AI User Safety Application

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Mirsaeid Abolghasemi
Varun Bhaseen
Gulnara Timokhina

May 2021

APPROVED

Vijay Eranti, Project Advisor

Dan Harkey, Director, MS Software Engineering

Rod Fatoohi, Department Chair

ABSTRACT

AI User Safety Application

By

Mirsaeid Abolghasemi, Varun Bhaseen, Gulnara Timokhina

A user's privacy protection is of prime importance nowadays. The majority of user privacy violations happen through phishing attacks. As per the reports published by the "Anti-phishing working group for Q4 2020", it has been observed that the number of phishing sites detected was 637,302 of which 84% now use Secure Sockets Layer (SSL) protection. A phishing attack can be detected from paid or freeware anti-virus software, corporate email phishing detection, and user intelligence or awareness.

The primary goal of a phishing attack is to exploit human weaknesses. The challenge with current phishing detection is the lack of availability of a reliable state-of-art tool that could compensate for these weaknesses. The majority of phishing detection technology is based on a classical approach where the agent (detector) relies on information on which it has been trained. The agent does not factor into a real-time artificial intelligence-based detection that can detect the most recently evolved techniques on which phishing attacks are based. Also, most commercial applications are highly resourced intensive on memory footprint and CPU usage. There are no quick, small, and reliable solutions in the market.

In this project, we are proposing an artificial intelligence-based real-time detector that can protect user's privacy details by constantly scanning a web page for malicious scripts, phishing contents, domain authenticity, and logo identifiers. Our approach uses not only natural language processing but also computer vision to detect the authenticity of the web page and the use of logos or images on that web page. The outcome is a plugin on a browser that can consume minimal resources and can give a quick scan notification about the safety of the web page.

Acknowledgments

The authors are deeply indebted to Professor Vijay Eranti for his invaluable comments and assistance in the preparation of this study.

Table of Contents

List of Figures

Chapter 1. Project Overview

1.1. Introduction

Phishing is a security vulnerability that aims to trick unsuspecting users by mixing social engineering and website spoofing techniques into stealing their sensitive details (e.g., password, bank, or financial details). A typical phishing attack's lifecycle begins with the receipt of a fake e-mail, SMS, or instant message from scammers trying to make users think and believe that it comes from a legitimate source. The messages typically use persuasive claims and a link pointing to a fake web page that mimics the legitimate web page of the target brand. If the user enters their credentials, the life cycle of the attack will conclude by submitting confidential information to phishers which can be misused for online fraud or the misuse of personal data.

Phishing is one of the oldest and well-known security vulnerability exploitation techniques which relies more on human weaknesses rather than a technological weakness as stated by Bozkir et al. [4] and because of the human factor involved it becomes difficult to eliminate it. Although there are many ways in which a phishing website can be detected the traditional approach has been to rely on databases that maintain a list of these websites and can prevent them from opening at user's browser but as mentioned by Abdelnabi et al. [5] any zero-day detections are not possible in these traditional approaches. The update of the database takes a considerable amount of time and usually, a certain number of users have already been affected.

Hence to compensate for human weakness Huang et al [1] in their paper have suggested a more artificial intelligence or deep learning-based technique in which the model can more readily be used to identify the malicious URL for the web page. Based on validation the output of the model will classify whether the web page is phishing or not. Whereas Le et al. [2] suggest a more focused and object detection-based approach where they have enhanced certain features and validate the URL based on a certain character and word frequencies.

Here in this paper, we try to combine various techniques for phishing detection and aim to create an AI agent which can assist users by flagging messages ensuring that users are aware of the current web page status. The AI agent will scan the entire web page to determine if the web page is safe or is intended as phishing.

1.2. Proposed Areas of Study and Academic Contribution

The project scope is bounded on assumption that attackers will try to modify or morph slight changes to existing web pages and use the same to deceit an unsuspecting user into revealing details. The slight changes can be minute to human awareness like changes to Logo, or web page URL. The project hence is focusing on three major aspects with first being validating the authenticity of a logo followed by validating the URL address and finally trying to predict if the web page has a malicious (phishing) intent.

The project is combining all three checking or validation criteria into a single ensemble model. The models are based on deep learning techniques and will be ensembled based on multitasking multi-classification techniques. The model will eventually be saved and used in a web browser extension as a knowledge model or AI agent model which will compensate and enable users to be safe from phishing attacks.

Following are the proposed areas of study for this project from an academic perspective:

- Use pipelines to create a production-level application model.
- Combining object detection techniques with NLP (Natural Language Processing) techniques.
- Deploying an ensemble model on a web browser application framework optimized for performance and speed without compromising on CPU/GPU usage and memory footprint at runtime.

1.3. Current State of the Art

Phishing as stated earlier and with reports from APWG (Anti Phishing Work Group) is a multi-million-dollar industry and hence many traditional techniques are available to use commercially. Here we have listed out few techniques in a similar context to our project which is based on ML/DL algorithms, and we have inspired few techniques from each of them.

One of the novel techniques we came across was in the article "LogoSENSE" [4]. In this paper, the author highlights what are the different underlying features that are present in the dataset and how to carry out feature extraction. Also, it explains techniques like HOG (Histogram of Oriented Gradients) and what role it plays in detecting the phishing website by comparing logos and images. This paper uses machine learning techniques rather than deep learning techniques, but the approach to enhance the features in the dataset is useful and can be reused. The authors of this research paper have created their dataset (LogoSENSE) on which they carry out feature engineering and data preprocessing to identify whether a given page is phishing or not by simply analyzing the logo on the page.

The second paper that we have identified is "Phishing URL Detection via CNN and Attention-Based Hierarchical RNN" [1]. The authors have used CNN and RNN to extract URL texts or characters or words and use them as features to authenticate and verify a website. The authors worked on a dataset that takes a screenshot of a web page and then uses neural networks to extract features and classify the web pages.

The third technique which we identified is URLNet [2] The authors of this paper focus on using lexical features in a URL to identify whether the website is phishing. This is achieved by using a CNN model that can focus on character level and word level embeddings. The authors have built their dataset and carried out feature extraction using position-sensitive bigrams and trigrams. The conclusion for this paper was to build a state-of-the-art model which can detect phishing website by analyzing the URL. We will be using the feature extraction technique and try to combine it with other URL validation techniques listed earlier for the detection of the phishing website.

Then there is Malicious Web Content Detection Using Machine Learning [3]. This paper addresses the imbalanced dataset and what approach should be taken for sampling. The paper also addresses the different dependencies that are there for the creation of applications for data science (Machine learning and Deep learning models).

VisualPhishNet highlights the zero-day phishing website detection approach. What it essentially highlights is that if a phishing website is created today and it is not a part of any phishing website database then how good the model is to detect this fresh new website as a Phishing website [5].

Chapter 2. Project Architecture

2.1. Browser Extension Application Architecture

The figure below provides a high-level architecture for the application. In our application concept, we use two deep learning models based on YOLO multi-class multi-label classification and Char-CNN with binary classification. To get better phishing prediction, these algorithms will be applied to the features extracted from web pages like logo, texts, URL, domain name, and address bar.

The deep learning models will be used in our application to get these extracted features and then it will classify the webpages to show that a webpage is phishing, suspicious, or legitimate. The application will be packaged as a web browser extension and/or a plugin.

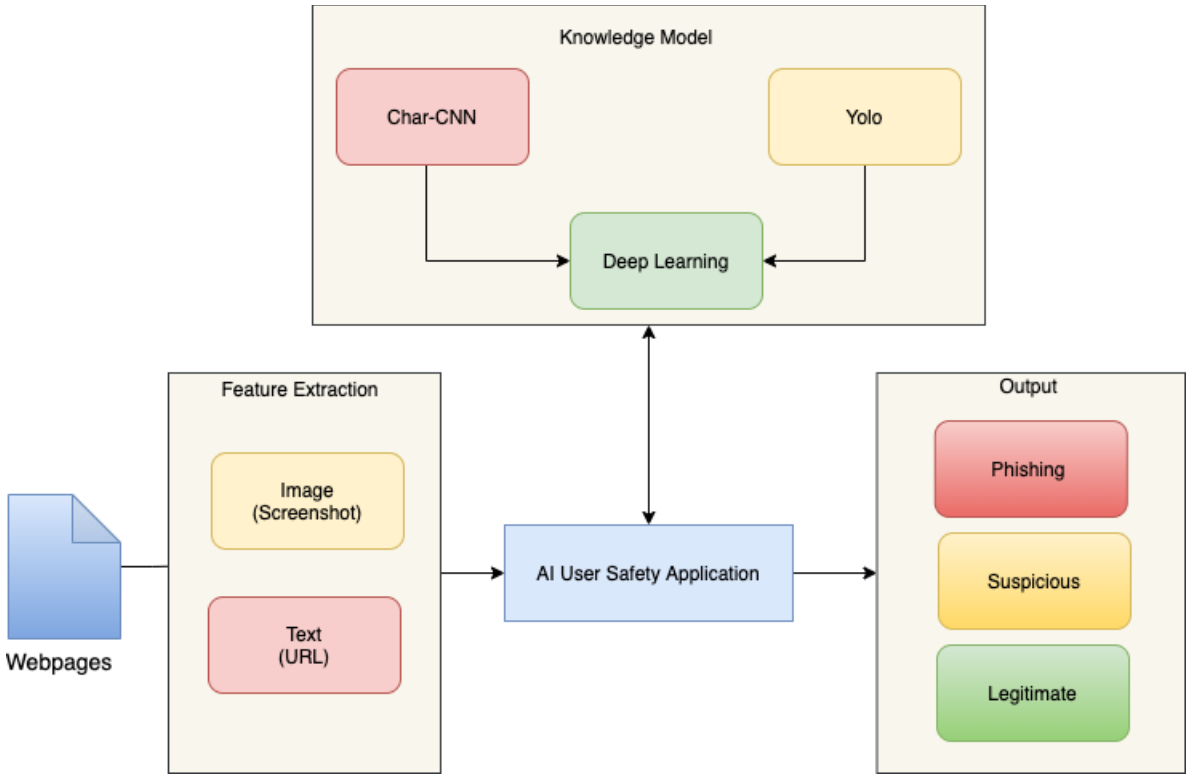


Figure 1: High-level application Architecture

2.2. Model Architecture

It is useful to understand the underlying sub-architecture of the deep learning-based artificial intelligence agent. As seen in figure-2 below we have two different techniques that we will ensemble using a multi-task multi-gate model.

The first technique is YOLO (You Only Look Once), which is an object detection technique that is used to validate a logo present on a web page. YOLO scans a web page, locates the logo as an object. Then it calculates a confidence score. We have set a threshold value (currently 65%) for the confidence score to qualify a logo as a legitimate logo. YOLO has multiple versions, and we are using Version-3 because at the time of design conception the latest was V3.

The second technique is the Char-CNN model (Character Convolutional Neural Network First, the URL of a web page is extracted and then Char-CNN is used to analyze the URL. The URL of a web page is usually morphed by phishers to subdue an unsuspected user and trick them into revealing their information. The model will check to ensure that the URL is valid or if it is a phishing URL and will subsequently raise the flag.

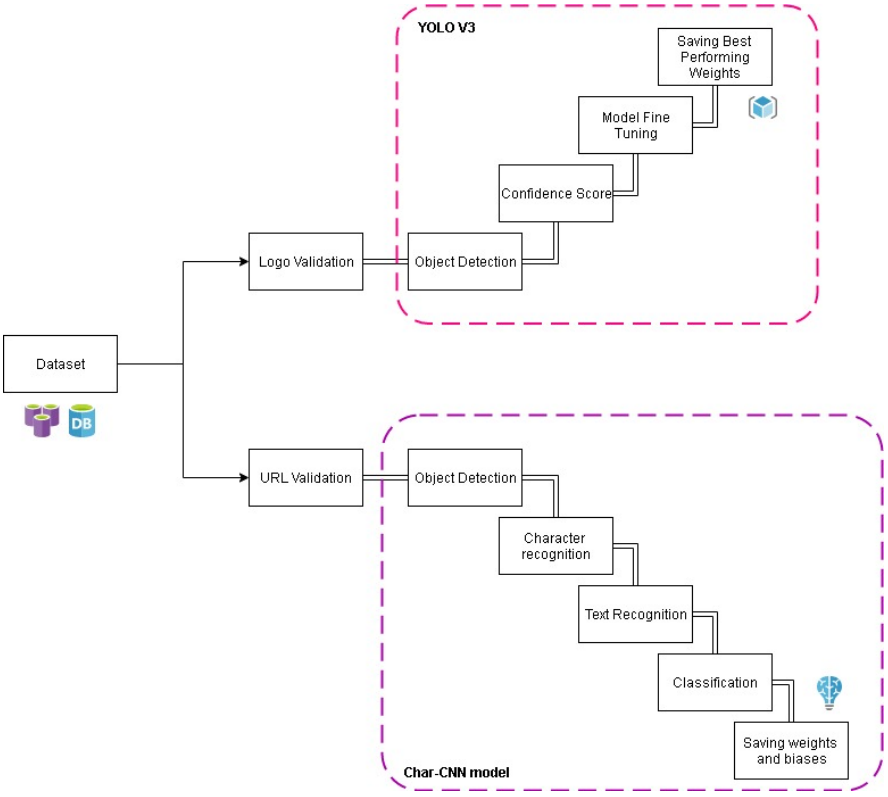


Figure 2: Deep Learning Model Architecture

2.3. Model Prediction and Result

The result is obtained by using a custom logic based on output results from Char-CNN prediction and YOLO model prediction. The detailed output can be seen in Chapter 9. Model Cards.

Chapter 3. Technology Descriptions

3.1. Client-End Plugin

3.1.1. Package Structure

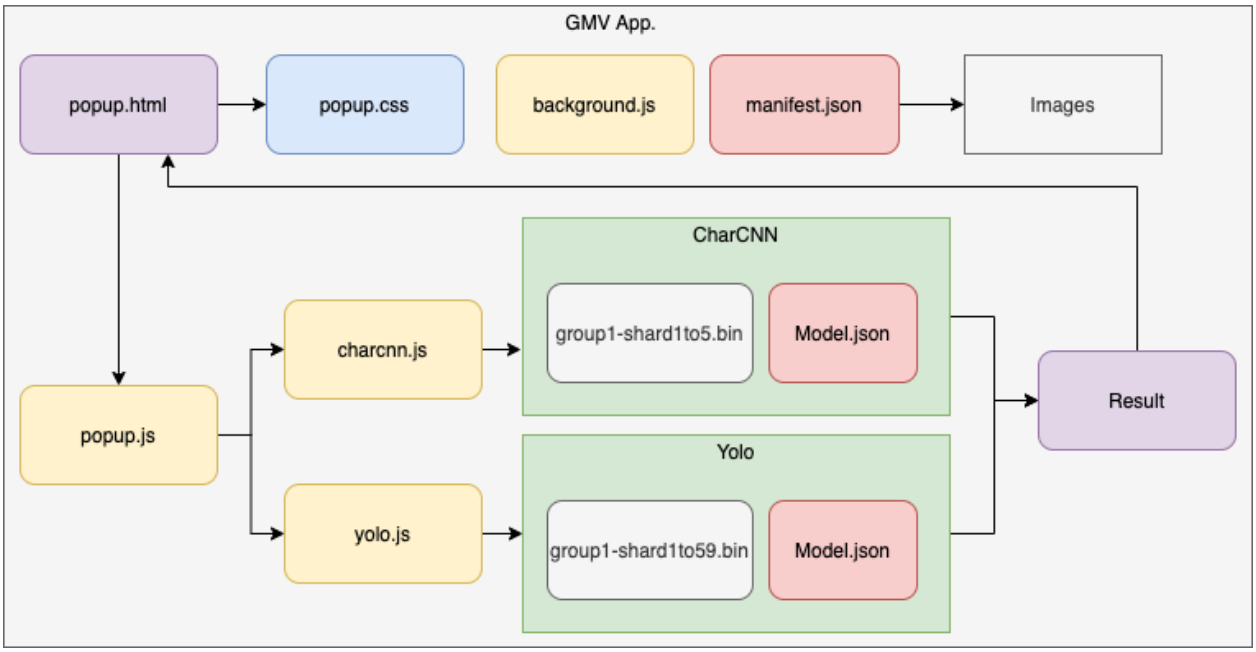


Figure 3: Package Structure of GMV application

We use the Google Chrome extension as our client-end plugin. So, to use the plugin app, first, the GMV AI User Safety Application extension should be installed on Google Chrome. The user should navigate to the "chrome://extensions/" address on Google Chrome. Then the user should click on the "Load unpacked" button to select the GMV extension folder. After that user can pin the GMV extension to check a website is phishing or not.

The app includes:

a) A popup HTML file that calls popup CSS file and popup JavaScript file.

b) The popup JavaScript file will call two JavaScript files charcnn.js and yolo.js.

c) The charcnn.js file will get the webpage URL and will pass to the Char-CNN model. The model will return a result tensor that shows the URL is a phishing URL or not.

d) The yolo.js file will take a screenshot and will send it to the Yolo model. The Yolo model will return a tensor that includes Confidence Scores.

e) Every Chrome extension package has a JSON file, manifest.json, that provides valuable information such as the name of the extension, the version, and other settings.

Background JavaScript can respond to browser events and execute certain actions, which means the background page is loaded or unloaded when it is required. To use this background JavaScript file, it should be declared in the manifest.json file.

3.1.2. Plugin Styling

The plugin app consists of static and template directories which comprises HTML-CSS styling scripts. The scripts are based on color coding (#0055A2, #E5A823, #00FF00). There are three files to create the user interface. The popup HTML file is the main page to run the app and show the result to the user. This HTML file will interact with popup.css and popup.js files.

3.1.3. Plugin Libraries

Following are the key libraries that are being used in the plugin for development purposes:

- TensorFlow JS: We are using TensorFlow JS to import ML-DL models in the plugin and predict the output tensors. We are also using the library to convert the inputs to tensors for the Char-CNN model

3.2. Middleware Hooks

3.2.1. HTTP POST Chrome transmission

We are using Flask and Werkzeug in the backend to handle the HTTP POST routing from the client end. Below is a screenshot for POST routing in the flask:

```
15  from flask import Flask, redirect, url_for, request, render_template
16  from werkzeug.utils import secure_filename
17  from gevent.pywsgi import WSGIServer
18
19  # Define a flask app
20  app = Flask(__name__)
21
22  for f in os.listdir("static\\similar_images\\"):
23      os.remove("static\\similar_images\\"+f)
24
25  print('Model loaded. Check http://127.0.0.1:5000/')
26
27
28  # @app.route('/', methods=['GET'])
29  # def index():
30  #     # Main page
31  #     return render_template('index.html')
32
33
34  # @app.route('/predict', methods=['GET', 'POST'])
35  @app.route('/', methods=['POST'])
36  def predict():
```

Figure 4: Screenshot for the backend Flask routing using HTTP POST

3.2.2. Data Encryption

Data encryption is being handled by a token exchange between the backend server and the client end based on the user session. The data is transferred as base64 encoded strings.

