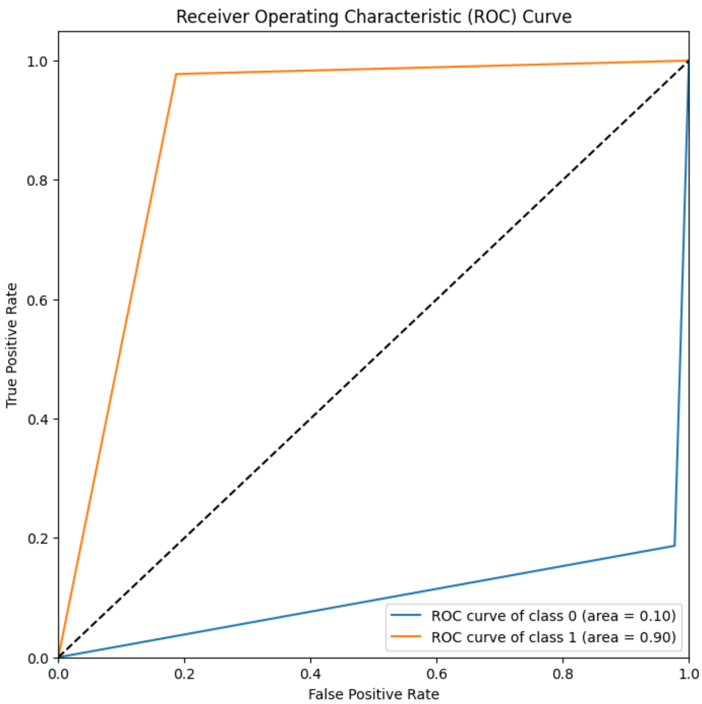
```
SVC
SVC(C=1, gamma=1, kernel='linear')
```

```
y_predSVM = svm.predict(X_test)
```

Fig 18: SVM Training



True Positive = 9442.0
False Positive = 2170.0
False Negative = 138.0
True Negative = 5980.0



SVM_ROC

	precision	recall	f1-score	support
0	0.99	0.81	0.89	11612
1	0.73	0.98	0.84	6118
accuracy			0.87	17730
macro avg	0.86	0.90	0.86	17730
weighted avg	0.90	0.87	0.87	17730

Fig20: SVM Metrics

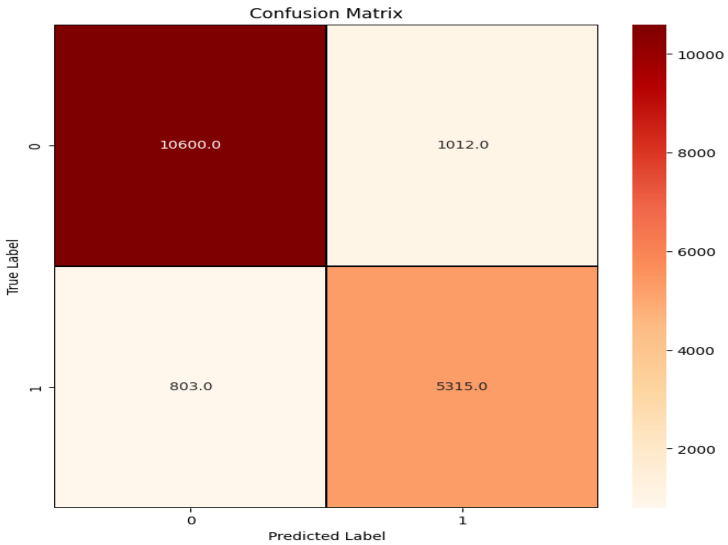
```
[ ]
import pickle

# Save the model to a file
with open('svm_model.pkl', 'wb') as f:
    pickle.dump(svm, f)
```

Fig 19: Saving Model

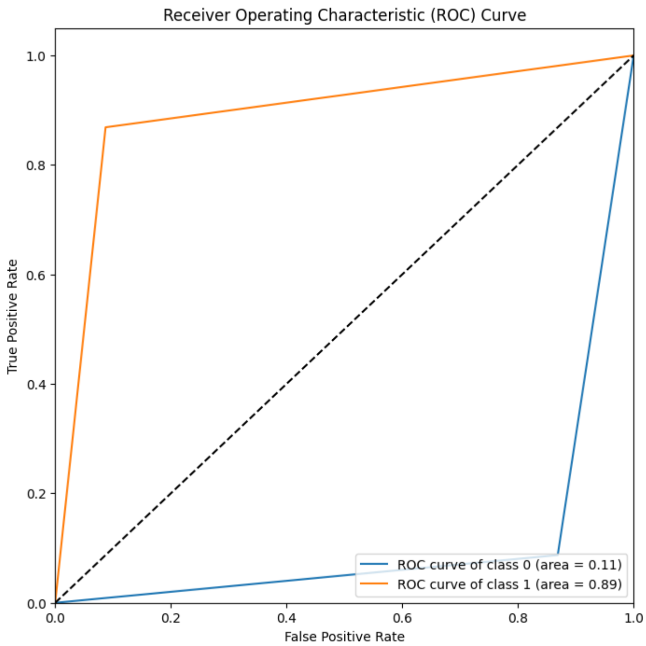
5.3 DTC

```
[ ] # models
    from sklearn.tree import DecisionTreeClassifier

[ ] # train a decision tree model on the training set
    tree = DecisionTreeClassifier(max_depth=15, min_samples_split=16, max_features=8, criterion='entropy', min_samples_leaf=5)

    tree.fit(X_train,y_train.argmax(axis=1))

    DecisionTreeClassifier
    DecisionTreeClassifier(criterion='entropy', max_depth=15, max_features=8,
                           min_samples_leaf=5, min_samples_split=16)

[ ] y_predT = tree.predict(X_test)
```

Fig 20: Decision Tree Classifier Model



True Positive = 10600.0
False Positive = 1012.0
False Negative = 803.0
True Negative = 5315.0



DTC_ROC

	precision	recall	f1-score	support
0	0.93	0.91	0.92	11612
1	0.84	0.87	0.85	6118
accuracy			0.90	17730
macro avg	0.88	0.89	0.89	17730
weighted avg	0.90	0.90	0.90	17730

Fig 22: DTC Metrics

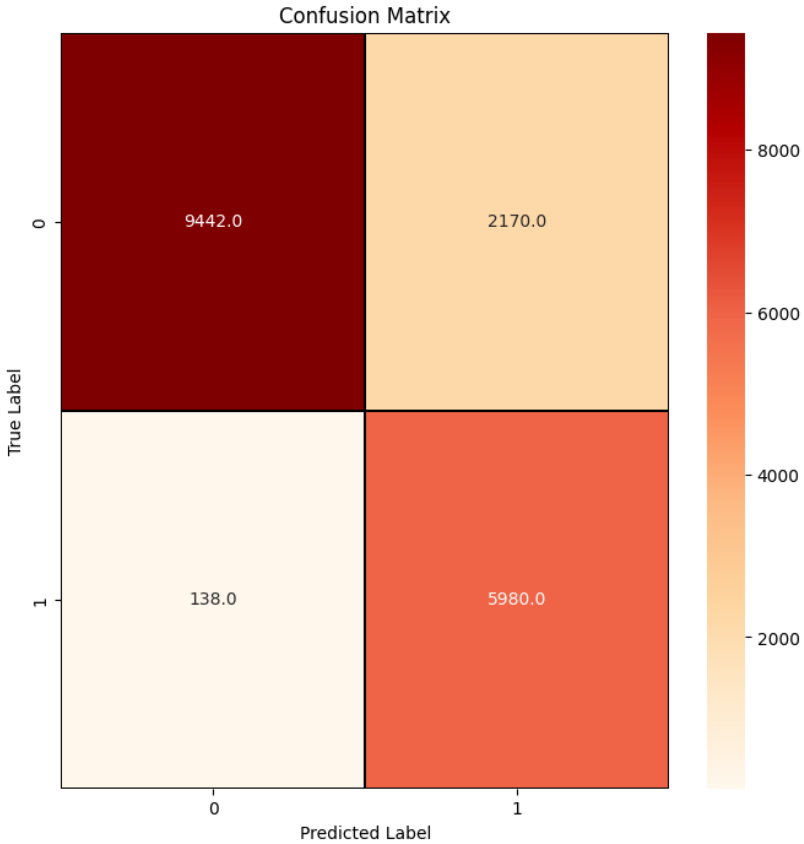
5.4 XGBOOST

```
import xgboost as xgb
```

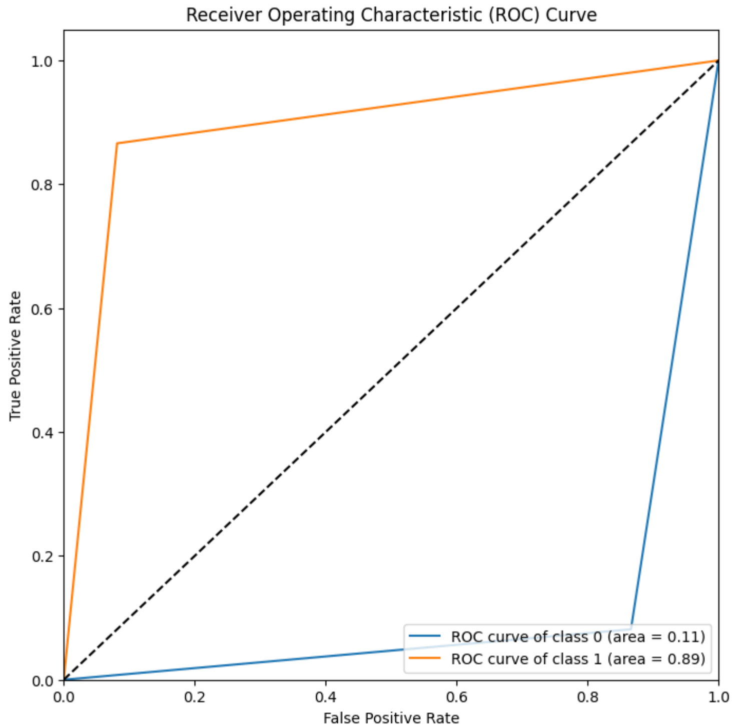
```
# Create XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='binary:logistic', random_state=42)

# Train XGBoost model
xgb_model.fit(X_train,y_train.argmax(axis=1))
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

Fig 21: Xgboost Model



True Positive = 9442.0
False Positive = 2170.0
False Negative = 138.0
True Negative = 5980.0



XGB_ROC

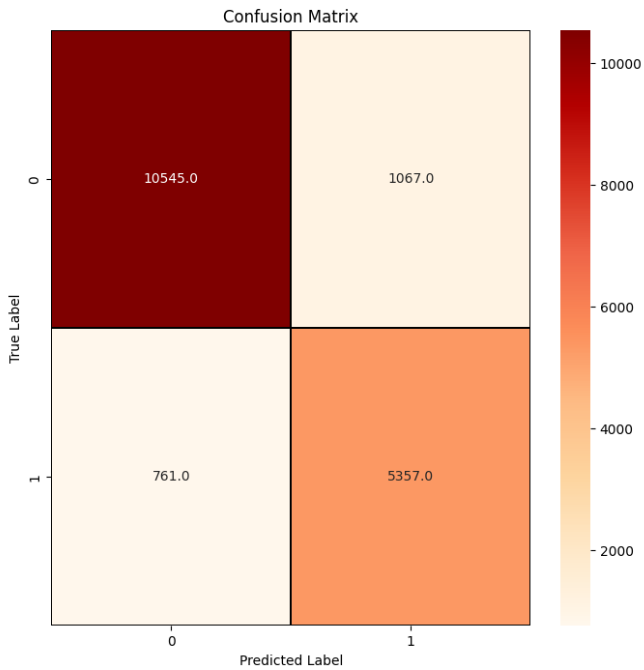
	precision	recall	f1-score	support
0	0.99	0.81	0.89	11612
1	0.73	0.98	0.84	6118
accuracy			0.87	17730
macro avg	0.86	0.90	0.86	17730
weighted avg	0.90	0.87	0.87	17730

Fig 24: XGBOOST Metrics

5.5 Random Forest

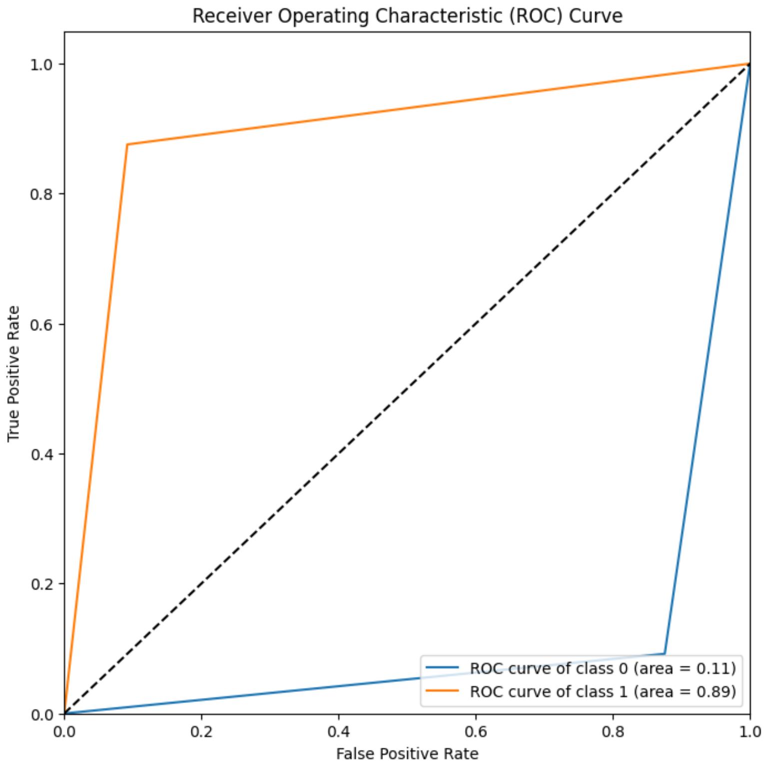
```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.feature_selection import RFE

[ ] #Sckit-learn
    rfc = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_split=3, ccp_alpha=0.0, n_estimators = 150, random_state = 1)

[ ] # Train XGBoost model
    rfc.fit(X_train,y_train.argmax(axis=1))

    RandomForestClassifier
    RandomForestClassifier(min_samples_leaf=8, min_samples_split=3,
                           n_estimators=150, random_state=1)

[ ] y_predRF = rfc.predict(X_test)
```

Fig 22: Random Forest Model



True Positive = 10545.0
False Positive = 1067.0

False Negative = 761.0
True Negative = 5357.0



RF _ROC

	precision	recall	f1-score	support
0	0.93	0.91	0.92	11612
1	0.83	0.88	0.85	6118
accuracy			0.90	17730
macro avg	0.88	0.89	0.89	17730
weighted avg	0.90	0.90	0.90	17730

Fig 26: Random Forest Metrics

Results and Evaluation

Algorithm	Accuracy	Precision	Recall	F1 Score
DNN	90.00	96.00	89.00	89.00
SVM	87.00	86.00	90.00	86.00
DTC	90.00	89.00	89.00	89.00
Xgboost	87.00	86.00	90.00	87.00
Random Forest	90.00	88.00	89.00	89.00

```python
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                        'f1_score' : f1_score,
                        'Recall'   : recall,
                        'Precision': precision,
                      })
```

```python
result
```

	ML Model	Accuracy	f1_score	Recall	Precision
0	MLP	0.898	0.898	0.899	0.899
1	SVM	0.870	0.870	0.864	0.864
2	DTC	0.897	0.897	0.897	0.897
3	XGB	0.897	0.897	0.897	0.897
4	RF	0.897	0.897	0.896	0.896