3.3. Backend Server

3.3.1. Cloud Service Provider

The backend server is hosted on AWS EC2.

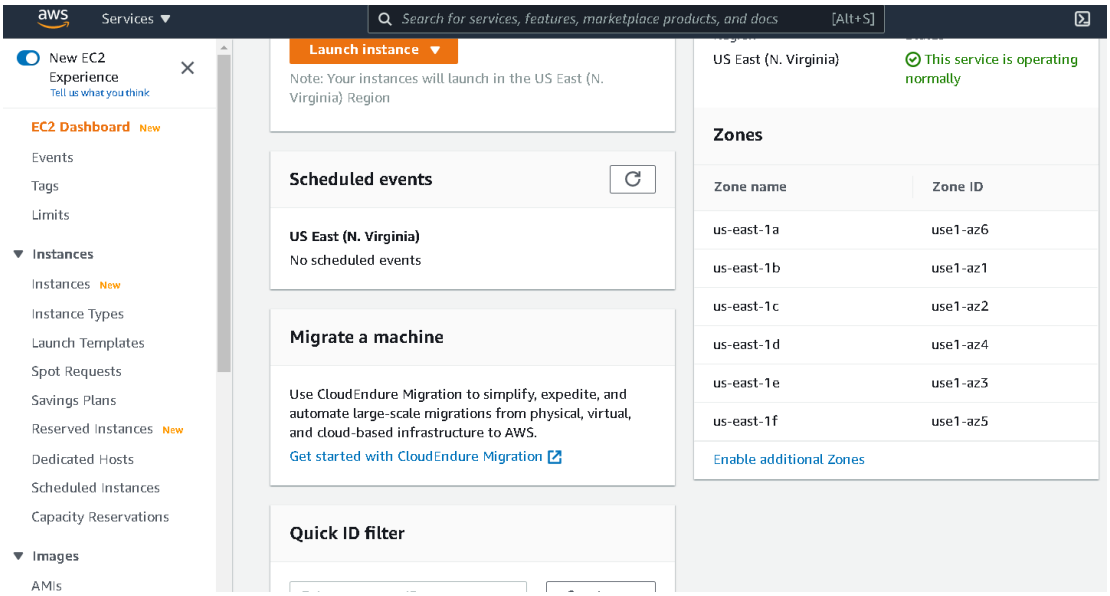


Figure 5: AWS EC2 console and Dashboard for Server Management

3.3.2. Web Server Host (Docker-Nginx-UWSGI)

The Container used for hosting the backend is on the Docker image.

```
[uwsgi]
module = wsgi:app

processes = 1
vacuum = true
die-on-term = true
socket = /tmp/myproject.sock
chmod-socket = 666

#master = true
master = false
```

```
FROM tiangolo/uwsgi-nginx-flask:python3.6

COPY requirements.txt /
COPY kf_upload.conf /etc/nginx/conf.d/
RUN pip install -i https://pypi.tuna.tsinghua.edu.cn/simple -r /requirements.txt
```

Figure 6: UWSGI and DockerFile configuration for the backend application

Chapter 4. Project Design

4.1. Plugin Application Design

4.1.1. Wireframe Design

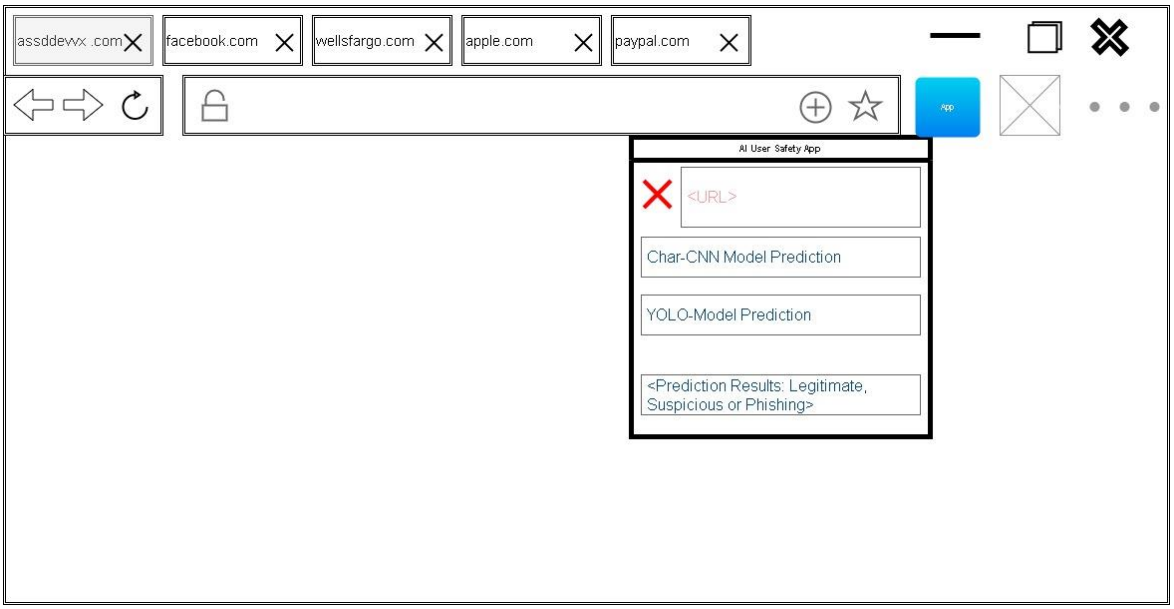


Figure 7: Wireframe for chrome extension plugin

4.1.2. UML Class Diagram

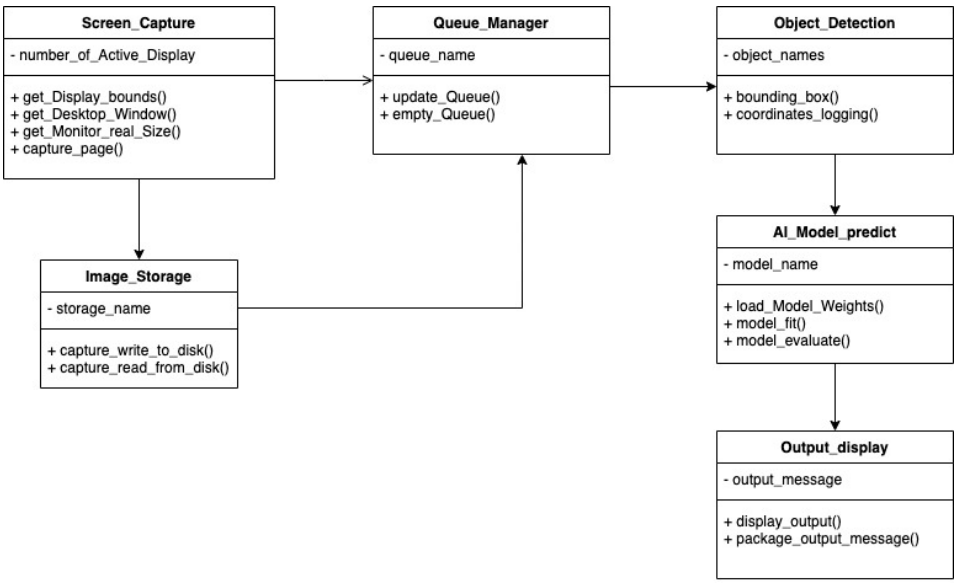


Figure 8: UML Class Diagram for Chrome Plugin Application

4.1.3. UML Sequence Diagram

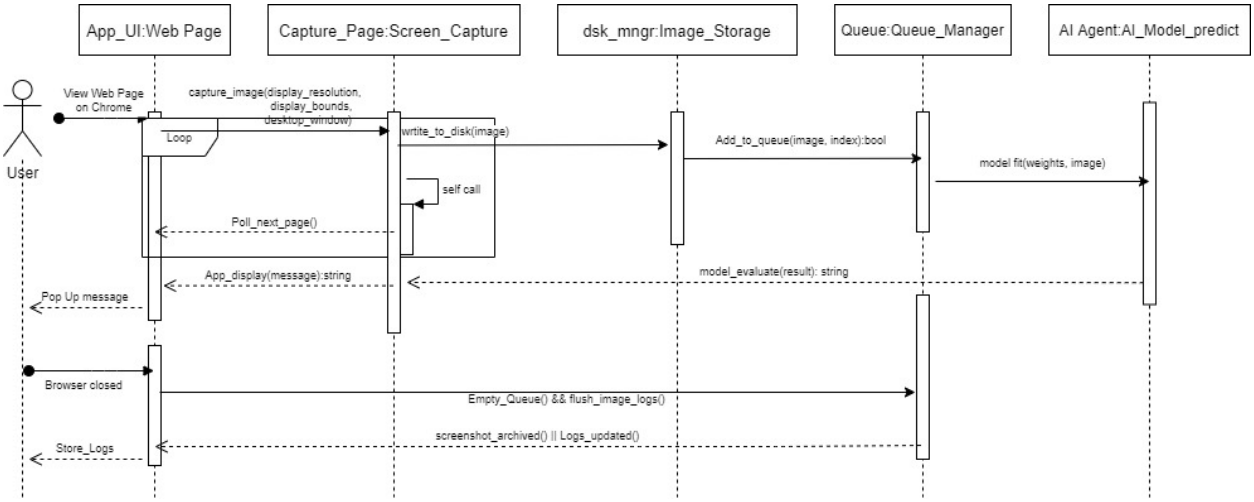


Figure 9: UML Sequence Diagram for Chrome Plugin Application

4.2. YOLO Graph Model Architecture (TensorBoard)

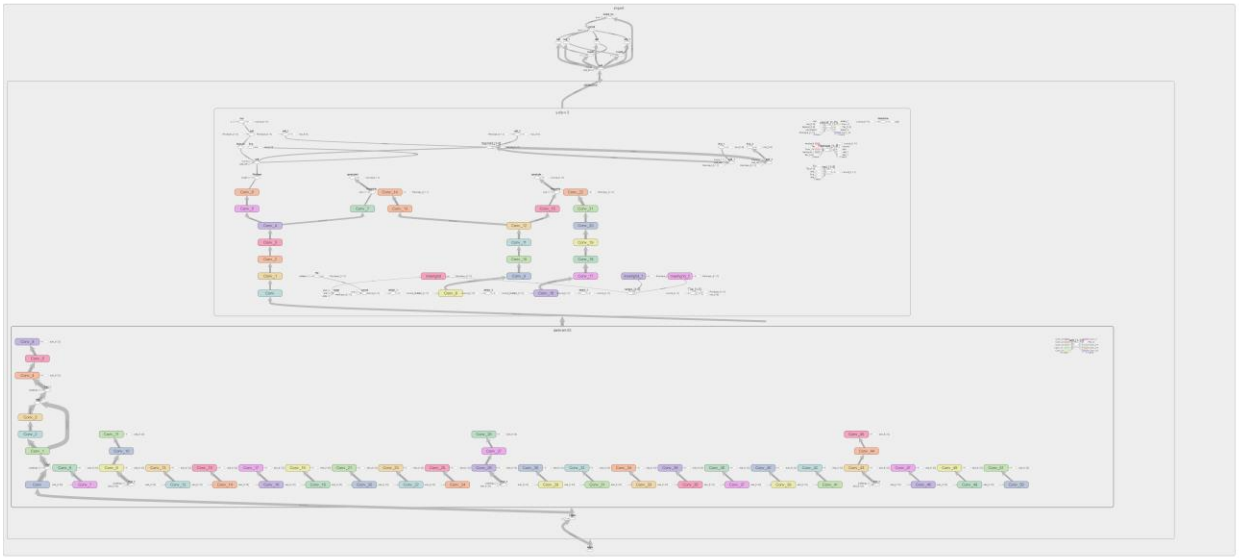


Figure 10: YOLO Graph Model Architecture in TensorBoard

4.3. Backend Prediction UML Class diagram

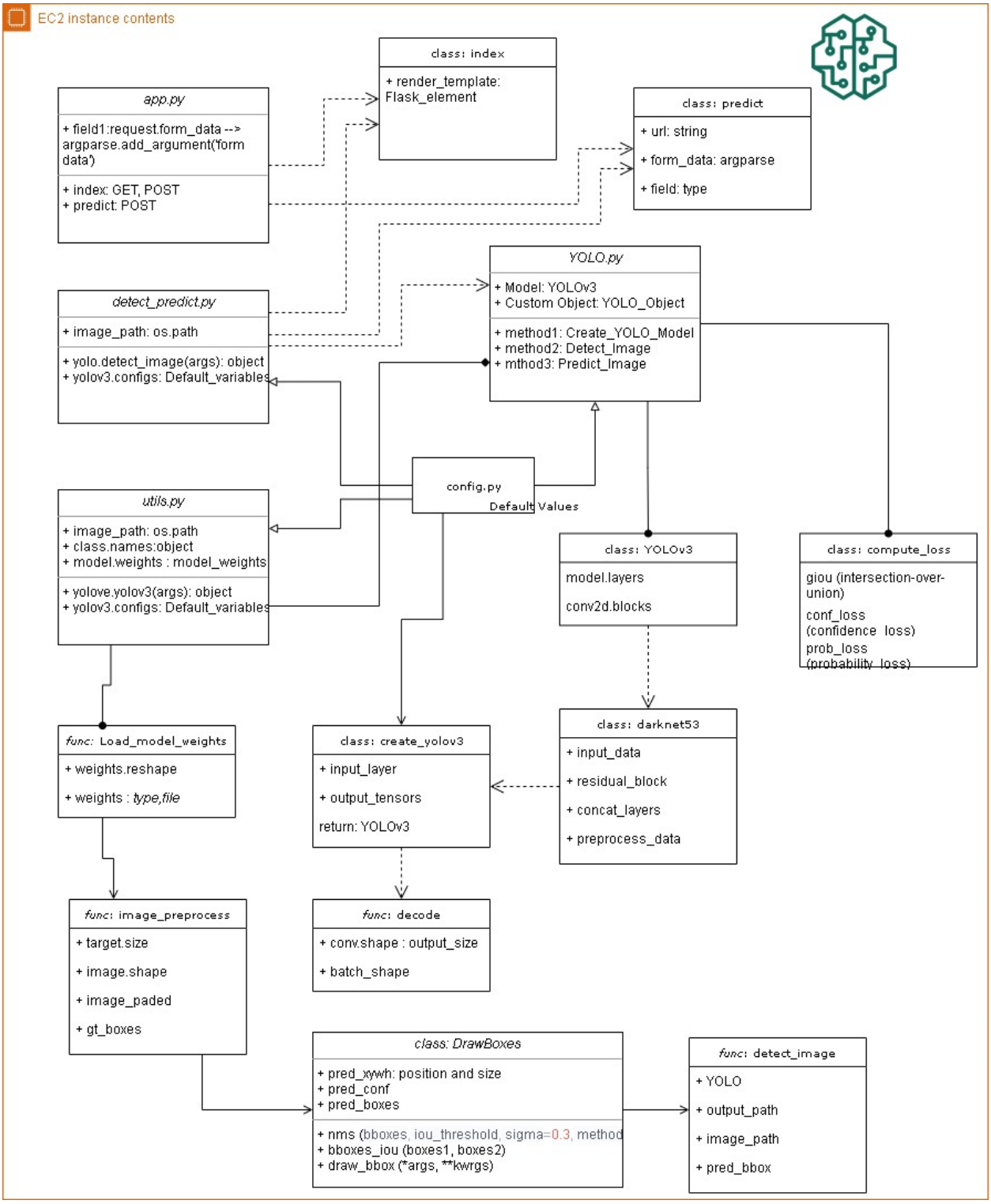


Figure 11: UML Class Diagram for the backend Flask Application on Object Detection using YOLO

Chapter 5. Project Implementation

5.1. Plugin Application Implementation

5.1.1. Key Objective

The primary objective of the project is to create a web browser plugin or extension which can act as a personalized Artificial Intelligent agent to detect a phishing website and protect personal data. Following is the list of objectives that are being achieved from the plugin:

a. Capture a user session and maintain a queue of web pages visited.

b. Validate URL and predict phishing using Char-CNN.

c. Validate the web page.

d. Capture URI's and send a POST to the backend server for object detection and prediction.

e. Compute and display the result.

5.1.2. Market Innovation

Internet is an essential utility in our day-to-day life, we manage our social connections, banking, most financial activities exclusively online. This usage and dependency have opened plenty of ways for our vital private information to be intercepted and misused for malicious activities or theft over the web. One such attack a user can be a victim of is Phishing. Phishers use social engineering and create a mockup of essential websites which can be used to steal our private information.

Here we are trying to have a personalized AI agent which can help protect a user's personal data from being stolen from a phishing website. This is executed by validating the URL for authenticity and validate the logo and get a confidence score on the authenticity of the same. This is followed by content validation to check and identify any tell-tale signs of spamming or phishing.

This is the first time that a solution is being developed which will be combining the results from a YOLO object detection with that of Char-CNN to validate a web page and check for phishing. YOLO is a state-of-art object detection model used to identify objects in a video or an image. We have created a custom dataset and trained both YOLO and Char-CNN models to help us achieve results aligned to our desired objective of phishing detection.