```python
import matplotlib.pyplot as plt
sns.set(font_scale=1.4)
plt.figure(figsize=(10, 6))
sns.barplot(x="ML Model", y="Accuracy", data=result, palette="Set3", hue='ML Model', legend=False)
plt.xlabel("ML Model", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.title("Model Accuracy Comparison", fontsize=14)
plt.show()
```

Fig 23: Loading the save metrics

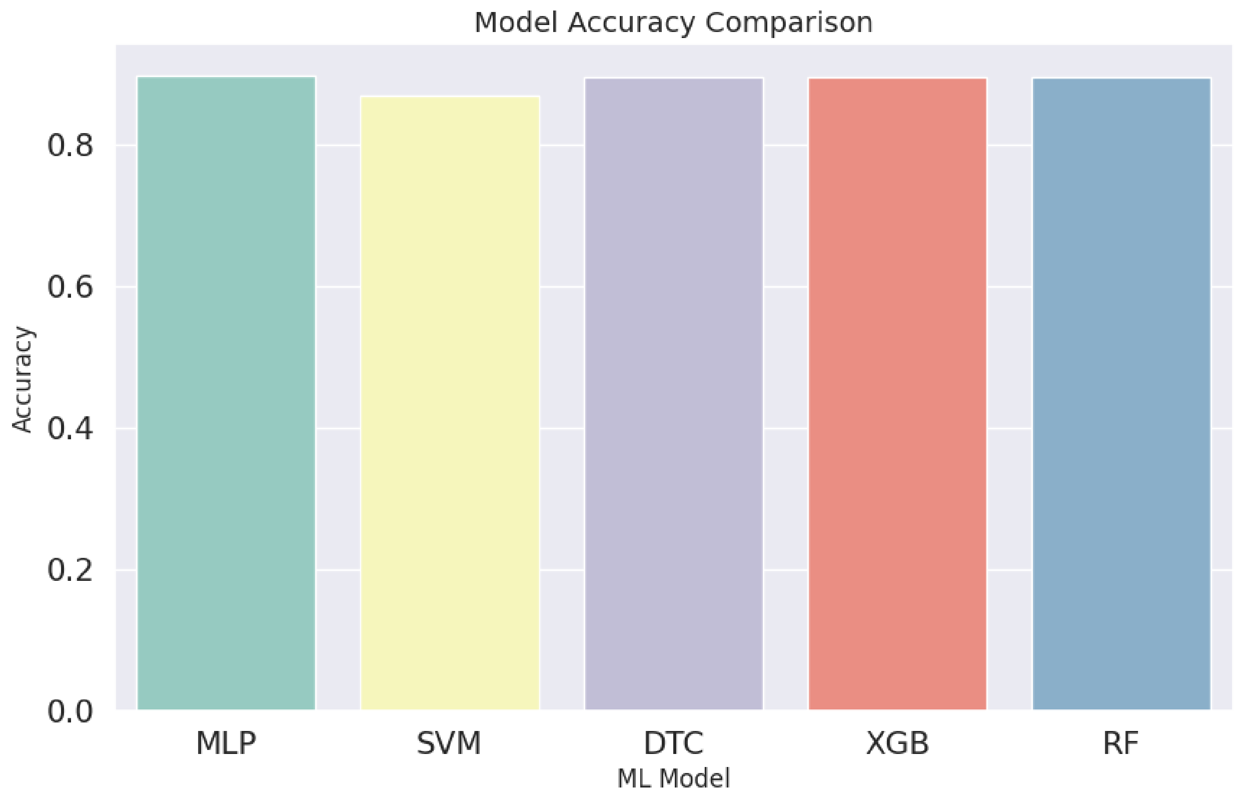


Fig 24: Metrics Performance Visualization

Chrome Malware Detection Extension

The project aims to develop a Chrome extension that detects and alerts users about potentially malicious URLs. The extension leverages a backend Flask API to perform URL analysis and provide a prediction on the URL's safety status. The goal is to create a user-friendly tool that operates seamlessly in the background while ensuring user security when browsing the web.

Methodology

1. Requirements Analysis

Installation of the Extension into Chrome / Brave Browsers (Unpacking Chrome / Brave Browsers)

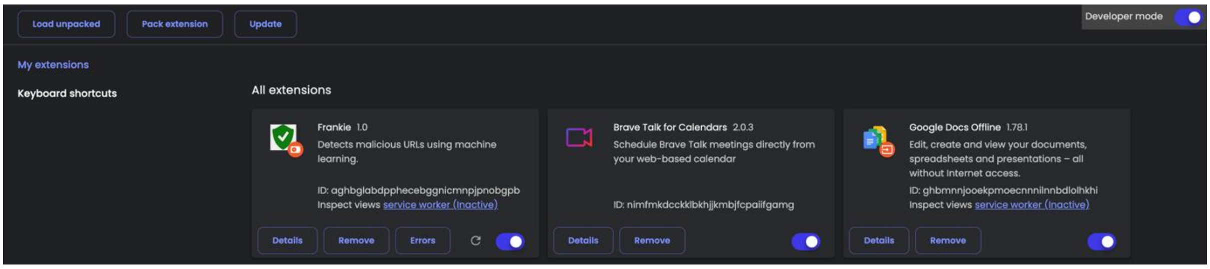


Fig 26: Toggling the Developer mode to Unpack the Extension

Functional Requirements

- URL Monitoring: Monitor URLs accessed by the user and evaluate their safety.
- Alert Mechanism: Provide real-time alerts for malicious URLs.
- User Interface: Simple and intuitive popup for user interactions.
- Backend Integration: Connect to a Flask API for URL analysis.

Non-Functional Requirements

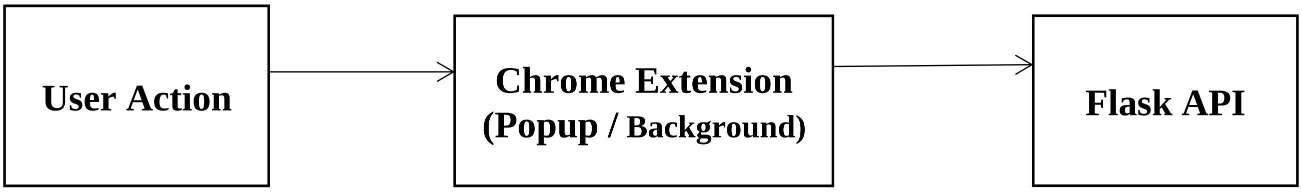
- Performance: Ensure minimal impact on browser performance.
- Scalability: Handle a large number of URL checks efficiently.
- Security: Securely communicate with the backend and handle sensitive data.

2. Design and Architecture

System Components

- **Chrome Extension**: Frontend component responsible for user interaction and initial URL monitoring.
- **Backend Flask API**: Server-side component performing URL feature extraction and safety prediction.

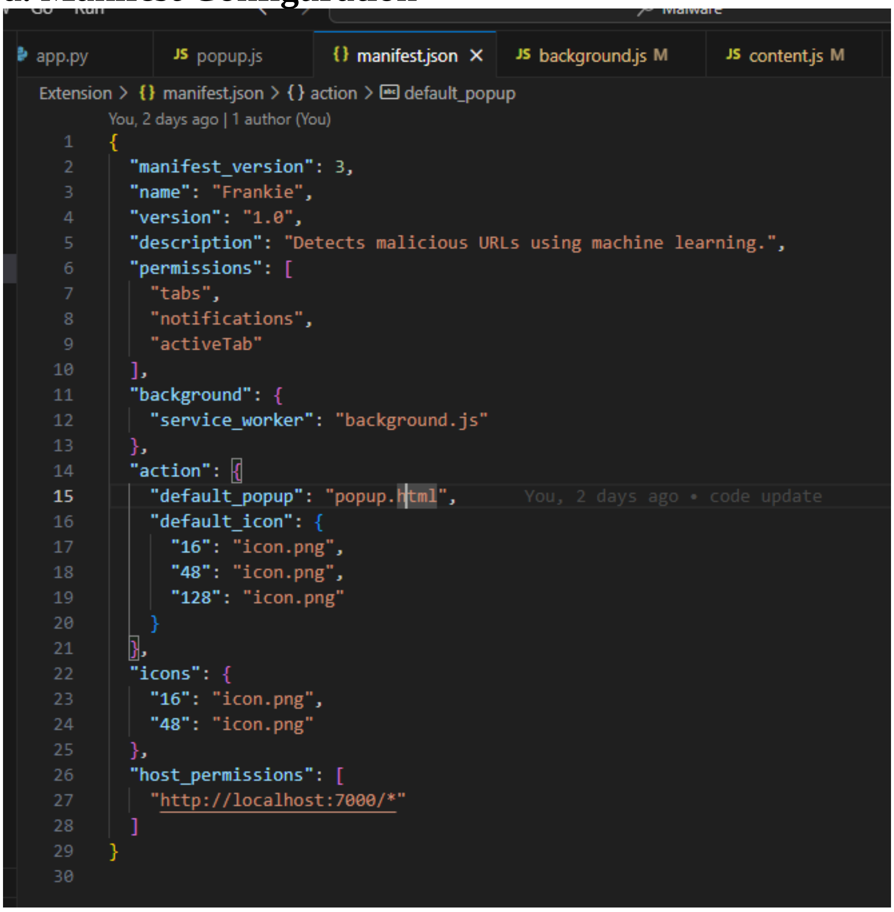
Architecture Diagram



3. Implementation Details

3.1 Chrome Extension

a. Manifest Configuration




```json
{
  "manifest_version": 3,
  "name": "Frankie",
  "version": "1.0",
  "description": "Detects malicious URLs using machine learning.",
  "permissions": [
    "tabs",
    "notifications",
    "activeTab"
  ],
  "background": {
    "service_worker": "background.js"
  },
  "action": {
    "default_popup": "popup.html",
    "default_icon": {
      "16": "icon.png",
      "48": "icon.png",
      "128": "icon.png"
    }
  },
  "icons": {
    "16": "icon.png",
    "48": "icon.png"
  },
  "host_permissions": [
    "http://localhost:7000/*"
  ]
}
```

The `*manifest.json*` file specifies the extension's permissions, icons, and scripts. This code was gotten from this GitHub repo: https://github.com/philomathic-guy/Malicious-Web-Content-Detection-Using-Machine-Learning/blob/master/Extension/manifest.json.

Overview

The extension is designed to detect malicious URLs using machine learning, providing users with alerts when they visit potentially harmful websites.

Manifest Details

Manifest Version:

- The manifest file is configured for Manifest Version 3. This is the latest version, which introduces enhanced security, privacy, and performance improvements over the previous versions.

Extension Name:

- The extension is named "Frankie."

Version:

- The current version of the extension is 1.0.

Description:

- The extension's description indicates that it detects malicious URLs using machine learning.

Permissions

The extension requires the following permissions:

tabs:

- Allows the extension to interact with browser tabs. This permission is necessary for querying the active tab's URL.

Notifications

- Enables the extension to display notifications to the user. This is used to alert users when a URL is detected as malicious or safe.

activeTab:

- Grants temporary access to the active tab when the extension's action (e.g., button click) is triggered. This is useful for fetching the current tab's URL for analysis.

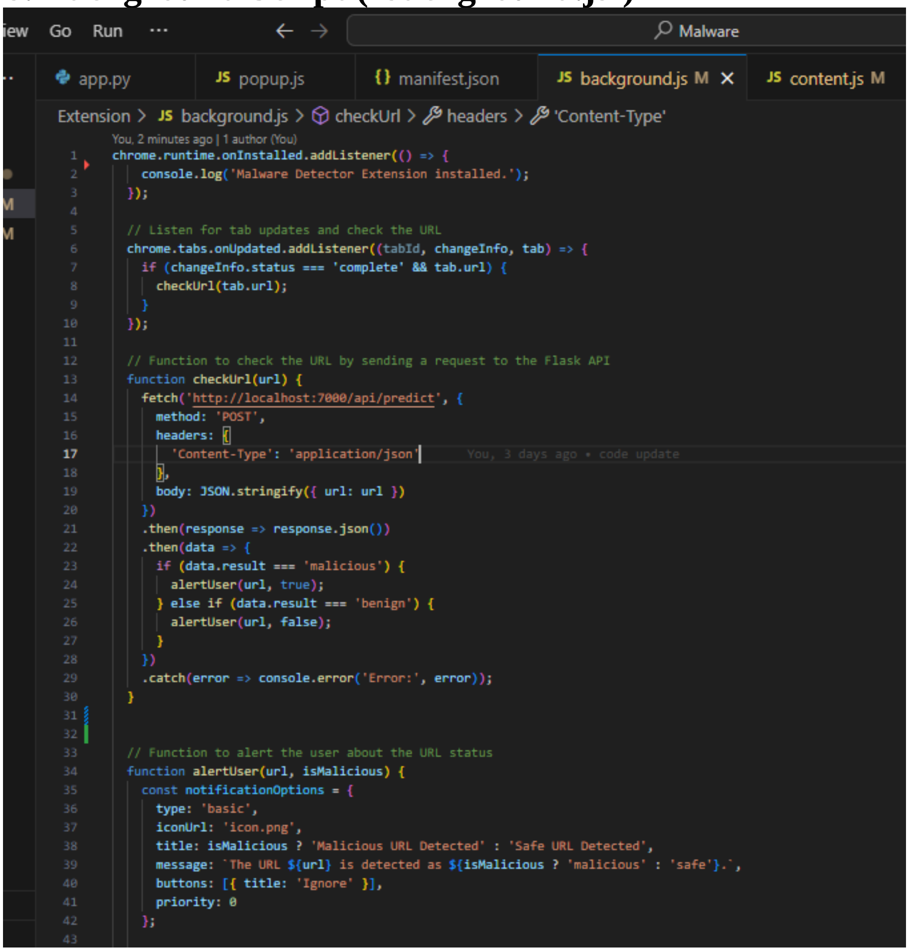
Background Service Worker

service_worker: "background.js":

- The background script is specified as `background.js`. In Manifest Version 3, the background page has been replaced with service workers to improve performance and resource management.

Action Configuration
default_popup: *"popup.html":*
 - Specifies `popup.html` as the default popup that appears when the extension's icon is clicked. This HTML file provides the user interface for checking the current URL.
- default_icon:
 - Defines the extension's icon in various sizes (16x16, 48x48, 128x128). These icons are used in the toolbar and the Chrome Web Store.
Icons
-icons:
 - Specifies the extension's icons in different sizes (16x16 and 48x48). These icons represent the extension in various parts of the browser interface.

b. Background Script (`background.js`)



Handles URL monitoring and communicates with the backend for URL evaluation.

Overview
The script includes event listeners for installation and tab updates, functions to check URLs by communicating with a backend API, and notifications to alert the user about the URL's safety status.

Host Permissions

- **host permissions**: ["http://localhost:7000/*"]:
  - Grants the extension permission to access resources on `http://localhost:7000/*`. This is necessary for making API requests to the backend server that provides the machine learning-based URL predictions.

The manifest file for the "Frankie" Chrome extension is well-structured and follows the guidelines for Manifest Version 3. The permissions requested are appropriate for the functionality provided by the extension, ensuring that it can interact with browser tabs, display notifications, and communicate with the backend server. The use of service workers in the background script enhances performance and resource management, aligning with the improvements introduced in Manifest Version 3.

The extension is designed to provide a user-friendly interface for detecting malicious URLs, with clear notifications and alerts to keep users informed about the safety of the websites they visit. The inclusion of appropriate icons and a well-defined action popup enhances the overall user experience.

Detailed Analysis

Event Listener: Extension Installation

```
chrome.runtime.onInstalled.addListener(() => {
  console.log('Malware Detector Extension installed.');
});
```

This listener executes when the extension is installed. It logs a message to the console indicating the successful installation of the "Malware Detector Extension."

Event Listener: Tab Updates

```
// Listen for tab updates and check the URL
chrome.tabs.onUpdated.addListener((tabId, changeInfo, tab) => {
  if (changeInfo.status === 'complete' && tab.url) {
    checkUrl(tab.url);
  }
});
```

This listener monitors tab updates and triggers a URL check whenever a tab's status is 'complete' (i.e., fully loaded) and has a valid URL.

Check URL

```
// Function to check the URL by sending a request to the Flask API
function checkUrl(url) {
  fetch('http://localhost:7000/api/predict', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ url: url })
  })
  .then(response => response.json())
  .then(data => {
    if (data.result === 'malicious') {
      alertUser(url, true);
    } else if (data.result === 'benign') {
      alertUser(url, false);
    }
  })
  .catch(error => console.error('Error:', error));
}
```

This function sends a POST request to a Flask API to check the URL. It includes the URL in the request body and specifies the content type as JSON.

Alert User

```
// Function to alert the user about the URL status
function alertUser(url, isMalicious) {
  const notificationOptions = {
    type: 'basic',
    iconUrl: 'icon.png',
    title: isMalicious ? 'Malicious URL Detected' : 'Safe URL Detected',
    message: `The URL ${url} is detected as ${isMalicious ? 'malicious' : 'safe'}.`,
    buttons: [{ title: 'Ignore' }],
    priority: 0
  };

  chrome.notifications.create(notificationOptions);
}
```

This function creates a notification to alert the user about the safety status of the URL. It sets different titles and messages based on whether the URL is detected as malicious or safe.

The background script for the "Frankie" Chrome extension is well-designed and effectively accomplishes its purpose of detecting malicious URLs. The script leverages Chrome's API to monitor tab updates, communicate with a backend machine learning API, and notify users of the results.

c. Popup Script (`popup.js`)