5.1.3. Design Principles and Approach Followed

The design principle and guidelines followed for this project can be seen below:

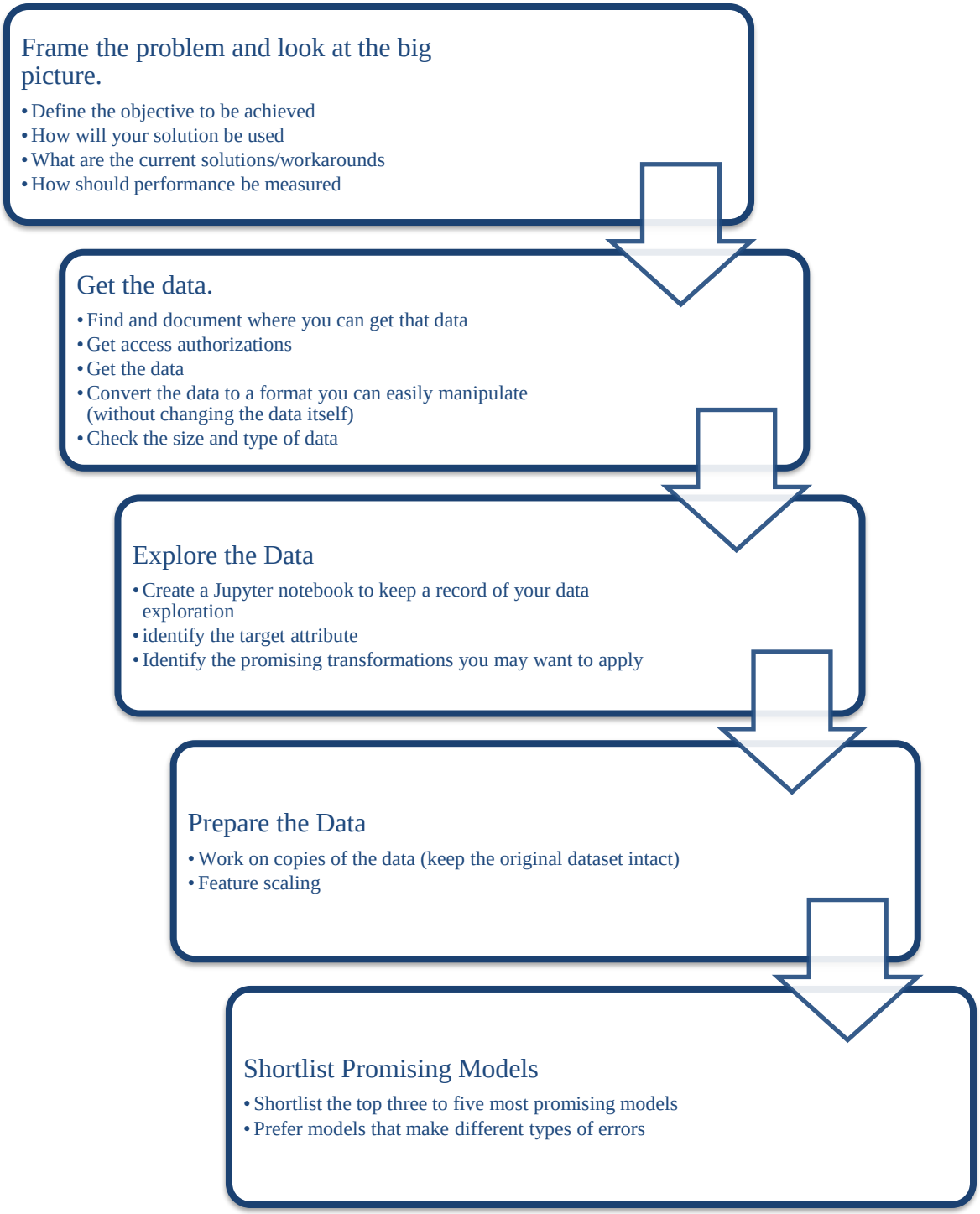


Figure 12: Design Principles and approach for the project design and development. The approach is based on the methodology outlined in the book "Hands-On Machine Learning using TensorFlow, Keras, and Sci-Kit Learn V2."

5.2. Deep Learning Model Implementation

5.2.1. Key Objectives

The primary objective of the AI User Safety Application deep learning models (YOLO and Char-CNN) was to serve as an intelligence engine that can help the plugin to respond quickly to an analysis of a webpage and determine if the evaluated web page is phishing or not.

The objectives targeted to achieve by the backend model are:

a. Process the input received from the client end and determine the confidence score on the objects detected.

b. Ensure the entire data pipeline is secured and encrypted over the web.

c. Return the results to the client to be computed and published.

5.2.2. Ensemble vs Stand-alone vs Cascade modeling

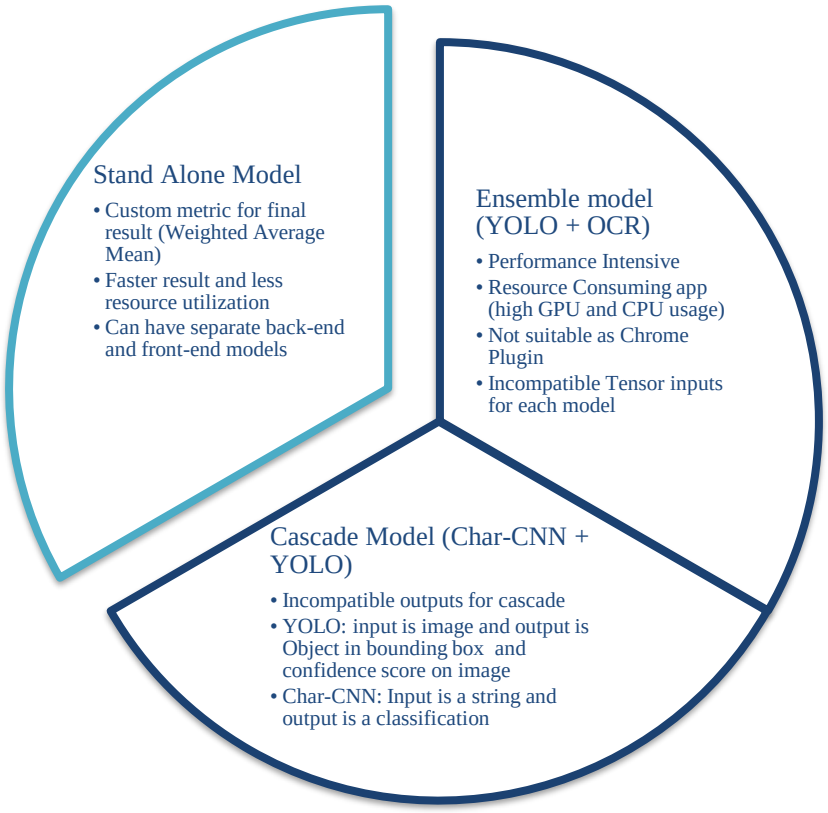


Figure 13: Comparison between Ensemble vs Standalone and Cascade Model.

Chapter 6. Testing and Verification

6.1. AI Model Validation and Testing

6.1.1. YOLO Model Validation and Testing Process

Table 1: YOLO Model Testing Approach

ID	MLDL001
Functionality	Logo Validation
Model	YOLO v3, YOLO v4-Tiny.
Technique	Transfer Learning
Data Format Input	Images
Dataset	Self-Annotated Logo Dataset
Sources	• Archive.org • Google Image Search Crawling
Volume	100 samples per target brand (Total 1000 Images)
Labelling	Hyperlabel
Split Ratio	• Train (Including Validation): 90% • Validation: 20% of Train • Test: 10%
Overfitting Check	• K-Fold Cross Validation (Training Dataset (80%)) • Image Augmentations (Training Dataset (80%))
Underfitting Check	• Number of Epochs in training 11200 • Best performing model to be selected
Exploding gradient and vanishing gradient checks	• Variable Learning rate after every 2000 epochs • Variable steps ranging from warm-up steps 1840 to total steps 93200. • Using Best Value Loss to save the best value as a Checkpoint.
Metrics	• Confidence Score • IOU (intersection over union) • NMS (Non-Maximum Suppression) • Mean Average Precision (MAP) • GIoU • Total Loss

6.1.2. Char-CNN Model Validation and Testing Process

Table 2: Char-CNN Model Testing Approach

ID	MLDL002
Functionality	URL Validation
Model	Char-CNN
Input	Strings
Technique	Transfer Learning or Retraining all layers
Dataset	Prelabelled
Sources	• PhishTank

	• PhishStorm • Majestic
Volume	164,940
Split Ratio	• Train (Including Validation): 80% • Validation: 20% of Train • Test: 20%
Overfitting Check	• Early Stopping • Regularization • Masking
Metrics	• Recall • Precision • F1 Score • Accuracy

6.2. Application Testing

6.2.1. Test Plan

The test plan for the plugin app covers different testing types and various levels to make sure with specific confidence that our application works as expected. These testing types are such as A/B Testing, Hypothesis testing, Vulnerability testing, Pentest, User Acceptance Test. To do testing at various levels requires balancing the time and level of confidence to reach and spend for testing. The test plan also covers three levels of testing like Unit Testing, Integration Testing, and End-to-end Testing. In Unit Testing, all individual units are tested. The module, component, or class will be tested to validate its correctness. In Integration Testing, the sub-systems and modules and their interfaces will be tested. In this case, the way that how various parts of the application interact together will be tested. In end-to-end testing, the application will be tested. For instance, the user interactions with the application and how to show the result should be tested.

In the A/B test, two versions (A and B) of a variable are compared, which are identical except for one variation that might affect a user's behavior. This test is considered the simplest form of controlled experiment. In fact, by adding more variants to the test, it becomes more complex.

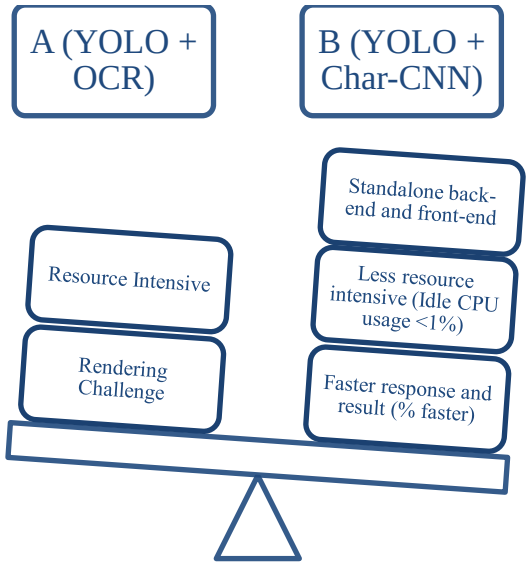


Figure 14: A/B Testing for two variations of prediction models

6.2.2. Unit Tests

By creating unit tests, the valid inputs and outputs will be declared for a given function. Therefore, refactoring the code will be easier because there will be confidence that the code works correctly when all the tests pass. Below is the test report conducted using the python tool nose tests:

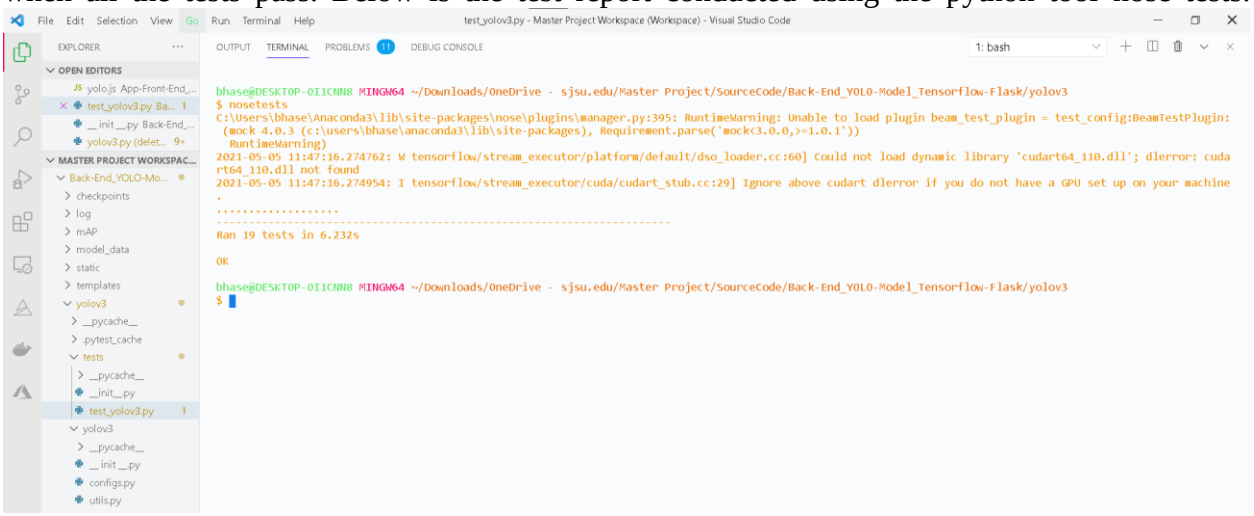


Figure 15: Screenshot of a sample of unit test cases from the backend server

Chapter 7. Performance and Benchmarks

7.1. Performance specs

Following are the specs that have been used for the performance evaluation:

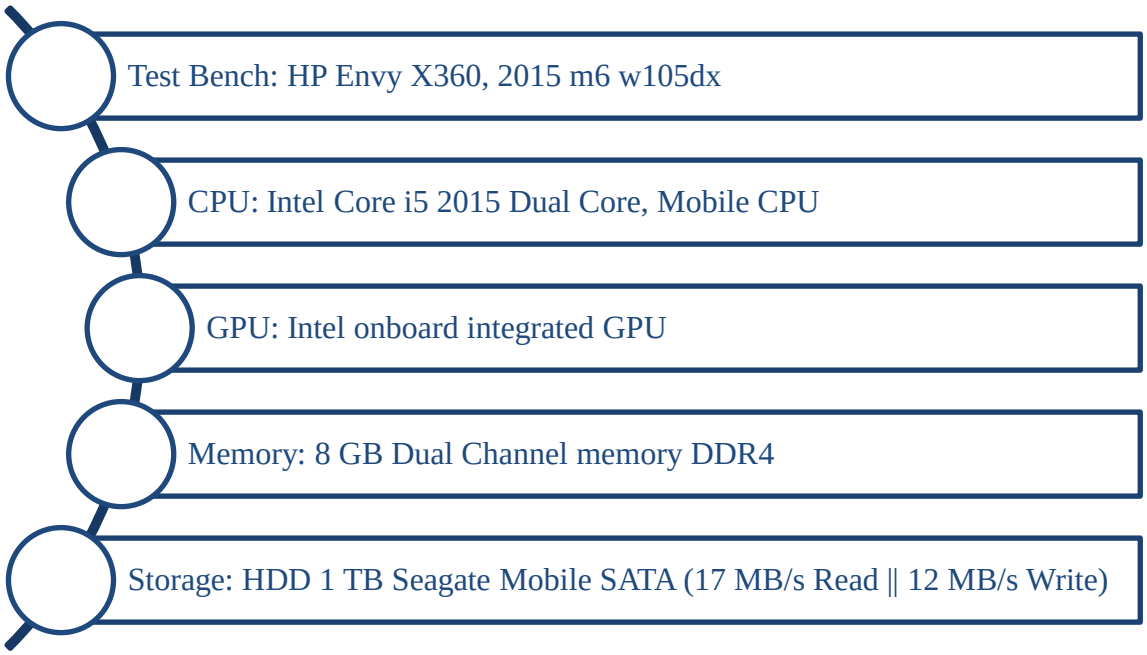
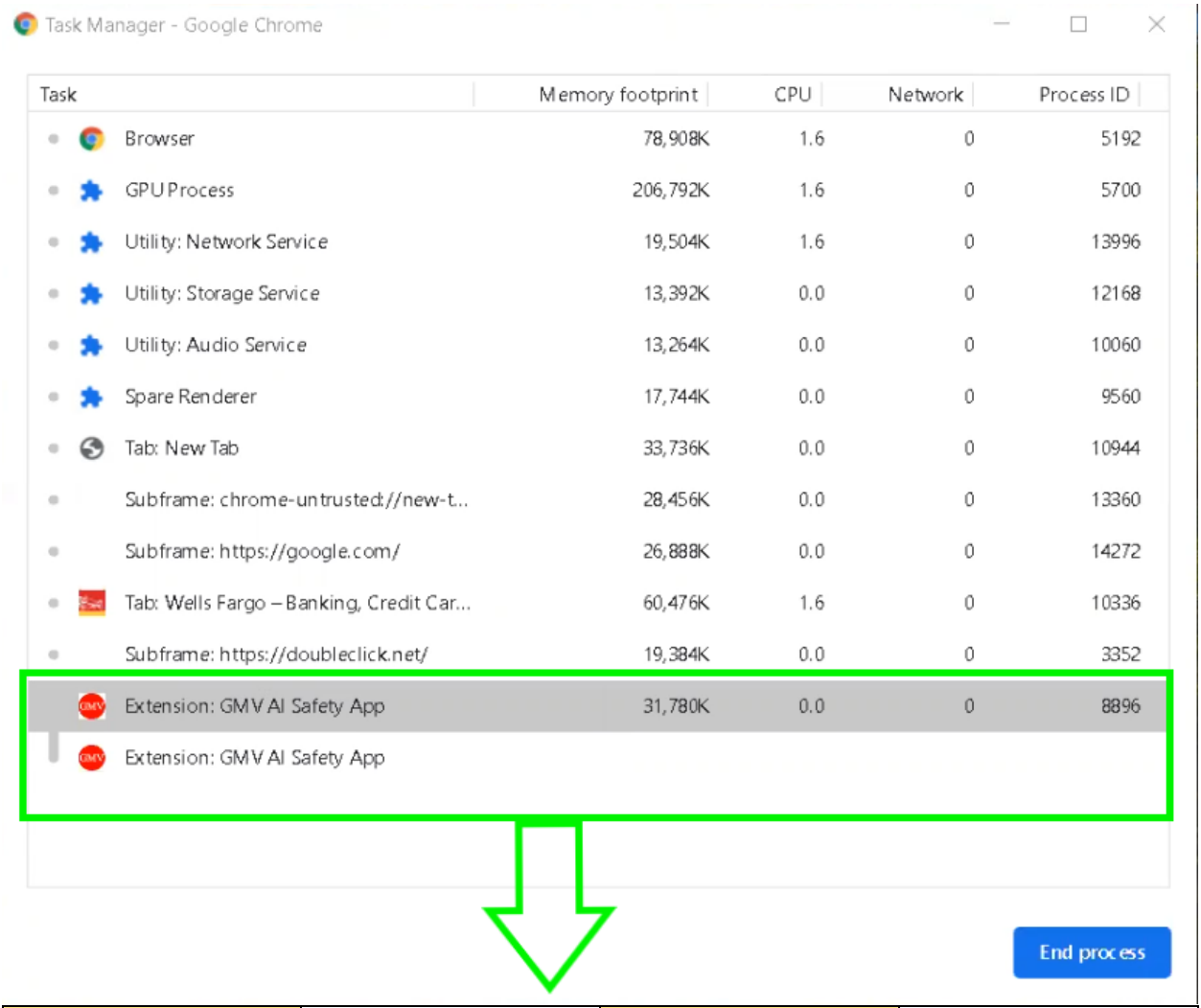


Figure 16: Test Bench Configuration

7.2. Performance of AI User Safety Application

7.2.1. Application Performance Running Idle

CPU/GPU utilization of application while in idle and no prediction is zero and continues to be zero until the application is not clicked.