```
app.py    JS popup.js ×    {} manifest.json    JS background.js M    JS content.js M
Extension > JS popup.js > document.addEventListener('DOMContentLoaded') callback > showPopup
You, 2 days ago | 1 author (You)
document.addEventListener('DOMContentLoaded', () => {
    const checkUrlButton = document.getElementById('checkUrl');
    const popup = document.getElementById('popup');
    const overlay = document.getElementById('overlay');
    const popupClose = document.getElementById('popupClose');
    const predictionText = document.getElementById('predictionText');

    checkUrlButton.addEventListener('click', () => {
        chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
            const currentTab = tabs[0];
            if (currentTab && currentTab.url) {
                checkUrl(currentTab.url);
            }
        });
    });

    popupClose.addEventListener('click', () => {
        closePopup();
    });

    overlay.addEventListener('click', () => {
        closePopup();
    });

    function checkUrl(url) {
        fetch('http://localhost:7000/api/predict', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ url: url })
        })
        .then(response => response.json())
        .then(data => {
            if (data.result === 'malicious') {
                showPopup(`The URL ${url} is malicious!`);
            } else {
                showPopup(`The URL ${url} is safe.`);
            }
        })
        .catch(error => console.error('Error:', error));
    }
```

Provides an interface for manual URL checks.

The script includes event listeners for user actions, functions to interact with the backend API, and mechanisms to display prediction results in a popup.

Detailed Analysis

Document Ready Event Listener

```
> JS popup.js > document.addEventListener('DOMContentLoaded') callback
ou, 7 days ago | 1 author (You)
ocument.addEventListener('DOMContentLoaded', () => {
   const checkUrlButton = document.getElementById('checkUrl');
   const popup = document.getElementById('popup');
   const overlay = document.getElementById('overlay');
   const popupClose = document.getElementById('popupClose');
   const predictionText = document.getElementById('predictionText');
```

DOMContentLoaded Event Listener

- Ensures that the DOM is fully loaded before executing the script.
- Initializes variables to reference various DOM elements (`checkUrlButton`, `popup`, `overlay`, `popupClose`, and `predictionText`).

```javascript
checkUrlButton.addEventListener('click', () => {
    chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
        const currentTab = tabs[0];
        if (currentTab && currentTab.url) {
            checkUrl(currentTab.url);
        }
    });
});
```

Event Listeners for User Interactions:

checkUrlButton: Adds a click event listener to the "Check URL" button. When clicked, it queries the active tab in the current window and invokes the `checkUrl` function with the tab's URL.

popupClose: Adds a click event listener to the popup close button to close the popup when clicked.

overlay: Adds a click event listener to the overlay to close the popup when the overlay is clicked.

```javascript
popupClose.addEventListener('click', () => {
    closePopup();
});
    You, 7 days ago • code update
overlay.addEventListener('click', () => {
    closePopup();
});

function checkUrl(url) {
    fetch('http://localhost:7000/api/predict', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ url: url })
    })
    .then(response => response.json())
    .then(data => {
        if (data.result === 'malicious') {
            showPopup(`The URL ${url} is malicious!`);
        } else {
            showPopup(`The URL ${url} is safe.`);
```

checkUrl Function: Sends a POST request to a Flask API endpoint (`http://localhost:7000/api/predict`) with the URL to be checked.
Parses the JSON response from the API to determine if the URL is 'malicious' or 'safe'. Invokes the `showPopup` function with an appropriate message based on the API response.

```javascript
    function showPopup(message) {
        predictionText.textContent = message;
        popup.style.display = 'block';
        overlay.style.display = 'block';
    }

    function closePopup() {
        popup.style.display = 'none';
        overlay.style.display = 'none';
    }
});
```

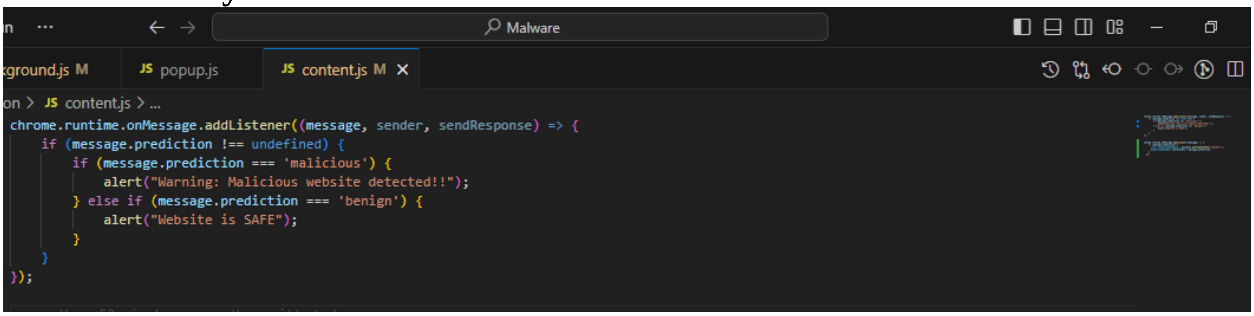
showPopup Function: Sets the `textContent` of the `predictionText` element to the given message. Displays the popup and overlay by changing their CSS `display` properties to 'block'.

closePopup Function:Hides the popup and overlay by setting their CSS `display` properties to 'none'.
The script for the "Frankie" Chrome extension provides a robust and user-friendly mechanism for detecting malicious URLs. It integrates well with the extension's UI and backend API, offering clear and informative feedback to users. The recommendations provided aim to further enhance the script's functionality, error handling, and performance.

Content.js

The listeners are designed to process predictions from a machine learning model that classifies URLs as either 'malicious' or 'benign' and to update the user interface accordingly.

Detailed Analysis



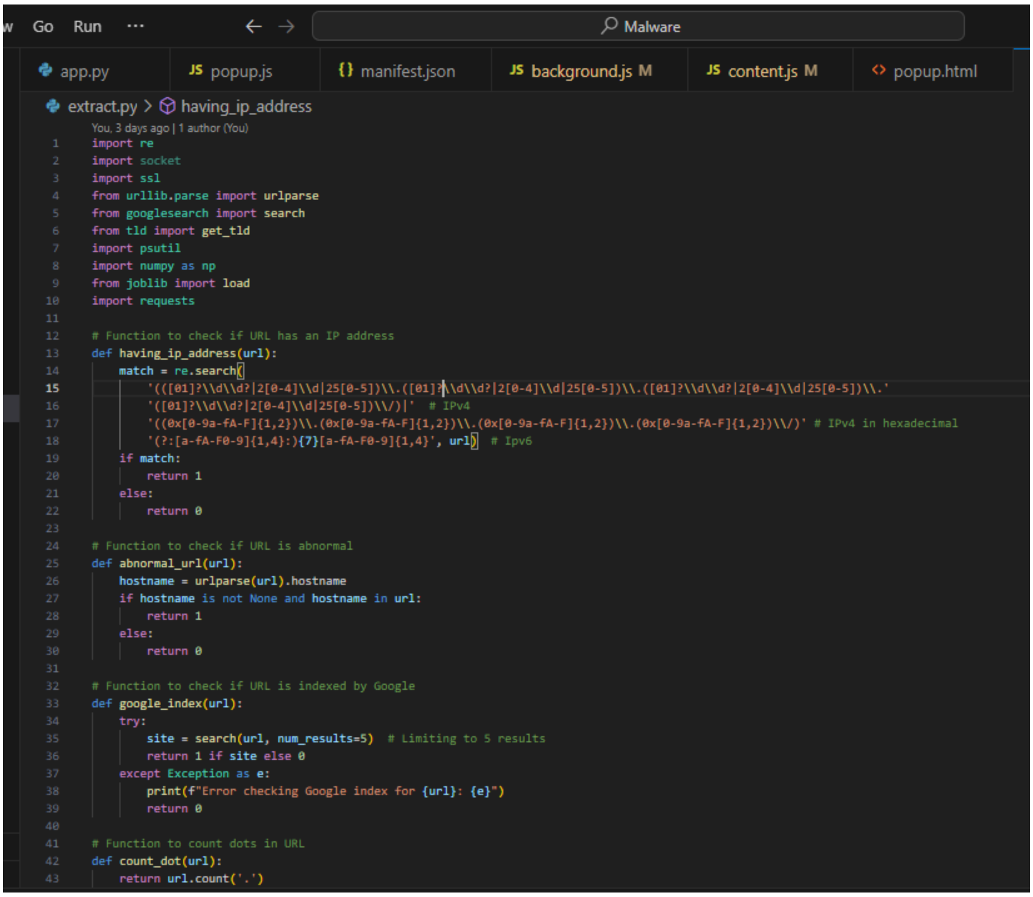
Message Listener for Alerts
This listener responds to messages containing a `prediction` property. Based on the prediction value, it displays an alert to the user indicating whether the website is 'malicious' or 'benign'.

Message Listener for Updating UI Element

This listener updates the text content of a UI element with the ID 'checkUrl' to display the prediction result.
The message listeners in the code are designed to handle predictions from a machine learning model, providing alerts and updating the UI.

3.2Backend Flask API

A. URL Feature Extraction