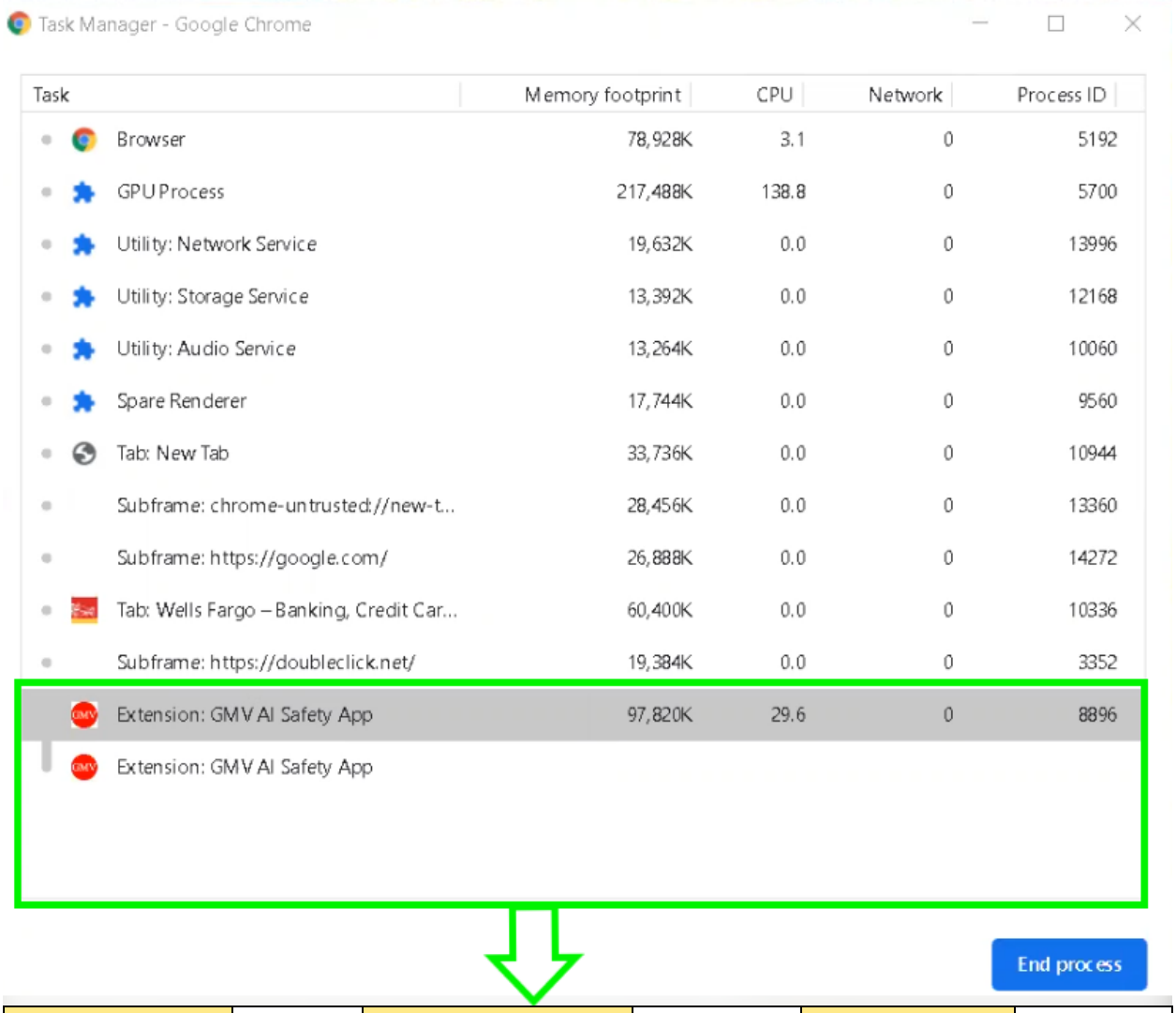


CPU Utilization	0 %	RAM Utilization	31.7 MB

Figure 17: Performance of AI User Safety Application when Idle

7.2.2. Application Performance During Prediction

CCPU/GPU utilization and memory consumption while the application is predicting the results are shown below.



CPU Utilization	29.6 %	RAM Utilization	97.8 MB	Duration	3 Seconds

Figure 18: Performance of AI User Safety Application during Prediction

7.3. Benchmark Performance

One of the most primary objectives of the application is to ensure that a faster, efficient, and accurate response can be achieved without compromising on the user's resources. Most of the commercial phishing solutions in the market are very resource-intensive with high usage of Memory and Processing power. Due to higher resource consumption, it is seen that the results published by these solutions can take time to predict. Hence, we designed our application by keeping this criterion as the boundary condition. Below are the test results from other commercial applications and the performance of AI user safety application against the same criteria.

The performance of AI user safety applications on phishing detection can be observed below from a resource perspective.

7.3.1. CPU Utilization Benchmarking

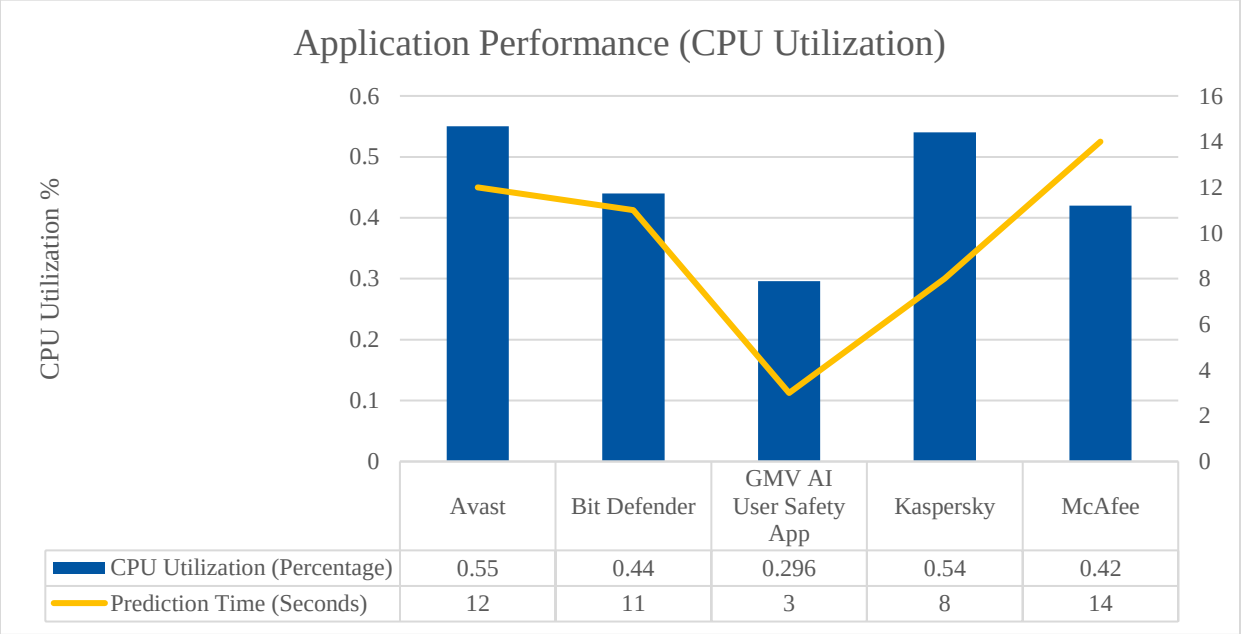


Figure 19: AI User Safety Application Benchmark performance CPU utilization

7.3.2. RAM Utilization Benchmarking

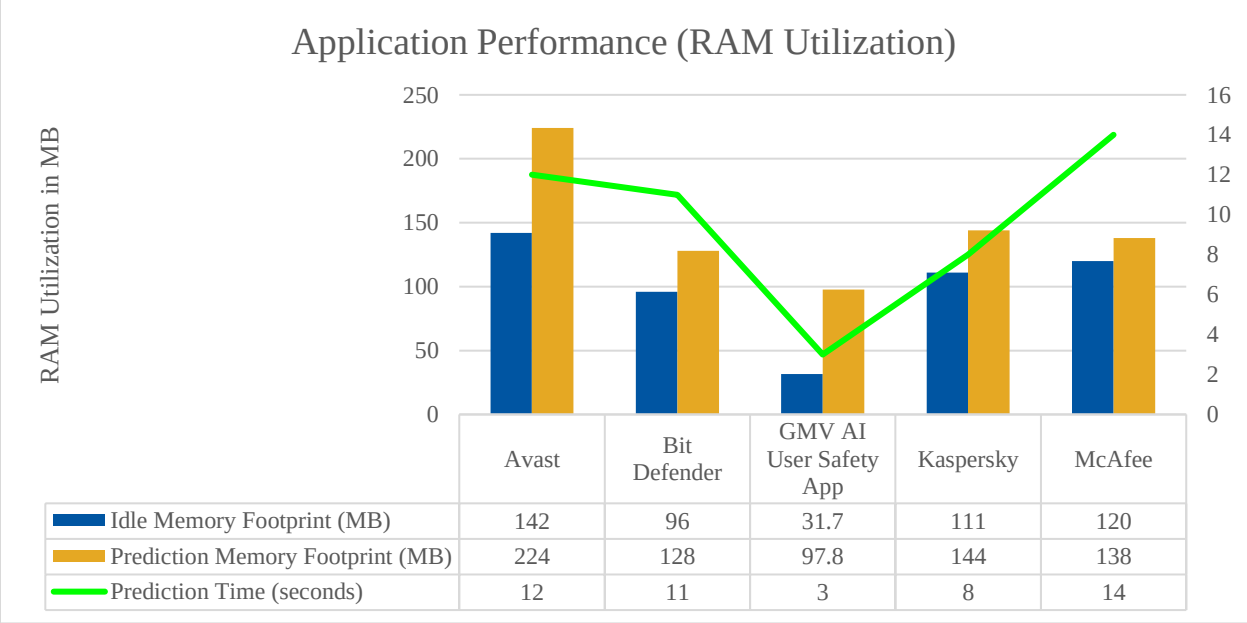


Figure 20: AI User Safety Application Benchmark performance RAM utilization

7.3.3. Time to Detect and Predict a phishing website

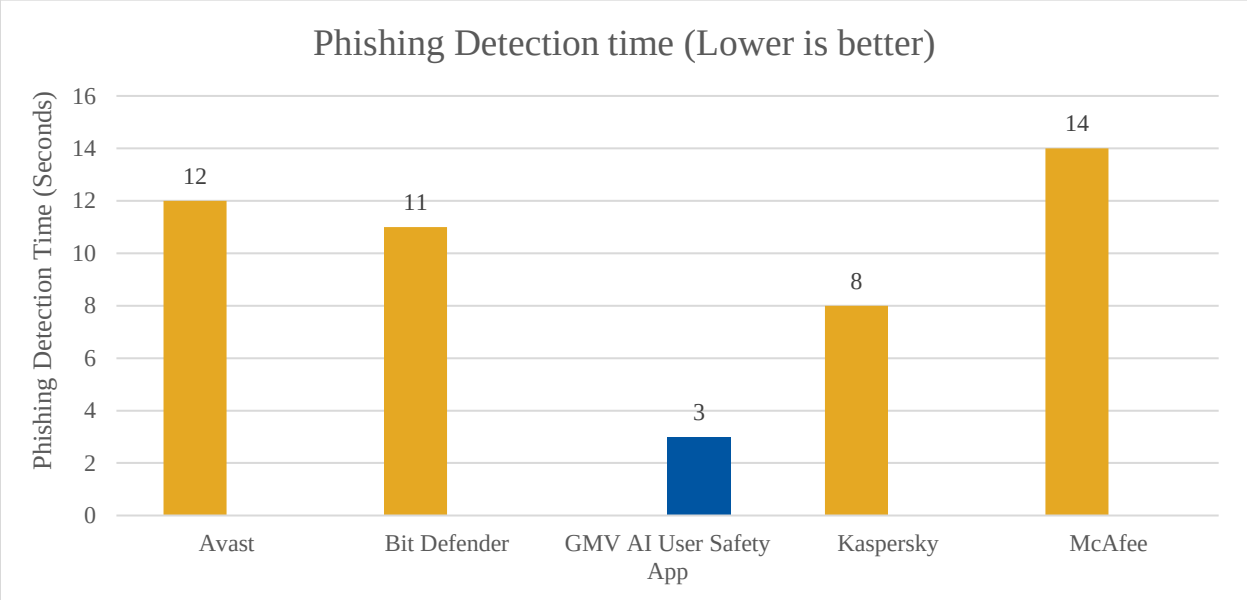


Figure 21: AI User safety Application Benchmark performance: Minimum Time to predict a phishing website

7.3.4. Performance Table

Table 3: Summary of Benchmarked Performance for AI User Safety Application

Resources	Bit Defender	Kaspersky	Avast	McAfee	GMV AI User Safety App
	Total Security 2021	Total Security 2021	Antivirus Service 2021 + Browser	Total Protection 2021	Version 12
Mean Idle CPU utilization	0%	0%	0%	0%	0%
Mean Idle Memory Utilization (MB)	96	111	142	120	31.7
Mean Prediction CPU Utilization	44%	54%	55%	42%	29.60%
Mean Time to predict (seconds)	11	8	12	14	3
Prediction Memory Utilization (MB)	128	144	224	138	97.8
Number of websites correctly categorized as phishing (120 links from PhishTank)	94%	93%	75%	81%	87%
AI model at backend detection	No	No	No	No	Yes
Prediction mode	Local Database lookup	Local Database lookup	Online Database lookup	Online Database lookup	Pattern Recognition

Chapter 8. Deployment, Operations, and Maintenance

The entire application is a plug and play and no configuration needs to be carried out by any user. The application download location and installation instructions can be seen in Chapter 10. Below we have the screenshot for application loading and results based on the initial project design.

8.1. Front-End Application Deployment

The front-end application is developed as a Google Chrome extension application. The client-side environment will be Google Chrome (Or any other Chromium platform browser).

8.1.1. Application placement in Chrome

The application is loaded as an Extension and can be seen on chrome extensions in the default place (right upper side of the browser) as seen below:

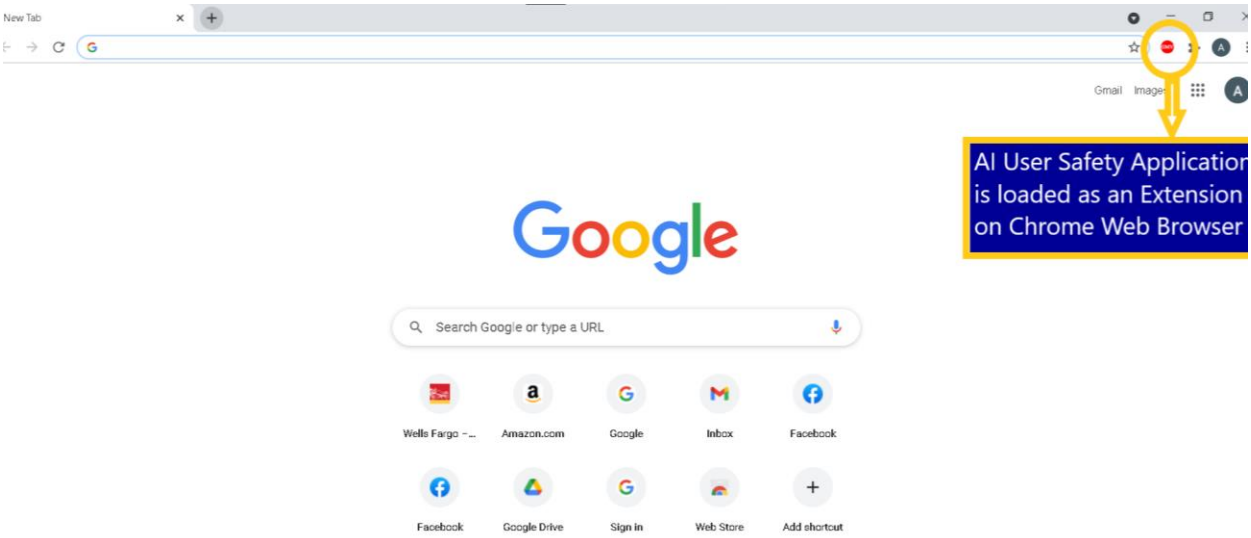


Figure 22: AI User Safety Application placement in a Web Browser

8.1.2. Application Loading and startup

The Extension application immediately starts up in google chrome the moment it is clicked:

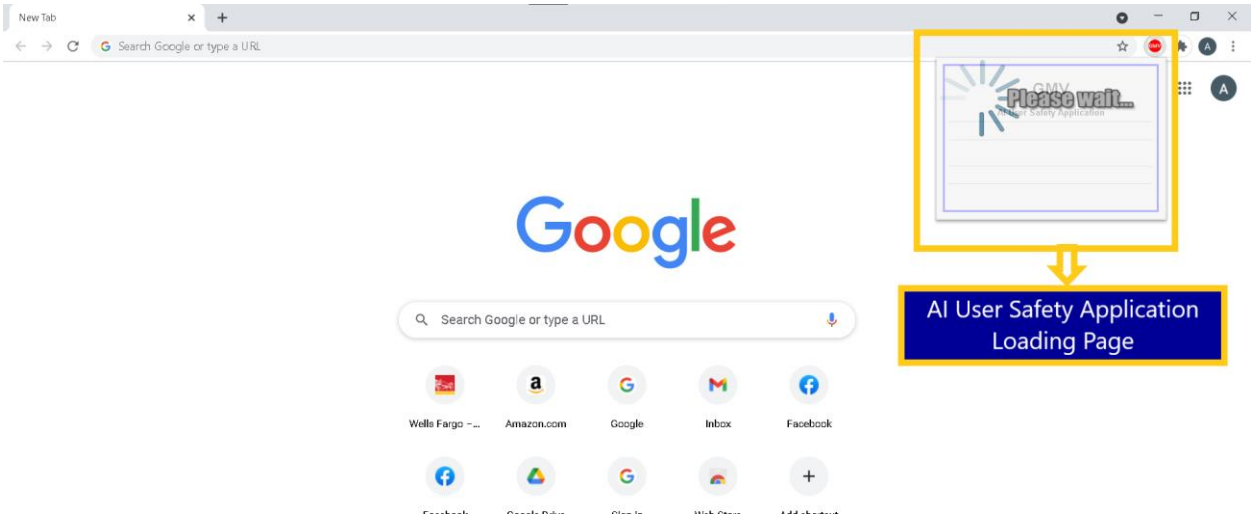


Figure 23: AI User Safety Application Loading Page

8.1.3. Application Prediction and Final Result

8.1.3.1. Legitimate Website Result

Below is the result for a legitimate website.

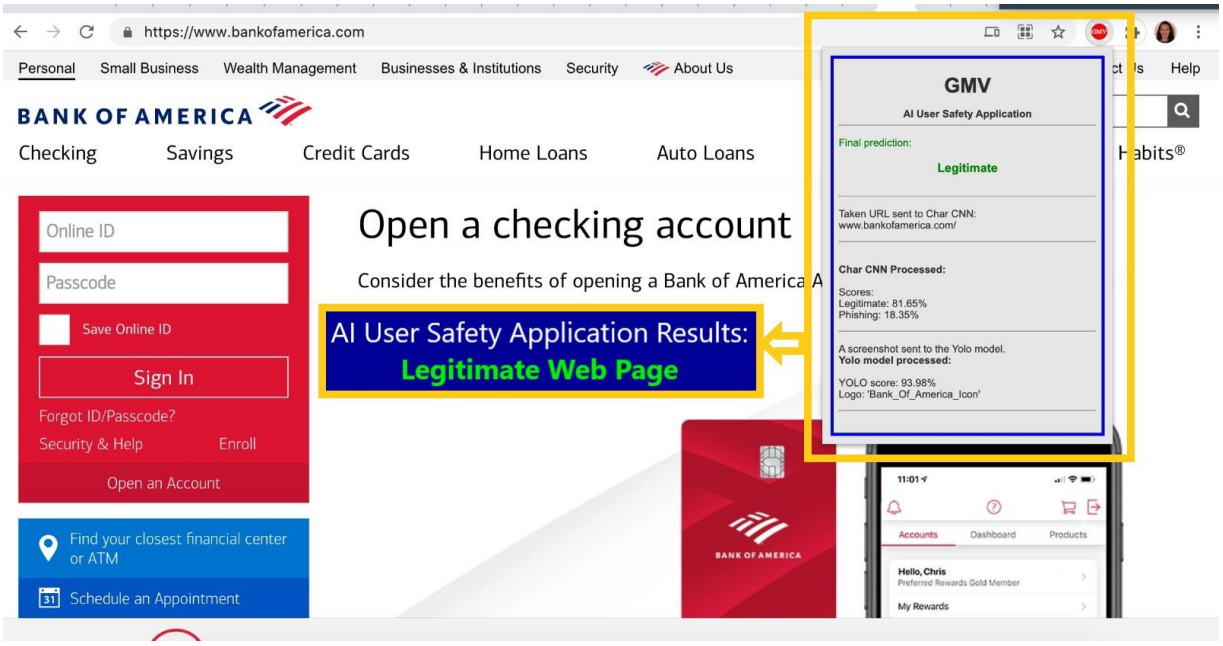


Figure 24: AI User Safety Application Prediction Results: Legitimate web page

8.1.3.2.Suspicious Website Result

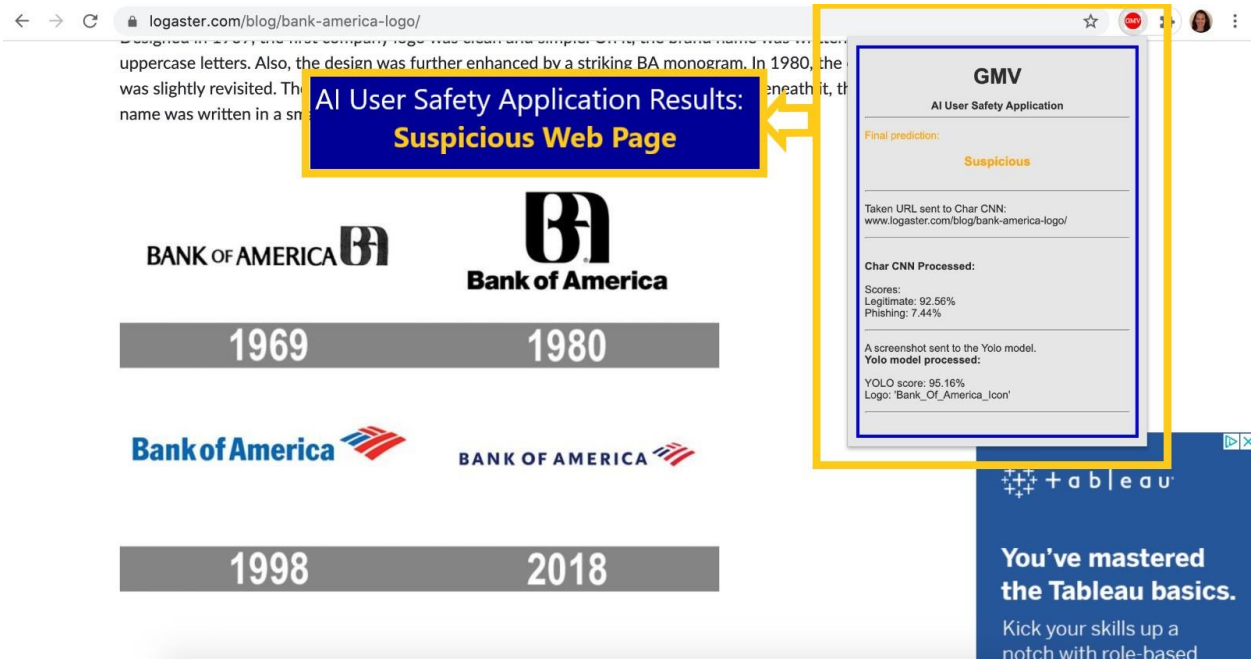


Figure 25: AI User Safety Application Prediction Results: Suspicious webpage

8.1.3.3.Phishing Website Result

Figure 26: AI User Safety Application Prediction Results: Phishing webpage

8.2. Backend Application Deployment

The backend of the application was a YOLO model which is very resource-intensive and hence has been deployed on AWS-EC2 as a Python-Flask-based server.

8.2.1. AWS EC2 Server

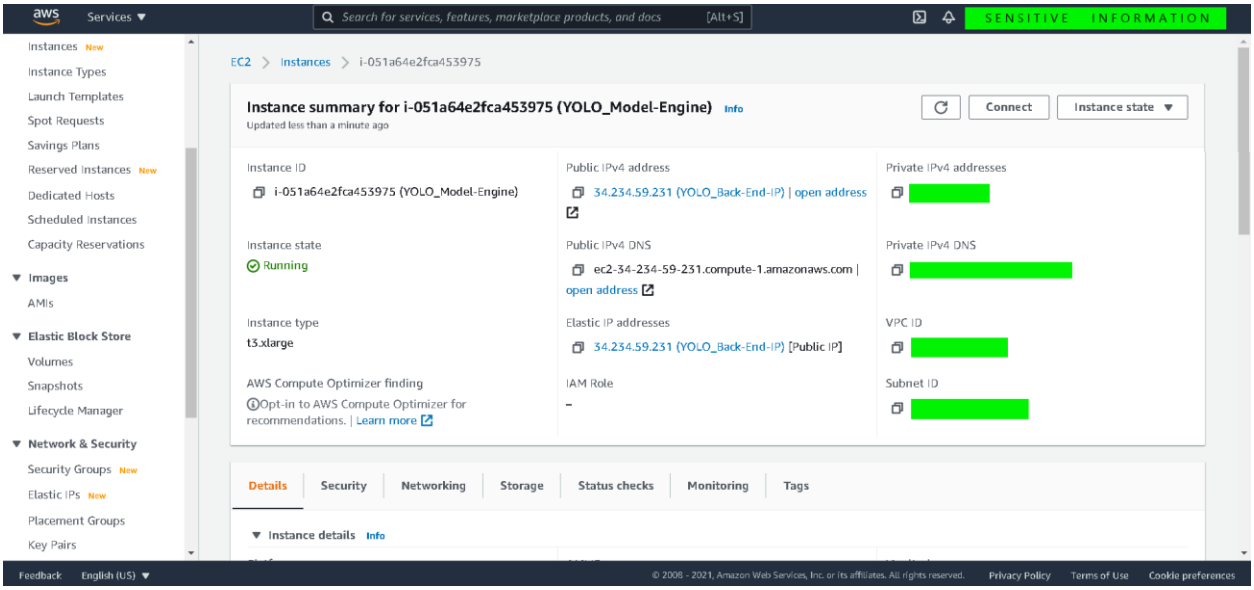


Figure 27: Backend Server Details AWS EC2 T3.xLarge

8.2.2. AWS Prediction Server Performance

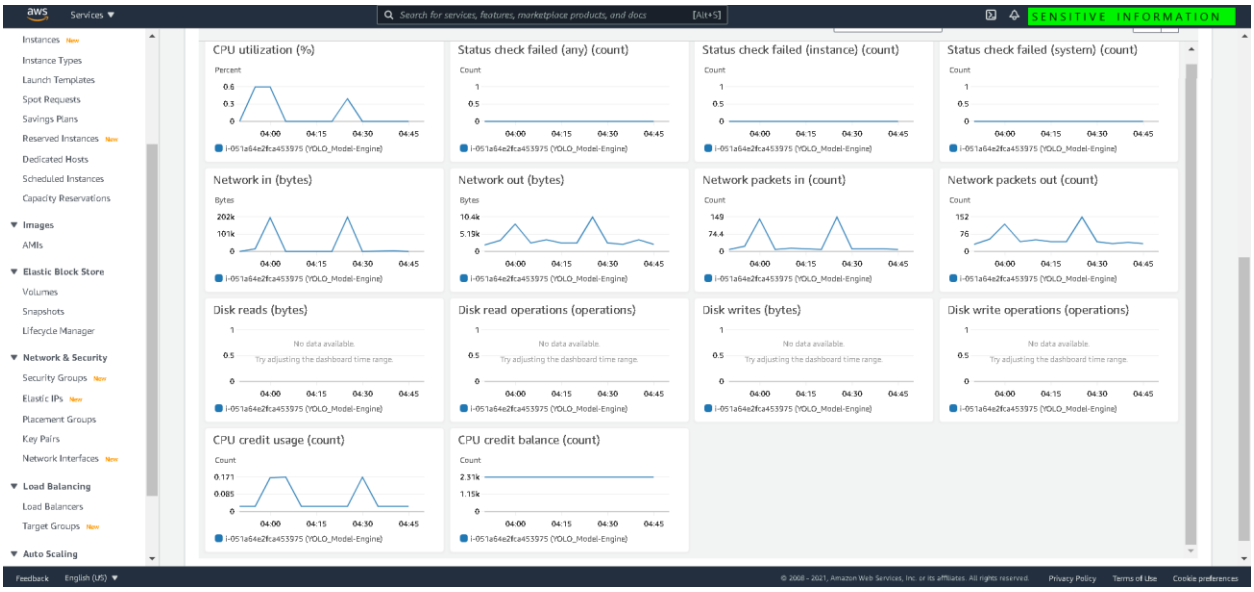


Figure 28: Backend Server Performance at the time of Prediction

8.2.3. *Docker-Flask Container for the Backend Hosting*

The backend development is done on the flask framework and python programming language. The application is then hosted on a Docker container. The entire application can be started and stopped using a single line of code which is $ *sudo docker start* and $ *sudo docker stop*. The docker container image can be seen below:

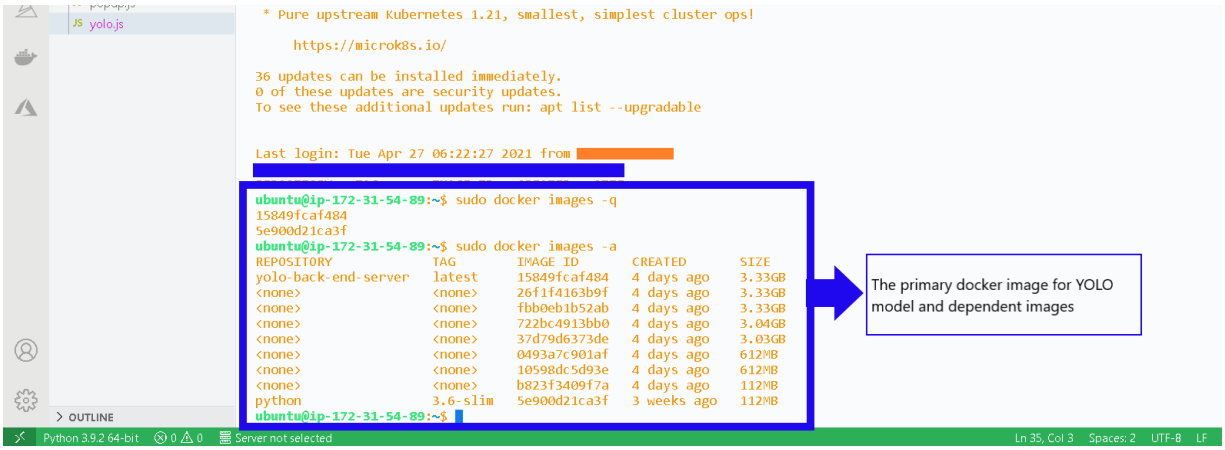


Figure 29: Docker Container Image for the backend Application

8.2.4. *Project Directory structure*

Below is the project tree that has been deployed on the backend server.

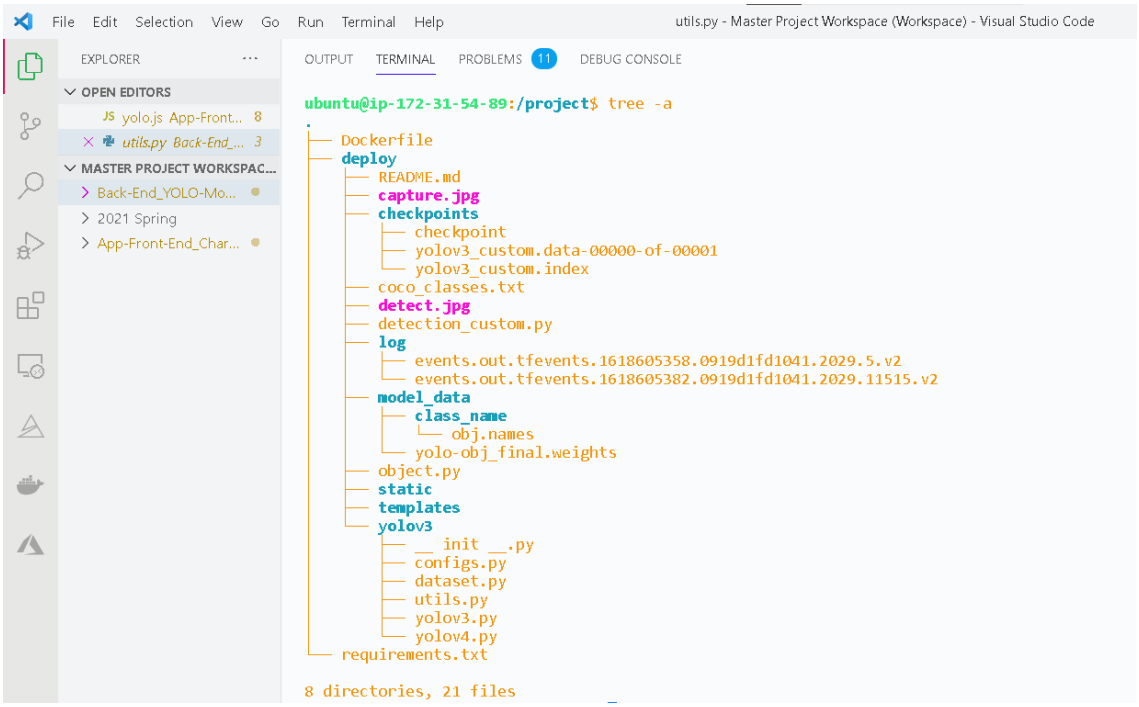


Figure 30: Backend Application Directory Tree

8.3. Operations and Maintenance

Presently the application is secured based on guidelines shared by Google so that it can be deployed on Chrome Web Store and can be installed directly from there instead of GitHub account.

Below is the operational process for supporting the application until it is retired

8.3.1. Maintenance Process

The models will be evaluated periodically (1st Saturday of every month) on the new and updated dataset. If the model performance is better than the existing model by more than 9% then the production model can be upgraded to the latest model.