```python
import re
import socket
import ssl
from urllib.parse import urlparse
from googlesearch import search
from tld import get_tld
import psutil
import numpy as np
from joblib import load
import requests

# Function to check if URL has an IP address
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        return 1
    else:
        return 0

# Function to check if URL is abnormal
def abnormal_url(url):
    hostname = urlparse(url).hostname
    if hostname is not None and hostname in url:
        return 1
    else:
        return 0

# Function to check if URL is indexed by Google
def google_index(url):
    try:
        site = search(url, num_results=5)  # Limiting to 5 results
        return 1 if site else 0
    except Exception as e:
        print(f"Error checking Google index for {url}: {e}")
        return 0

# Function to count dots in URL
def count_dot(url):
    return url.count('.')
```

A set of functions to extract various features from the URL for analysis. These include checks for IP addresses, suspicious words, URL length, etc. This code was obtained from this.

GitHub repo: https://github.com/philomathic-guy/Malicious-Web-Content-Detection-Using-Machine-Learning/blob/master/features_extraction.py

B. Prediction Function

```python
def main(url):
        features.append(check_ssl_certificate(url))
        features.append(check_xss_injection(url))
        return features

# Function to load model and predict if URL is malicious or benign
def get_prediction_from_url(url):
    try:
        features = main(url)
        features = np.array(features).reshape(1, -1)

        try:
            loaded_model = load('rf_model.joblib')  # Ensure path is correct
        except FileNotFoundError:
            raise RuntimeError("Model file not found. Ensure 'rf_model.joblib' is in the correct path.")

        prediction = loaded_model.predict(features)
        return "malicious" if int(prediction[0]) == 1 else "benign"
    except Exception as e:
        raise RuntimeError(f"Error predicting URL: {e}")
```

Loads the pre-trained model and makes a prediction.

```python
# Function to load model and predict if URL is malicious or benign
def get_prediction_from_url(url):
    try:
        features = main(url)
        features = np.array(features).reshape(1, -1)

        try:
            loaded_model = load('rf_model.joblib')  # Ensure path is correct
        except FileNotFoundError:
            raise RuntimeError("Model file not found. Ensure 'rf_model.joblib' is in the correct path.")

        prediction = loaded_model.predict(features)
        return "malicious" if int(prediction[0]) == 1 else "benign"
    except Exception as e:
        raise RuntimeError(f"Error predicting URL: {e}")
```

The `*get_prediction_from_url*` function is designed to predict whether a given URL is malicious or benign. This is achieved by extracting features from the

URL, reshaping these features into the appropriate format, loading a pre-trained machine learning model, and using this model to make a prediction.

def get_prediction_from_url(url):
Input: A single URL as a string.
Output: A string indicating whether the URL is "malicious" or "benign".

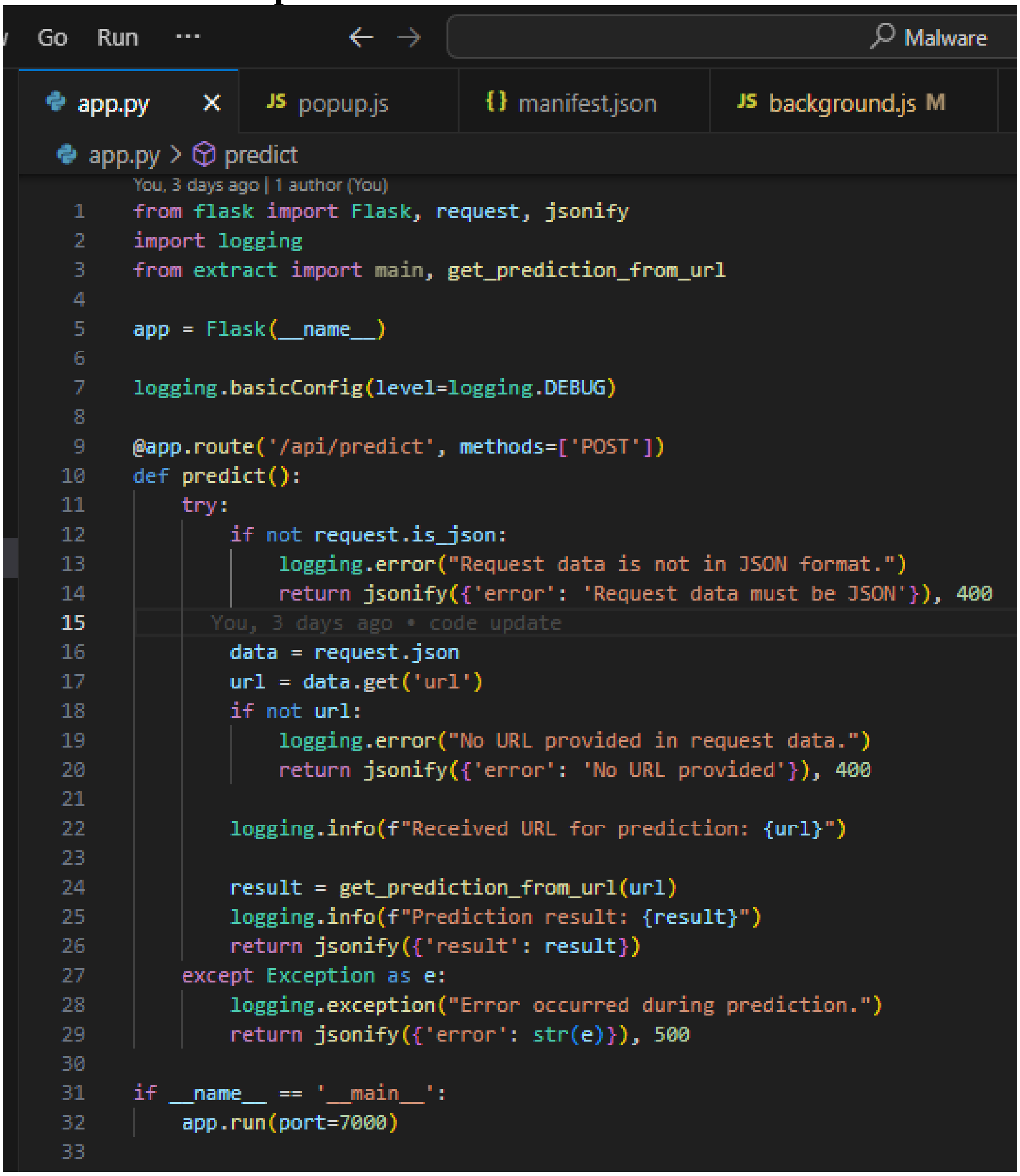
Feature Extraction
Extracts features from the given URL using the `main` function.
`features = main(url)` calls the `main` function, which returns a list of features extracted from the URL.
`features = np.array(features).reshape(1, -1)` converts the list of features into a NumPy array and reshapes it into a 2D array with one row. This is necessary because the model expects the input in this format.

Model Loading
This loads the pre-trained machine-learning model from a file.
loaded_model = load('rf_model.joblib')` attempts to load the model file a`*rf_model.joblib*`. If the model file is not found, a `FileNotFoundError` is caught, and a `RuntimeError` is raised with a clear message.

Prediction
Uses the loaded model to predict whether the URL is malicious or benign.
prediction = loaded_model.predict(features)` makes a prediction using the model.
The function checks the prediction value. If the value is `1`, it returns "malicious"; otherwise, it returns *"benign"*.

Error Handling
Catches any exceptions that occur during the feature extraction, model loading, or prediction steps and raises a *`RuntimeError`* with a descriptive message. This ensures that any issues encountered during the execution of the function are reported clearly.
The *`get_prediction_from_url`* function is a critical part of the URL classification pipeline. It effectively integrates feature extraction with a machine learning model to classify URLs as malicious or benign. The function handles errors gracefully, providing clear messages when something goes wrong.

C. Flask API Endpoint