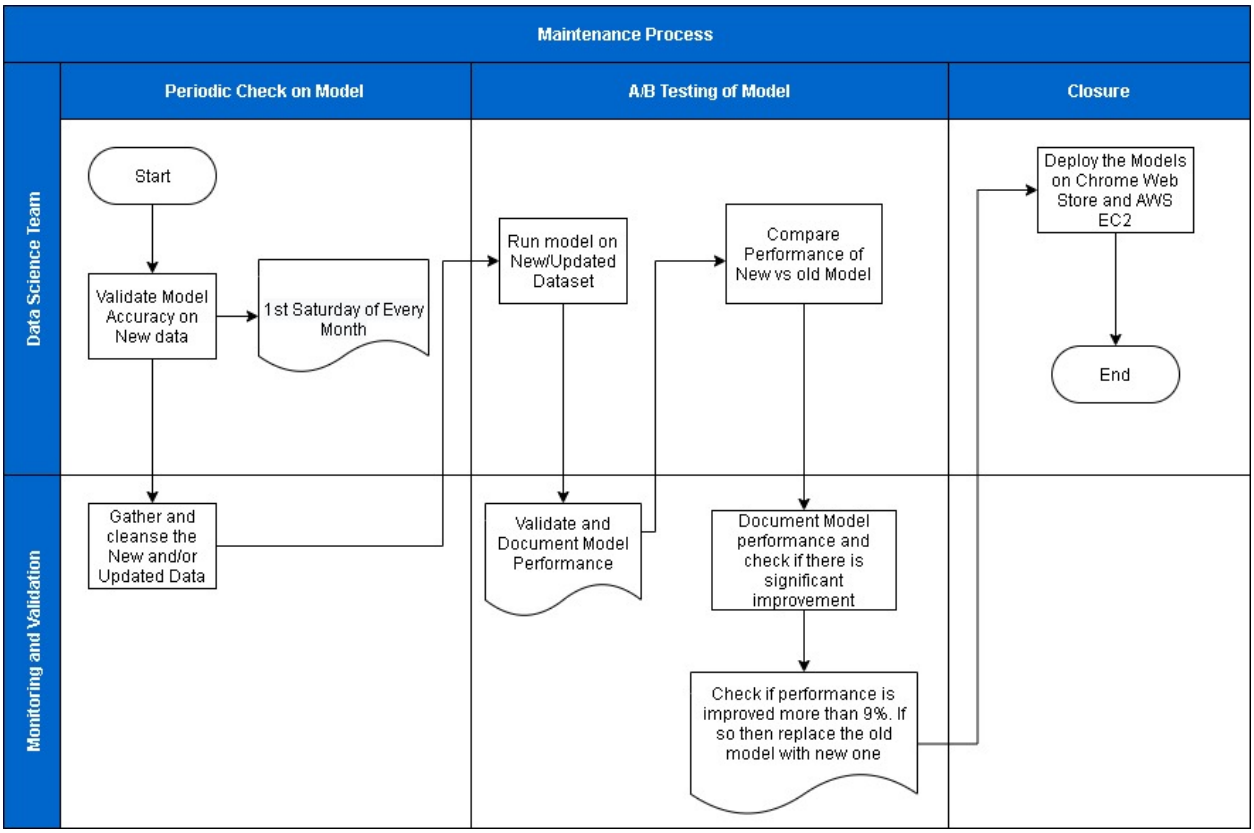


Figure 31: Maintenance Process for GMV AI User Safety Application

8.3.2. Incident Management

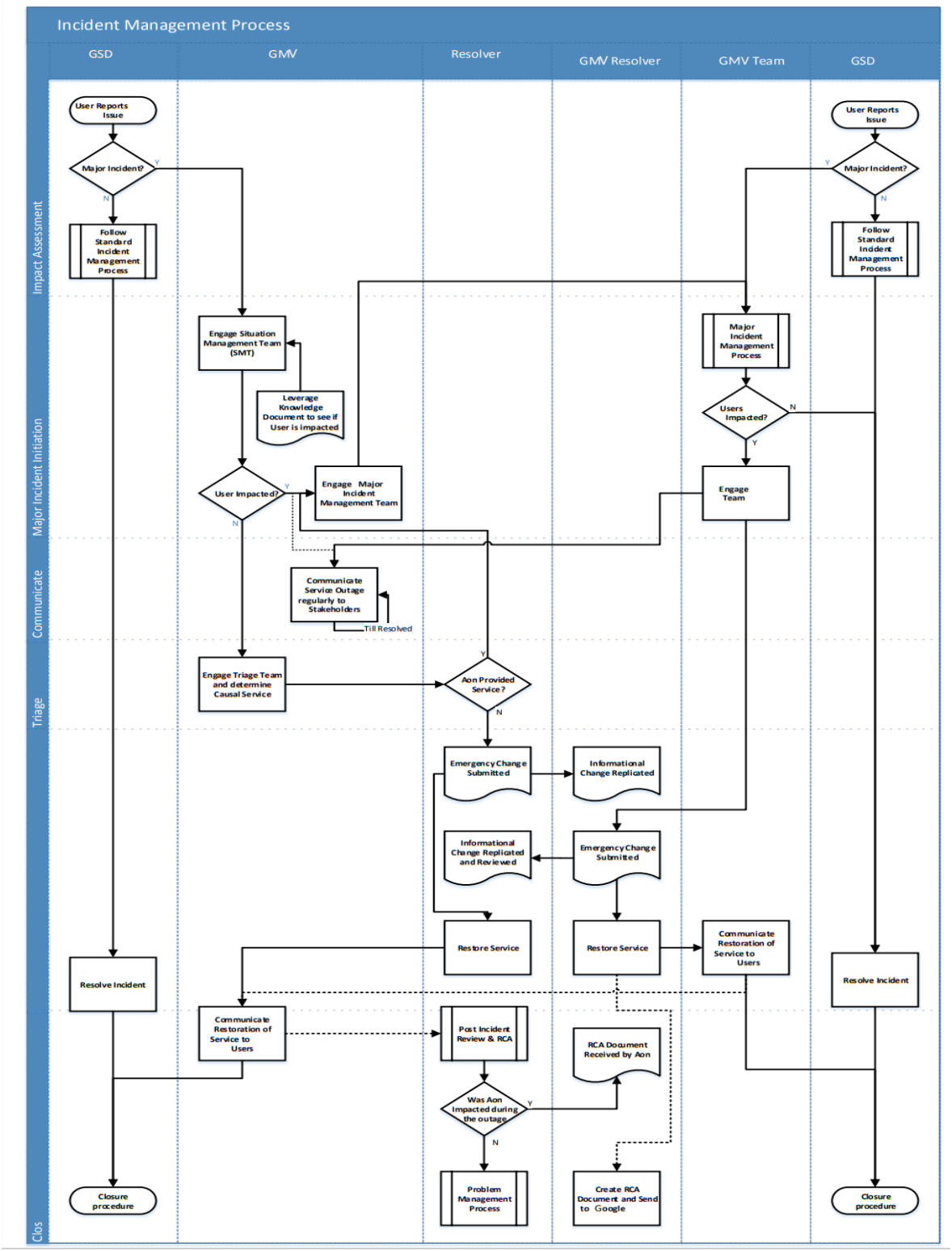


Figure 32: Incident Management Process for GMV AI User Safety Application

Chapter 9. Model Cards

Below are the model cards of ML/DL algorithms used and models created based on Google Model Cards [9].

9.1. YOLO v3 Object Detection

9.1.1. Overview

The YOLO model analyzed in this card detects one or more logos in a webpage as an object. The web pages are provided as an image that is base64 encoded string which is decoded at the backend into an image for input. If the logo provided is not there in the trained class, then the model returns an empty string. If the logo provided is in existence, then the model returns the *Class Name and Confidence Score*, along with the image with bounding boxes.

9.1.2. Model description

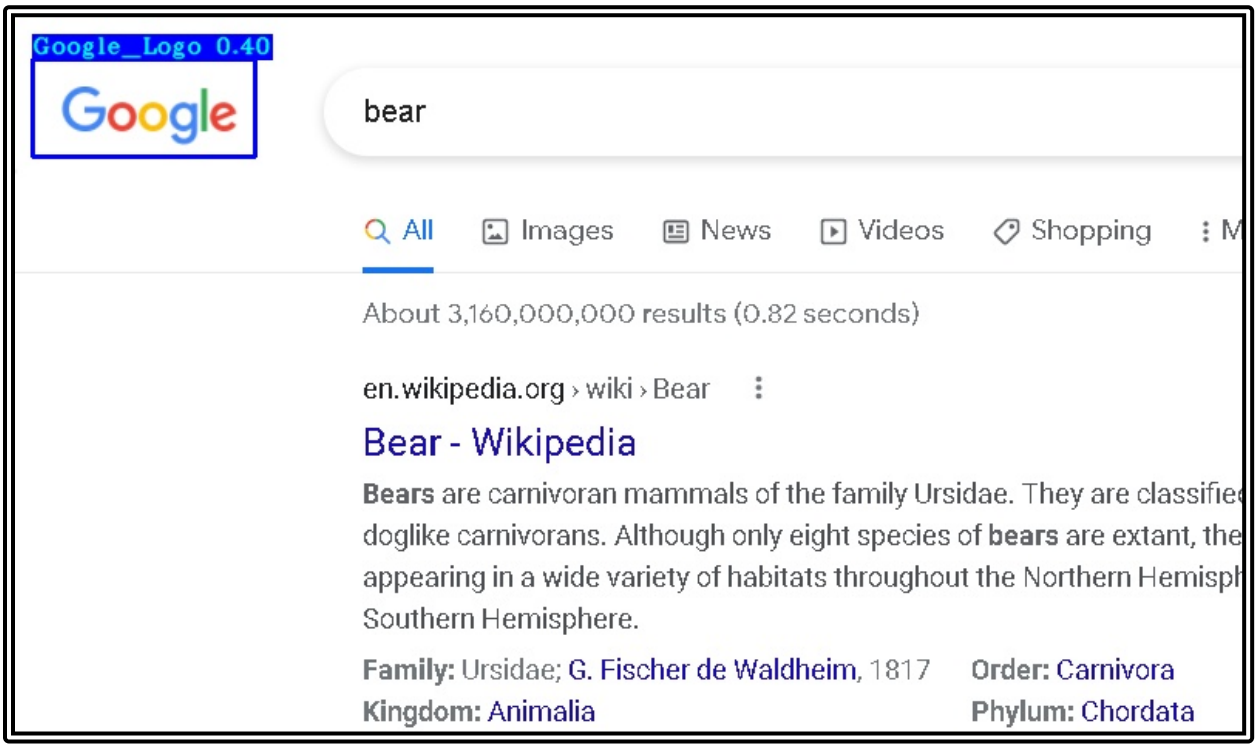


Figure 33: YOLOv3 Model Object Detection. The Output contains a Bounding Box with Label and Confidence Score

Input: Photo or Video

Output: The model can detect 12 classes based on the initial training data given. For each detected object, the model returns:

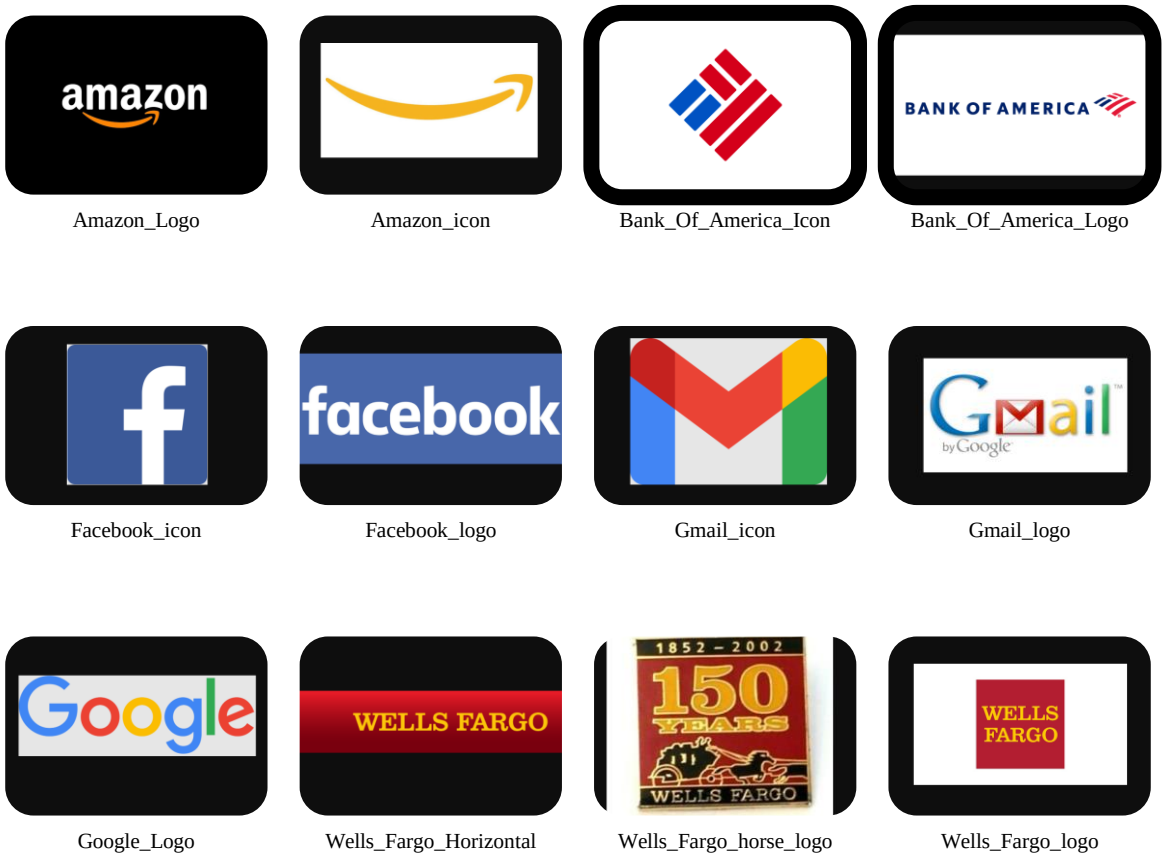
a) Class Name

b) Confidence Score

c) Bounding boxes coordinates

d) Image with bounding box and label names

Class Names: Following are logos that the model can currently predict accurately.



Amazon_Logo | Amazon_icon | Bank_Of_America_Icon | Bank_Of_America_Logo

Facebook_icon | Facebook_logo | Gmail_icon | Gmail_logo

Google_Logo | Wells_Fargo_Horizontal | Wells_Fargo_horse_logo | Wells_Fargo_logo

Figure 34: Class names of Logo that YOLO Model can accurately detect as an object in a web page.

Model Architecture: The model selected for logo detection is YOLO v3. The model is based on darknet architecture which was translated into a TensorFlow model.

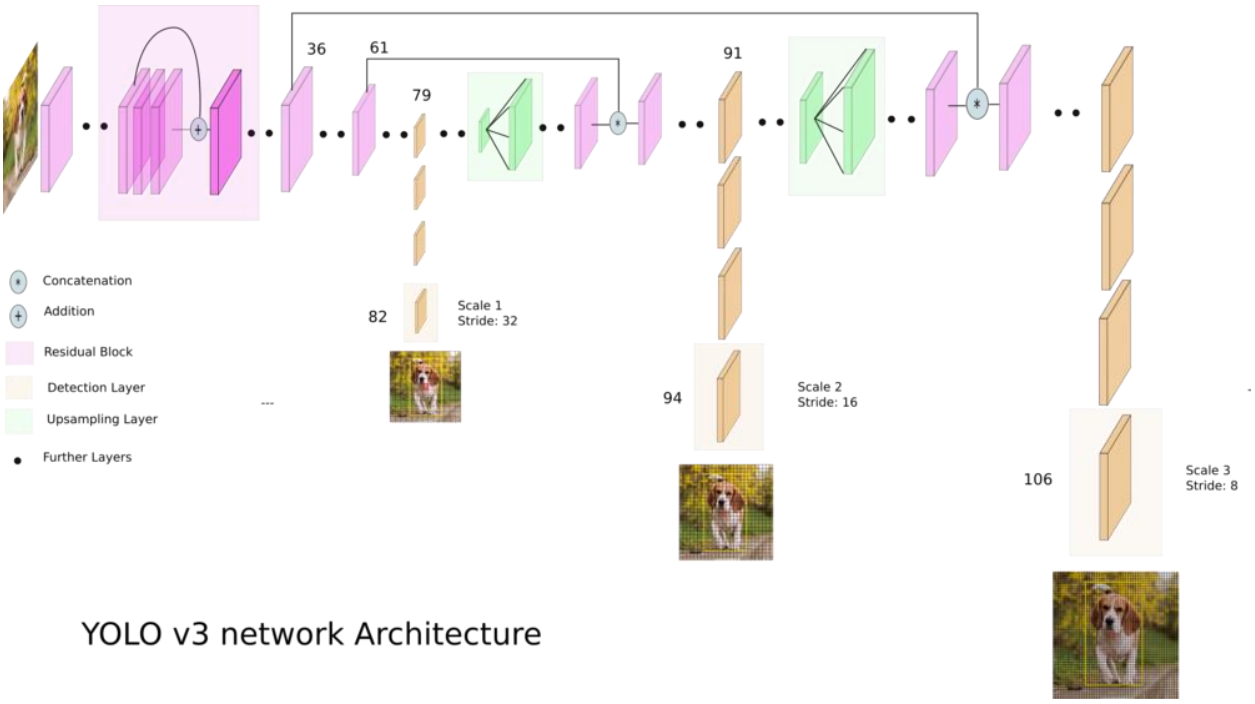


Figure 35: YOLO v3 Model Architecture. Source: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

9.1.3. Dataset

The dataset compiled for the model is a custom dataset that has been compiled using Hyper label. The dataset can be taken from the google drive link here.

9.1.4. Limitations

Following are the list of limitations of the model:

a) Model is correctly able to identify only 12 different types of logo

b) The current threshold set for logo detection as an object is for a confidence score of 0.7 or 70%. The confidence score of lower values will result in an output of empty string

c) The model takes input as an image. Since chrome does not allow for screenshot capture, we had to encode the entire webpage and pass it through the REST API in base64 strings

d) The minimum resolution of the image captured should be at least 416 X 416 X 3 (L X W X C) pixels for the model to give an accurate prediction

9.1.5. Performance Metrics

Below is the model performance for test results. The model is evaluated on the below parameters:

a) General Intersection Over Union (Giroux) loss: Bounding box regression is the crucial step in object detection. In existing methods, while `n-norm loss is widely adopted for bounding box regression, it is not tailored to the evaluation metric, Intersection over Union (Iowa) et al Zheng [7]. For our project, we calculate the Giroux loss as defined under:

$$GIoU = iou - 1.0 \times \frac{(enclosed\ area - \ union\ area)}{enclosed\ area}$$

The value returned is the *Reduced Mean* value of *Sum of Giroux loss.*

b) Confidence loss: The confidence score suggests how confident the model is after detecting the logo (object) and how accurately it is captured by the bounding box. It works under the criteria that if the largest IoU is less than the threshold it is considered that the prediction box contains no objects, and it returns a zero-output else it can be defined as under:

$$Confidence\ Score = probability\ (object)\ \times iou\ (predicted\ output\ and\ actual\ output)$$

Each bounding box consists of the following parameters (x, y, w, h, confidence). (x, y) are box coordinates, (w, h) is height and width of the bounding box, confidence is predicted confidence tensor values from the final layer of the YOLO model

c) Mean Average Precision: Mean Average Precision can be defined as the average of Average Precision (AP). It is defined as under:

$$mAP = \mu\left(\sum \frac{TP}{TP + FP}\right)$$

TP is true positives and FP is false positives identified by the model.

d) Total Loss: Total loss is defined directly from the loss function identified from the research paper of YOLO et al Redmon [6] and can be seen from the below image:

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

Figure 36: Total Loss Equation took from YOLO research paper [6]

9.1.6. Performance Results from TensorBoard

9.1.6.1. Training Results

Below are the training results based on the performance metrics targeted. It is to be noted that the learning rate was variable and was changing over time based on changes in loss fluctuations:

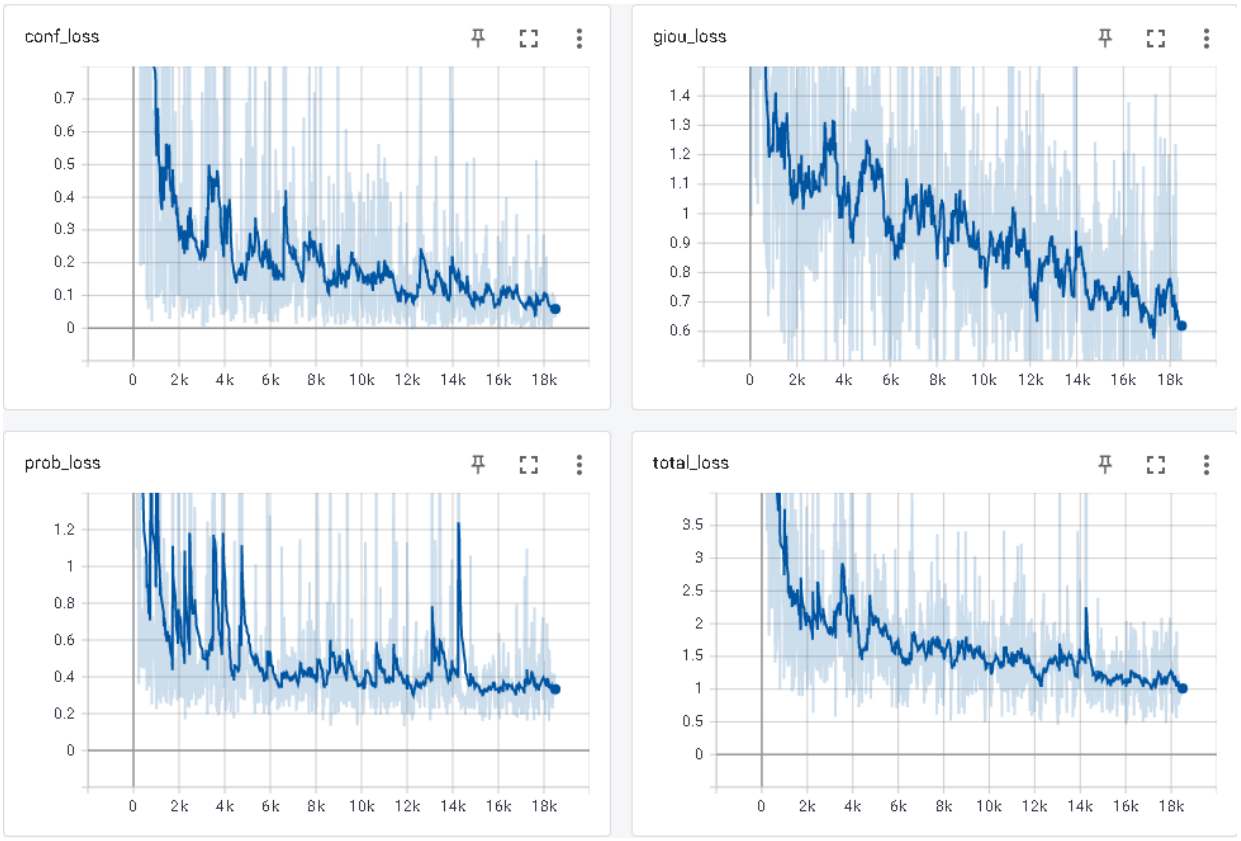


Figure 37: Training Results from TensorBoard. Loss is gradually decreasing over time. The spikes are observed with a change in learning rate.