```python
from flask import Flask, request, jsonify
import logging
from extract import main, get_prediction_from_url

app = Flask(__name__)

logging.basicConfig(level=logging.DEBUG)

@app.route('/api/predict', methods=['POST'])
def predict():
    try:
        if not request.is_json:
            logging.error("Request data is not in JSON format.")
            return jsonify({'error': 'Request data must be JSON'}), 400

        data = request.json
        url = data.get('url')
        if not url:
            logging.error("No URL provided in request data.")
            return jsonify({'error': 'No URL provided'}), 400

        logging.info(f"Received URL for prediction: {url}")

        result = get_prediction_from_url(url)
        logging.info(f"Prediction result: {result}")
        return jsonify({'result': result})
    except Exception as e:
        logging.exception("Error occurred during prediction.")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(port=7000)
```

Handles POST requests and returns the prediction.

Overview

The given code sets up a Flask web server that provides an endpoint for predicting whether a given URL is malicious or benign. The server exposes a single API endpoint (`/api/predict`) which accepts POST requests with a JSON payload containing the URL to be checked. It uses logging to track requests and errors and returns JSON responses to the client.

1. Imports

```python
from flask import Flask, request, jsonify
import logging
from extract import main, get_prediction_from_url
```

Flask: The web framework used to create the server.
request, jsonify: Flask utilities for handling requests and responses.
logging: Python's built-in logging module for tracking events and errors.
main, get_prediction_from_url: Functions imported from the `extract` module for feature extraction and URL prediction.

2. Flask Application Initialization

```python
app = Flask(__name__)

logging.basicConfig(level=logging.DEBUG)

@app.route('/api/predict', methods=['POST'])
def predict():
```

Initializes a Flask application instance.

3. Logging Configuration

Configures logging to display messages of level DEBUG and above.

```python
@app.route('/api/predict', methods=['POST'])
def predict():
    try:
        if not request.is_json:
            logging.error("Request data is not in JSON format.")
            return jsonify({'error': 'Request data must be JSON'}), 400

        data = request.json
        url = data.get('url')
        if not url:
            logging.error("No URL provided in request data.")
            return jsonify({'error': 'No URL provided'}), 400

        logging.info(f"Received URL for prediction: {url}")

        result = get_prediction_from_url(url)
        logging.info(f"Prediction result: {result}")
        return jsonify({'result': result})
    except Exception as e:
        logging.exception("Error occurred during prediction.")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(port=7000)
```

4. API Endpoint Definition

Defines a POST endpoint at `/api/predict`.

5. Request Handling

JSON Check: Ensures that the request content type is JSON. Logs an error and returns a 400 response if the check fails.
URL Extraction: Extracts the URL from the JSON data. Logs an error and returns a 400 response if the URL is missing.
Logging and Prediction
Logs the received URL.

Calls `get_prediction_from_url` to predict whether the URL is malicious or benign.
Logs the prediction result.
Returns the result in a JSON response.

Error Handling

- Catches any exceptions that occur during the prediction process.
- Logs the exception with a traceback.
- Returns a 500 response with the error message.

Server Execution

- Runs the Flask server on port 7000 if the script is executed directly.

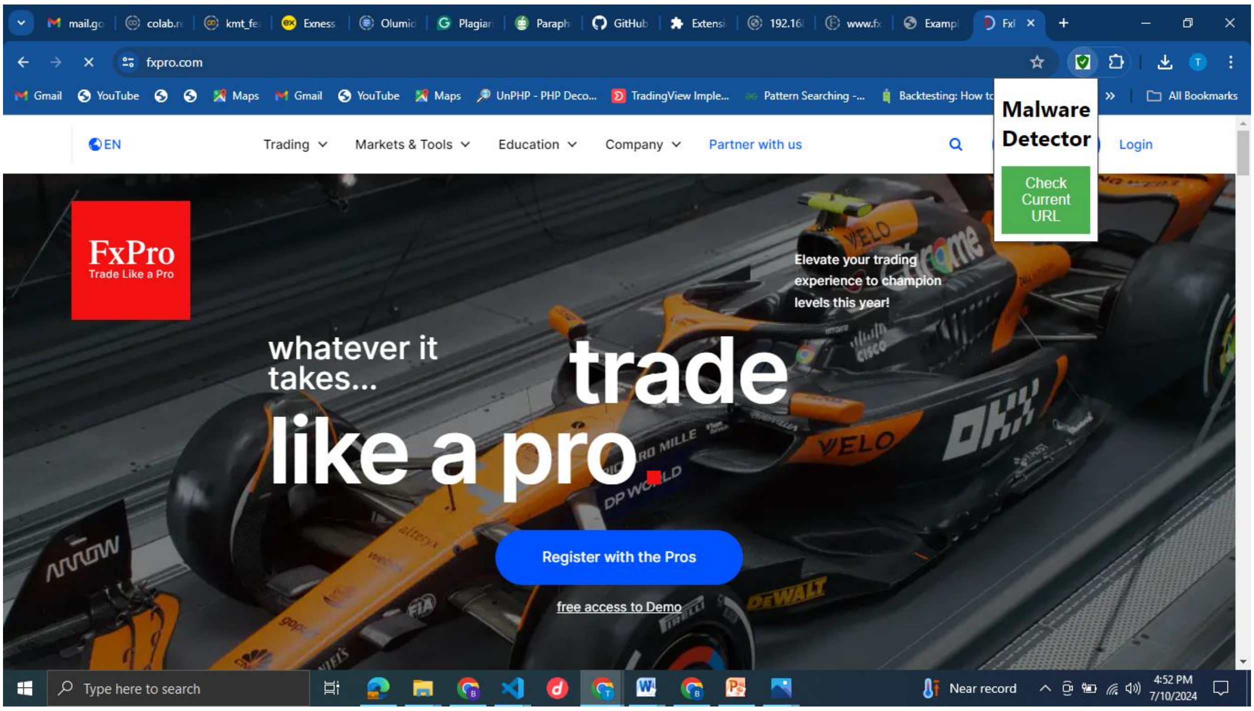
This Flask application provides a simple and robust interface for predicting whether URLs are malicious or benign. Key features include:

- JSON Validation: Ensures the request data is in JSON format and contains a URL.
- Logging: Comprehensive logging for debugging and monitoring.
- Error Handling: Graceful handling of errors with informative responses

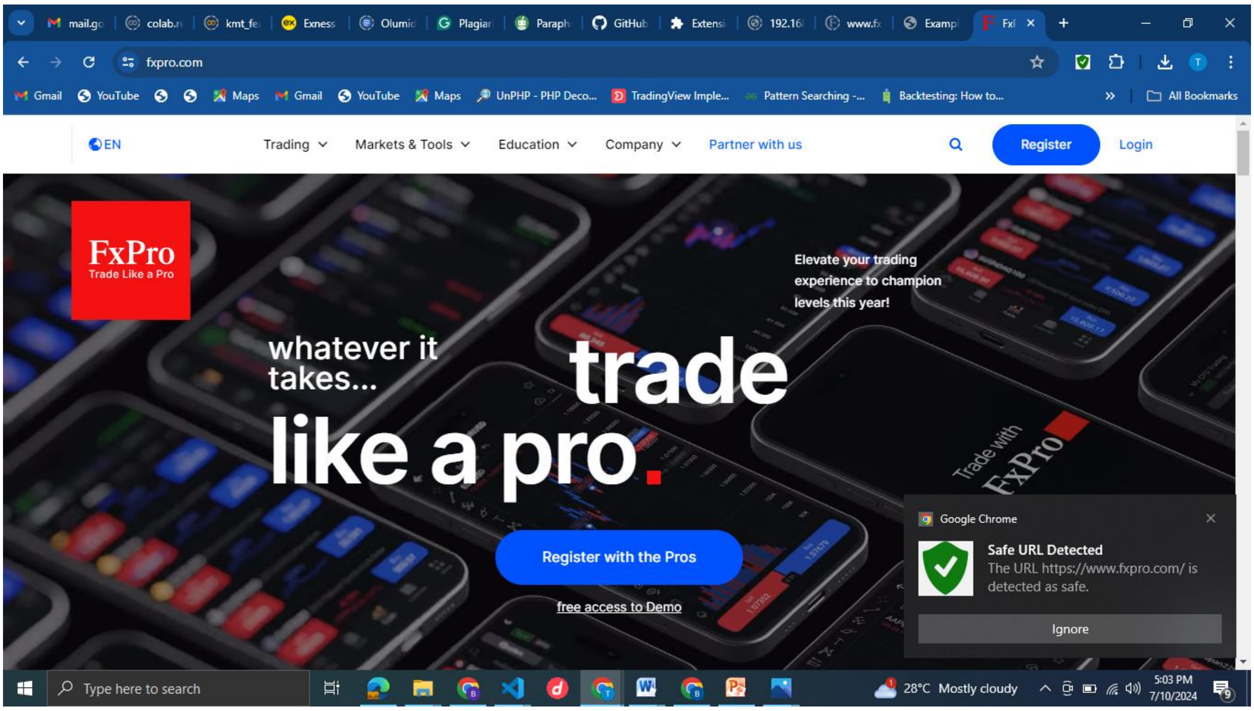
4. Testing and Validation

a. Local Testing

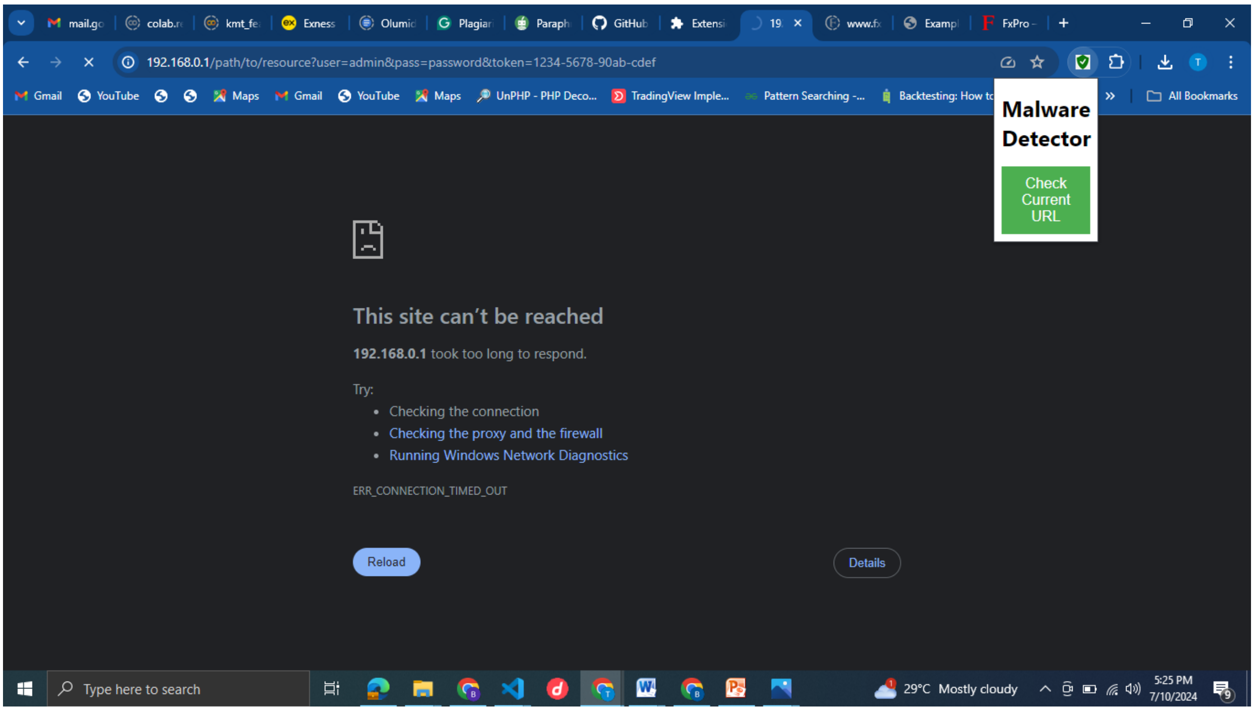
- **Extension**: Tested the Chrome extension locally to ensure URL checks and notifications function as expected.
- **Backend:** Verified the Flask API responds accurately to URL predictions.



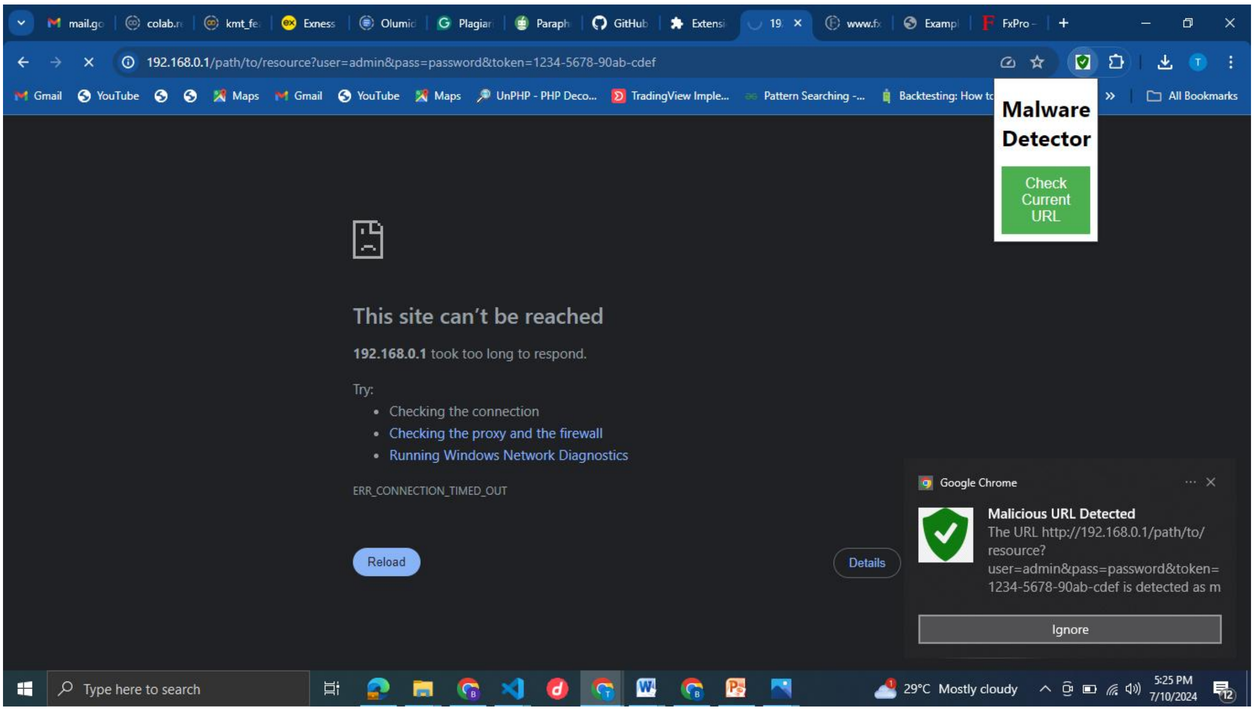
Testing of a safe URL



Result of the testing with a pop up notification.



Testing of a malicious URL



Result of the testing with a pop up notification

b. Integration Testing

- **End-to-End Testing**: Simulated real user scenarios by navigating to various URLs and observing the extension's response.
- **Error Handling**: Checked for graceful handling of API errors and invalid URLs.

These were some of the malicious sites tested –

http://example-login-secure.com/banking/login.php?session=1234567890

http://malicious-site.example.com/login.php?username=admin&password=admin123
http://malicioussite.example.com/login.php?username=admin&password=admin123
http://192.168.0.1/fakebank/login.php?user=admin&password=admin123

http://example.com/this/is/a/very/long/path/that/looks/suspicious/and/may/be/malicious?user=1&id=100&session=xyz

http://example-login-secure.com/banking/login.php?session=1234567890

http://secure-update.example.com/verifyaccount/login.php?user=admin&token=abcdef123456

http://account-security.example.com/update-info/login.php?email=user@example.com&session=xyz123

http://secure-login.example.net/update-info.php?user=admin&auth=abcdef123456

http://account-verification.example.com/secure-login.php?user=admin&token=xyz987654

http://malicious-site.example.com/login.php?username=admin&password=admin123

http://malicious-site.example.com/login.php?username=admin&password=admin123

http://example.com/this/is/a/very/long/path/that/looks/suspicious/and/may/be/malicious?user=1&id=100&session=xyz

http://example-login-secure.com/banking/login.php?session=1234567890

http://pub-0fac81924c9e47b7901a9cc6d41b136a.r2.dev/megproctect.html http://pub-ba8507aed7c44524b1e60764505db63c.r2.dev/index3.htm