9.1.6.2.Validation Results

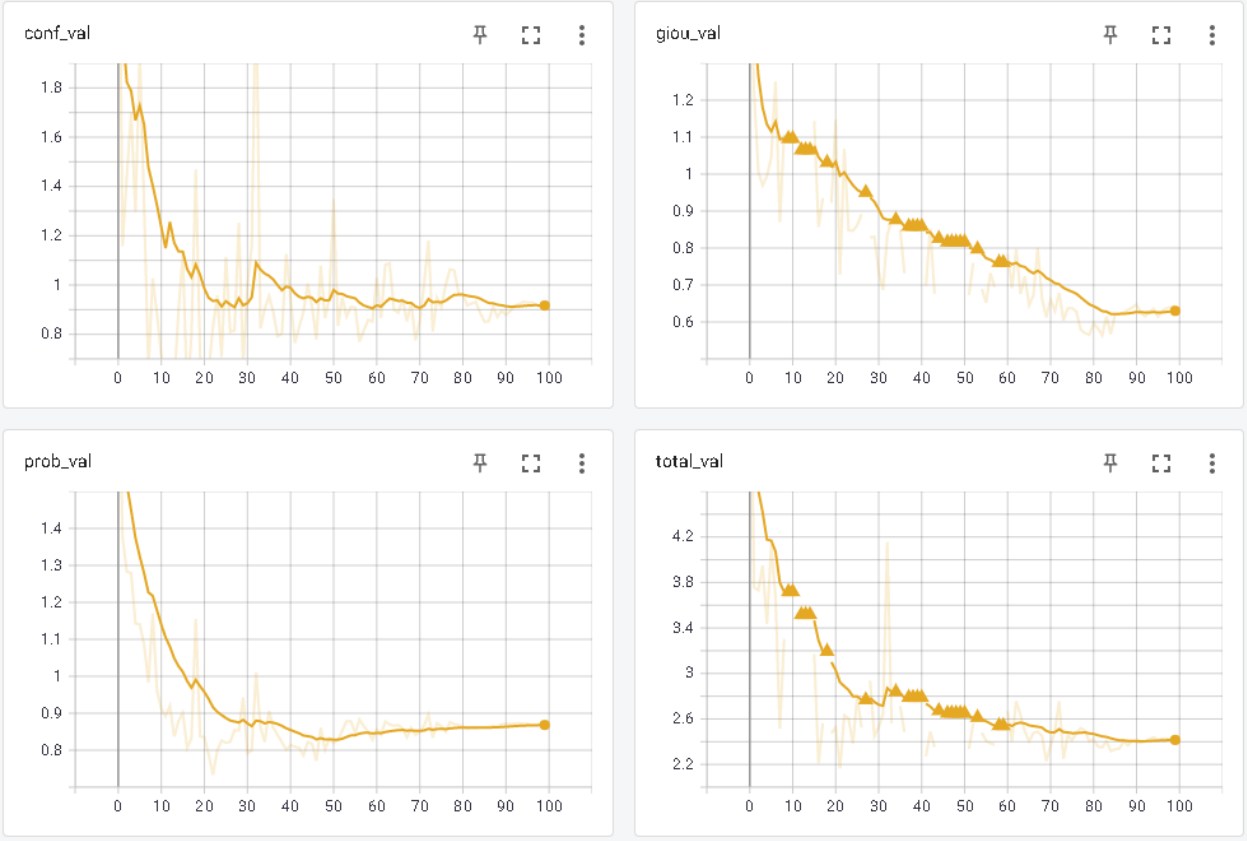


Figure 38: Validation results from TensorBoard. The model predicts accurately and well within the range of acceptance

9.1.6.3.Performance Summary

The model is trained and evaluated over a custom dataset. The entire dataset was hand-annotated, and each class has at least 160 images with a total cumulative dataset of 1014 images. The dataset was randomly split with a 90:10 ratio for training and testing respectively

a) The overall mAP score for the model is 90 (out of 100) and can accurately predict the objects as logo

b) The IoU threshold during training was set for 0.5 (50%). The final return confidence score threshold is set to 0.72 which means the model returns only when confidence is greater than ~70%.

c) The model does not bode well if the image resolution is below 416 px (w x h). Also, the greater the image size as input to the model the longer is the prediction time.

9.2. Char-CNN URL Validation

9.2.1. Overview

The model analyzed here is a character recognition model which is used to recognize different characters of the URL individually. The backbone of the model is a CNN (Convolutional Neural Network). The model is a classification model which takes the strings as input and carries out a scan to identify potential anomalies and phishing patterns and then determine the output as a binary classification (1 – Phishing, 0 – Legitimate) with Sigmoid activation giving output between 0.0 and 1.0. The output can be interpreted as the probability of the "Phishing" class that is encoded as 1. Assuming the threshold is 0.5 if the output is greater than or equal to 0.5 the predicted class is 1 (Phishing), and if the output is less than 0.5 the predicted class is 0 (Legitimate).

9.2.2. Model Description

Below is the model architecture for Char-CNN.

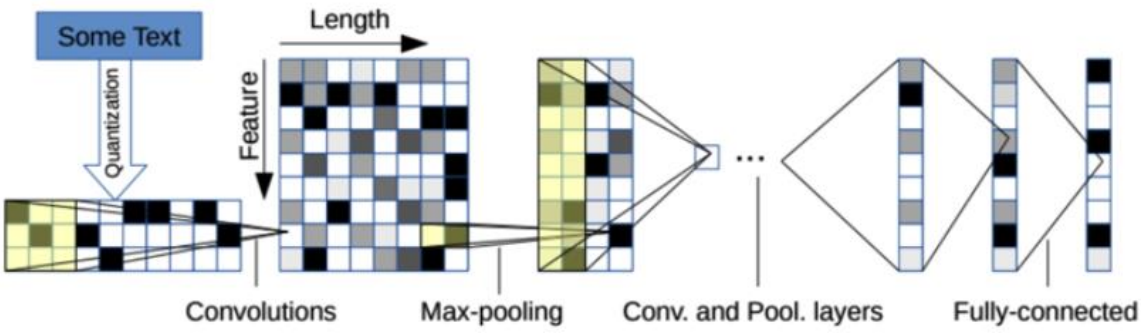


Figure 39: Char-CNN Model Architecture diagram: Source: Character level CNN with Keras. In this notebook, we will build a… | by Xu LIANG | Towards Data Science

The model is based on the proposed model by et al Kim [8] Convolutional Neural Network for Sentence Classification. Char CNN is a modification of this approach to help improve the OOV problem (Out of Vocabulary) problem.

9.2.3. Dataset

The dataset is an amalgamation of three datasets: PhishTank [14], PhishStorm [16], and The Majestic Million [15]. The total balanced dataset contains 164,940 URLs of phishing and legitimate websites.

9.2.4. Limitations

Following are the key limitations of the URL validation model:

a) The model is trained to recognize the patterns in URLs typically used by Phishers which are valid until Q2 2021. If the phishing trend changes then the char-CNN model needs to be retrained.

b) If a phisher identifies the patterns on which the model is trained on, then they can change the patterns of spoofing which will further impact the model accuracy.

c) The Char-CNN model only validates the URL and not the content of a web page.

9.2.5. Performance Results

Since it is a classification model, we are using standard metrics of Precision, Recall, Accuracy Loss, ROC curve, and F1 score as metrics for model performance evaluation. Below are the results from training and evaluation:

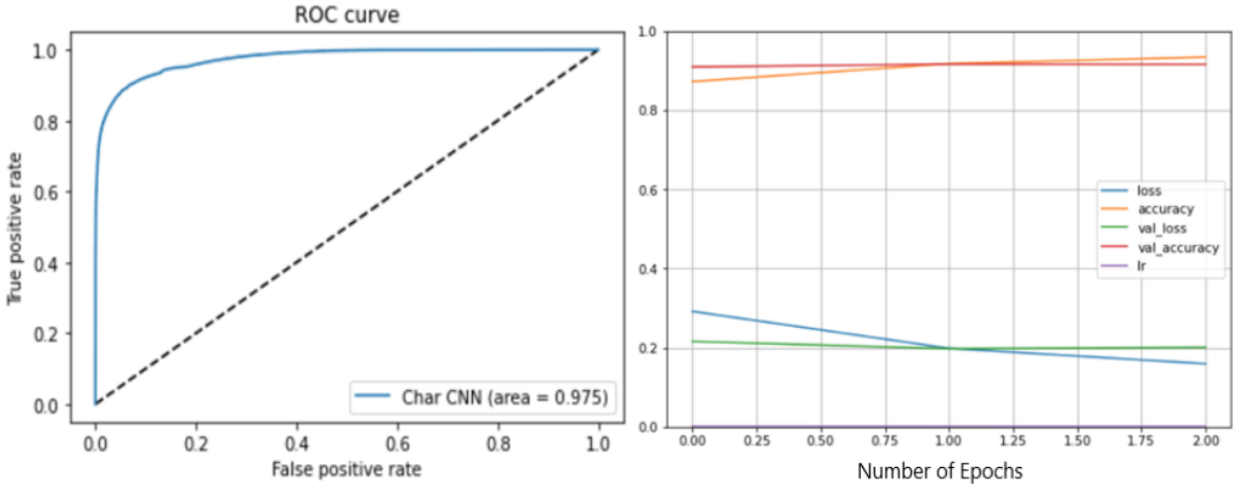


Figure 40: Performance of Char-CNN model over different metrics [AUC, Accuracy, Loss, Learning Rate]

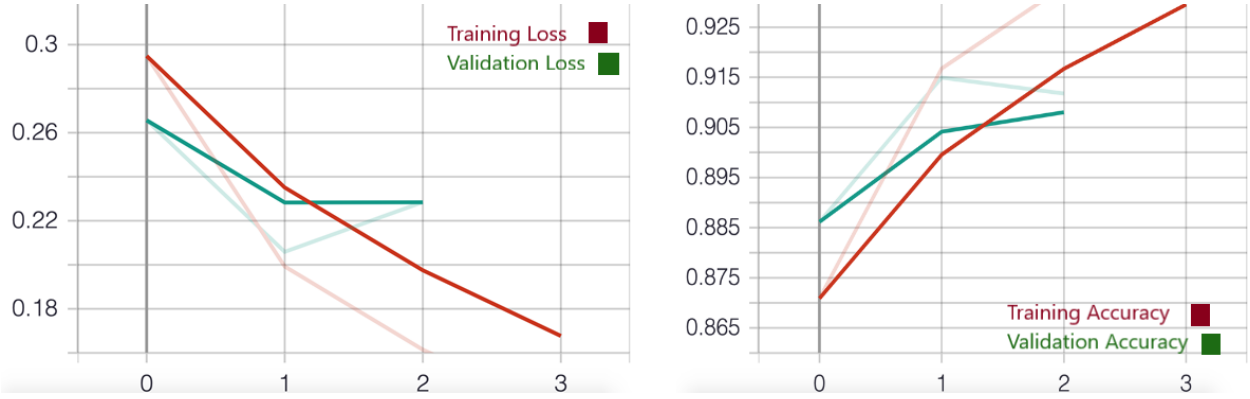
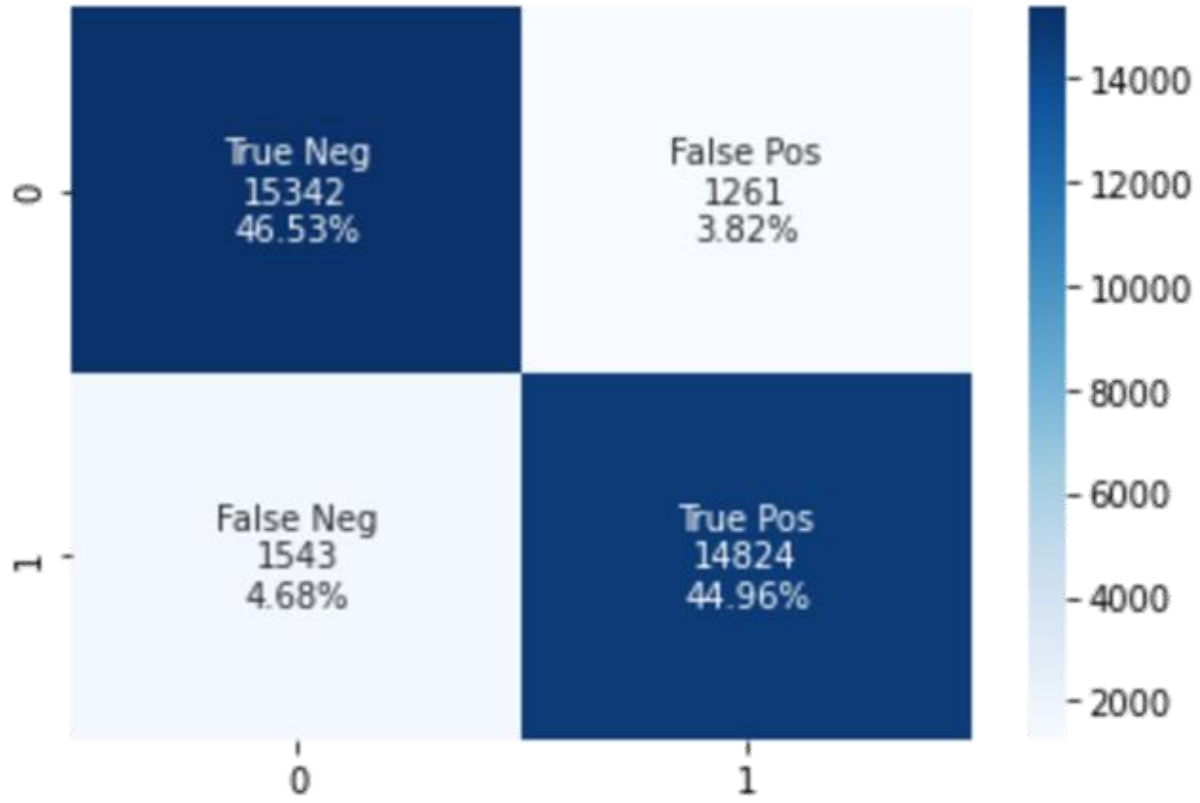


Figure 41: Tensorboard Evaluation Report for Char-CNN for Loss and Accuracy over Epochs



	precision	recall	f1-score	support
0	0.909	0.924	0.916	16603
1	0.922	0.906	0.914	16367
accuracy			0.915	32970
macro avg	0.915	0.915	0.915	32970
weighted avg	0.915	0.915	0.915	32970

Figure 42: Truth table for Char-CNN

9.3. Ensemble Modelling Logic

With the two deep learning models and having two varied outputs, we used custom logic to have the most efficacy from either of them. Below is the screenshot of the *pseudocode* implemented for the result display as a final output:

JS general_logic.js 3 ×

App-Front-End_Char-CNN-App-UI_TensorflowJS-JavaScript > JS general_logic.js > ...

```
1   var charCNN_Results = String;
2   var YOLO_Results = [];
3
4   if (charCNN_Results === "Phishing" && YOLO_Results["Conf_Score"] > "0.7") {
5       showMain_Results ("Phishing");
6   }
7
8   else if (YOLO_Results["Conf_Score"] > "0.7" && charCNN_Results === "valid") {
9       checkDomain();
10      showMain_Results("Suspicious");
11  }
12
13  else if (charCNN_Results === "Phishing") {
14      showMain_Results ("Suspicious")
15  }
16
17  else {
18      showMain_Results("Legitimate");
19  }
```

Figure 43: Pseudocode for Final Result Ensemble modeling logic

Chapter 10. Application Installation

The process to upload the application on Chrome Web Store is ongoing from which the users can directly download the application. Currently, the application is available to download from GitHub. The entire application is designed as a plug and play so no configuration is required. Follow the steps below to download the application:

10.1. Application Location

Click on the below-mentioned link to go to the application location:

https://github.com/varun-bhaseen/AI_User_Safety_Application.git

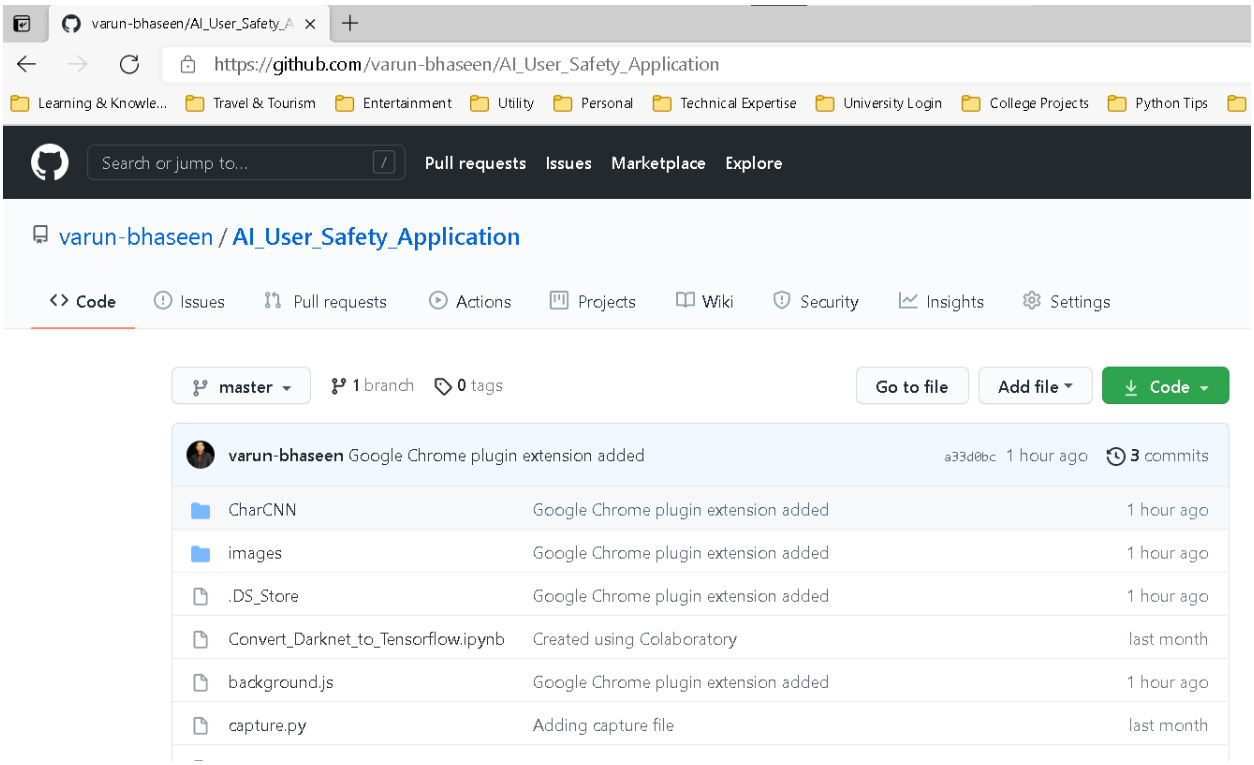


Figure 44: Application Source Location for End Users

10.2. Steps to Install the application

10.2.1. Step-1 Download the application

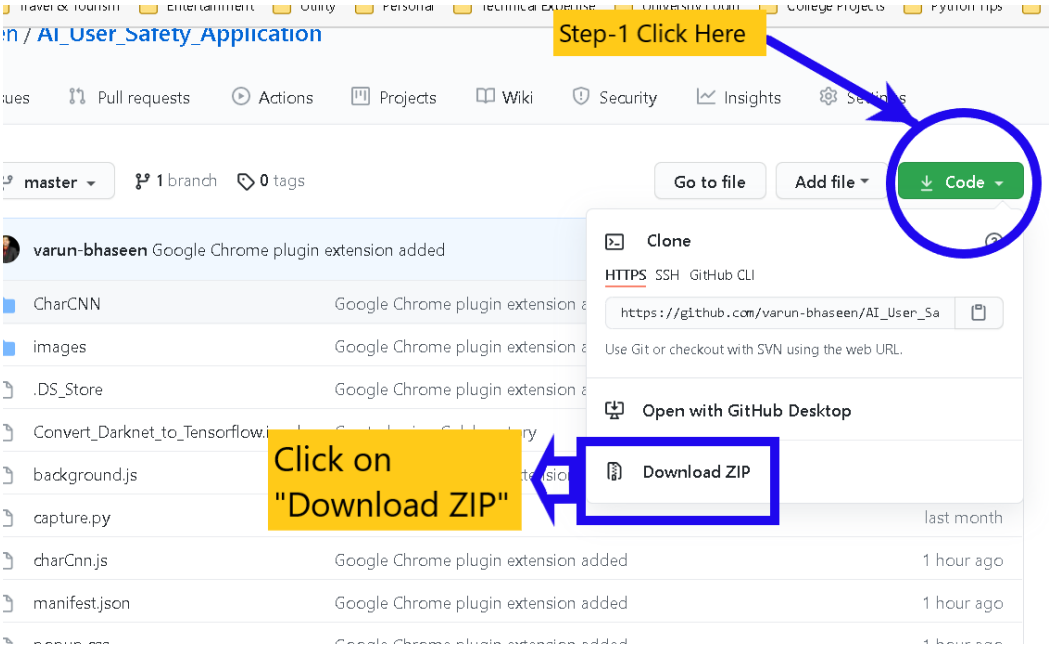


Figure 45: Download the Application

10.2.2. Step-2 Extract the contents of the Zip file

In this step first, go to the location where the Zip file has been downloaded by your browser.

(the default location in windows is the *Downloads* folder). Once the zip file is located, just right-click on the file and click on *Extract All*.

You can extract anywhere on your HDD but just remember the location of the extraction. The best practice is to create a folder named *Google* in *C:* and extract it there as shown below:

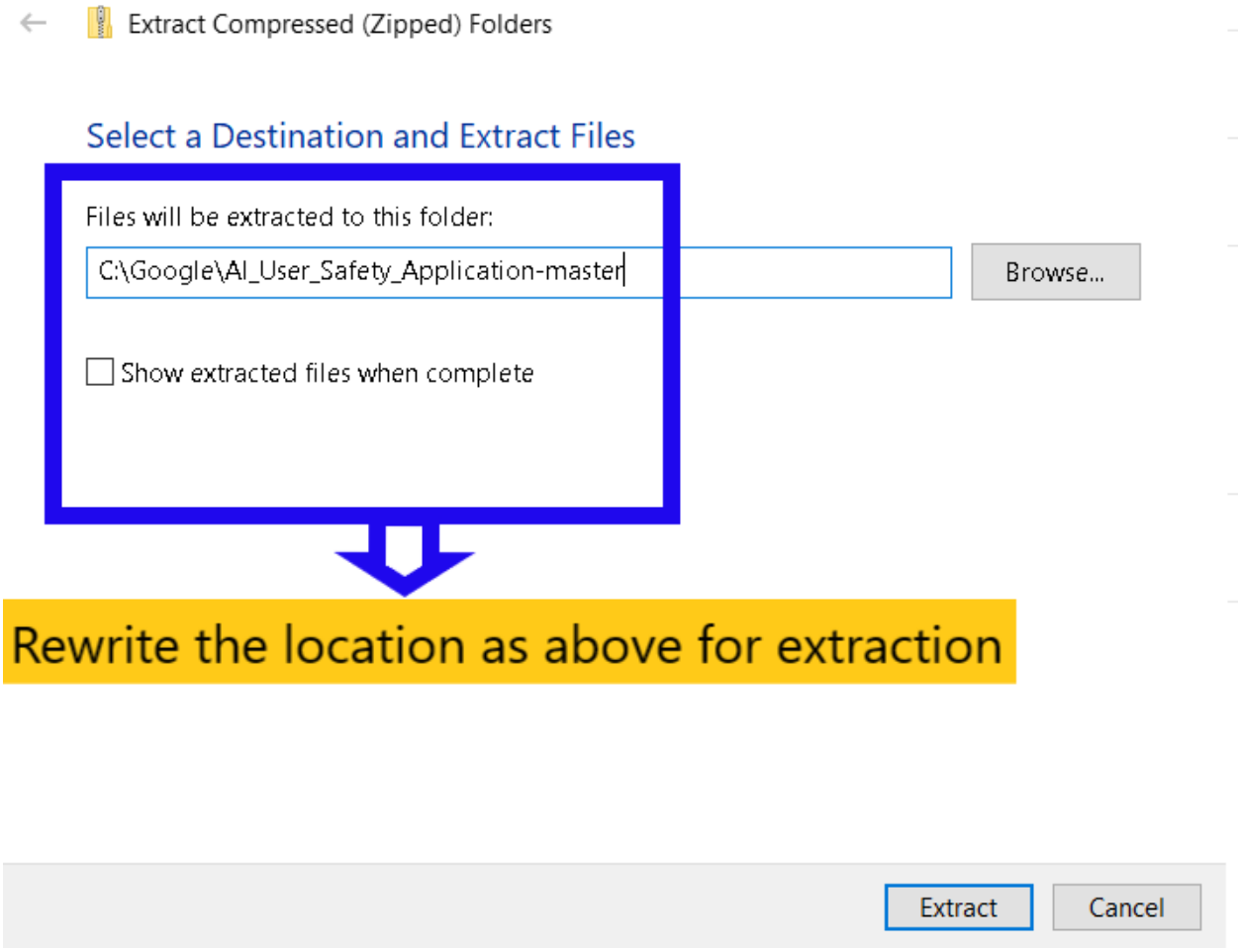


Figure 46: Extract the application

10.2.3. Step-3 Open Chrome and Go to Google Chrome tools and extension

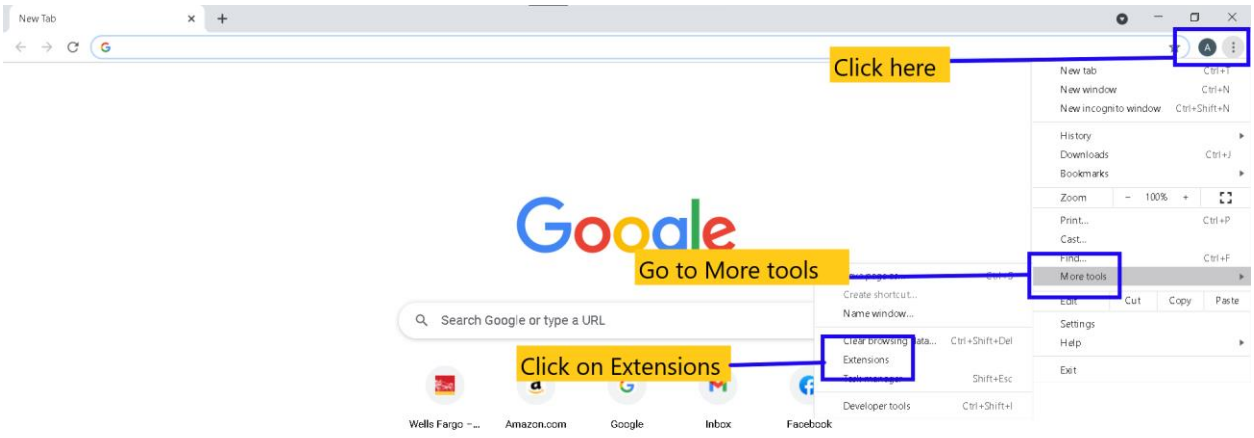


Figure 47: Chrome Extension loading source

10.2.4. Step-4 Set Chrome Developer mode active

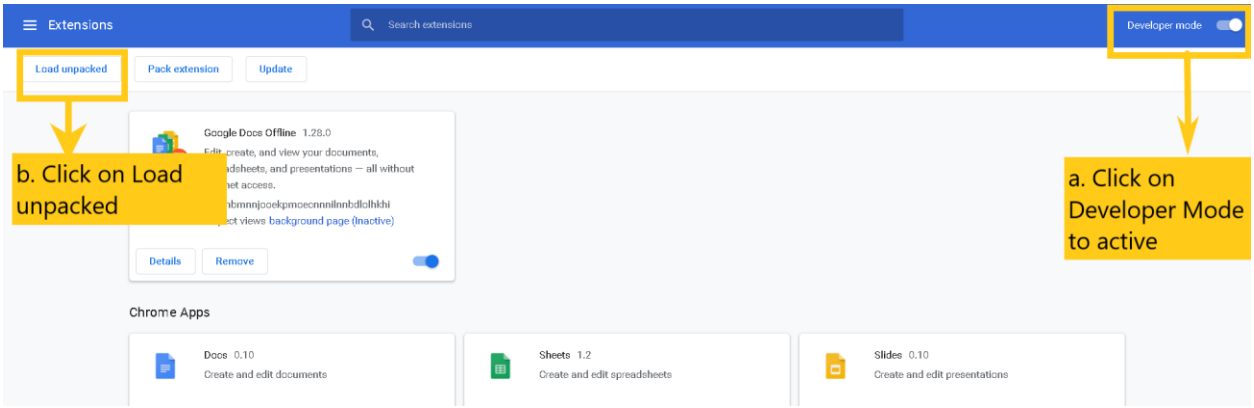


Figure 48: Activate Developer Mode

10.2.5. Step-5 Load the Application

Go to the location where the application has been extracted and click on the select folder

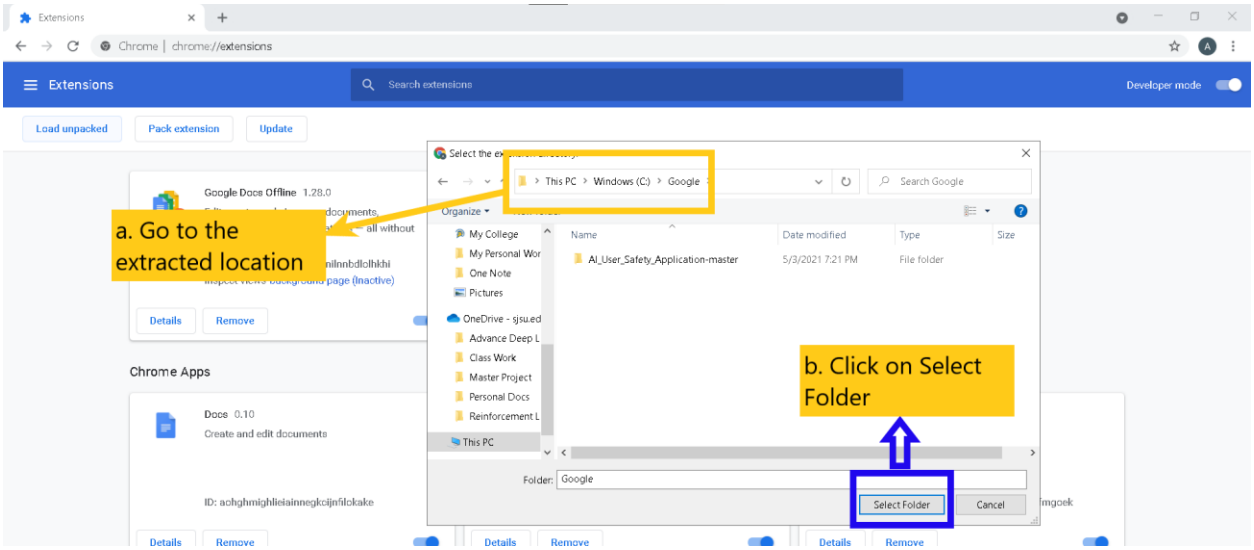


Figure 49: Load the GMV AI user Safety Application

10.2.6. Step-6 Application Unpacked

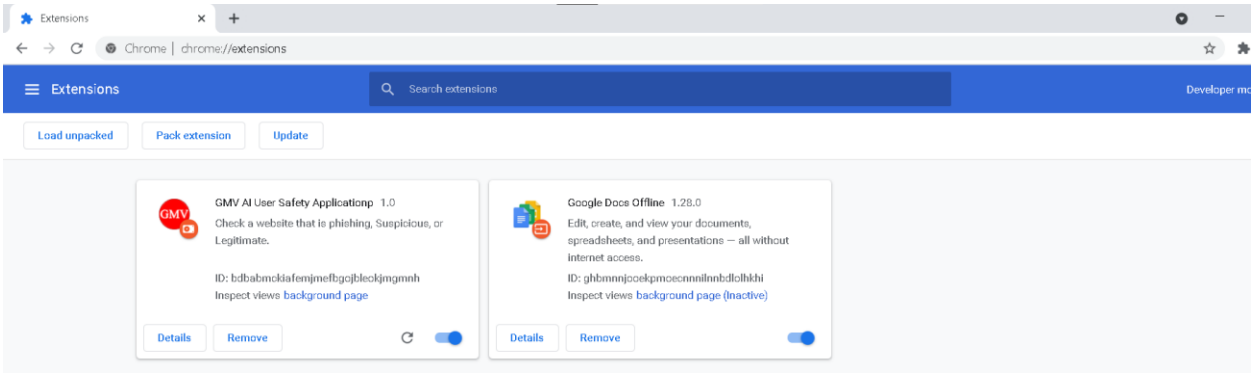


Figure 50: Unpack the GMV AI User safety Application

10.2.7. Step-7 Activate the plugin from Chrome

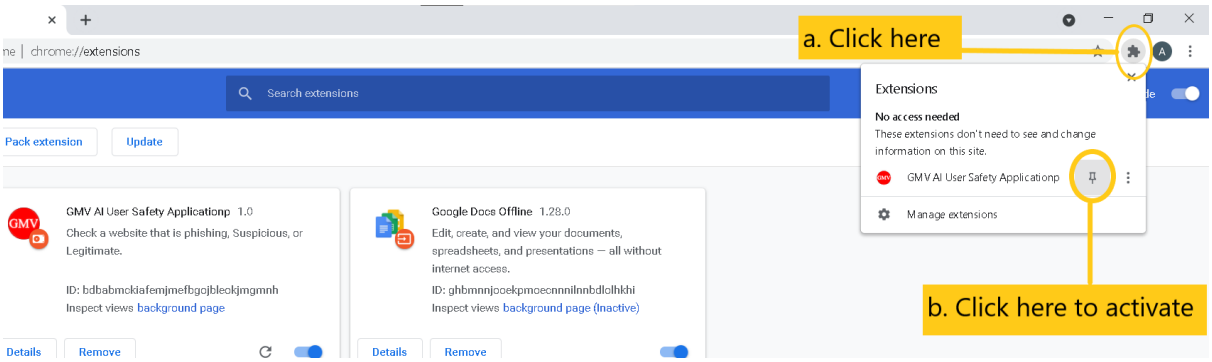


Figure 51: Activate the plugin in Google Chrome

10.2.8. Step-8 Go to Any webpage and click on application to see the result

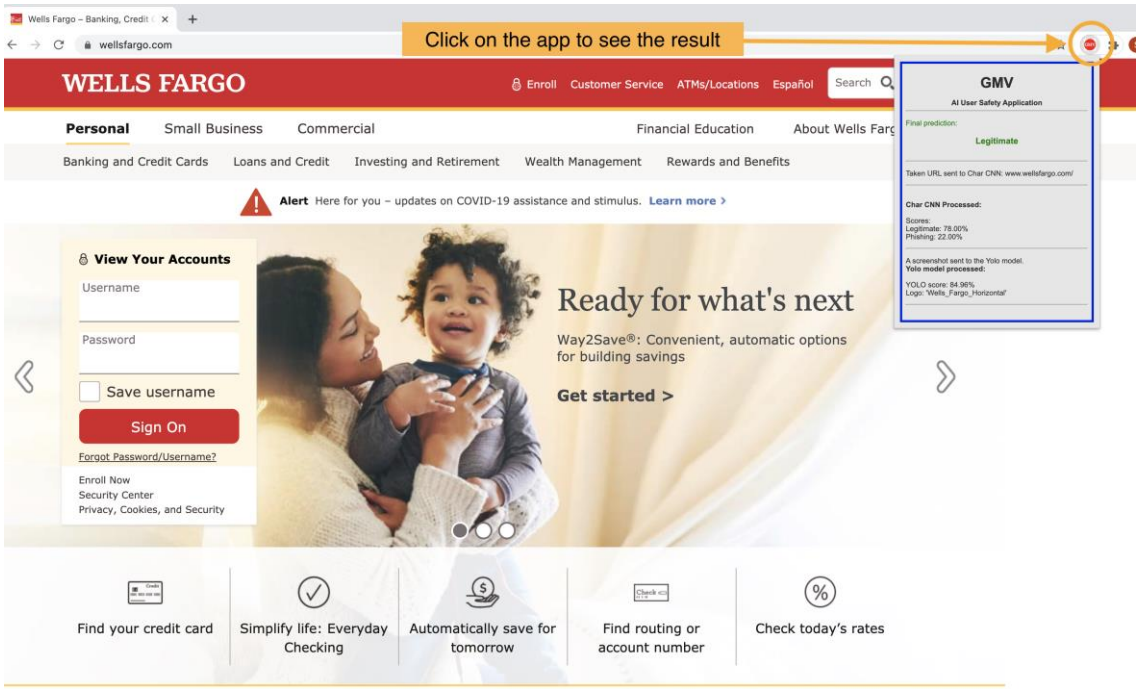


Figure 52: GMV AI User Safety Application installed

Chapter 11. Summary and Recommendations

11.1. Summary

The primary objective of the AI User Safety Application was to demonstrate an application that can detect phishing in a browser on a real-time basis. The approach was taken by us to validate a URL string and validate the Logo of the page has been a successful approach.

The overall idle memory footprint of the application is at least 60% less than that of commercial solutions available in the market. But in fairness, the other applications are doing not only doing phishing detection but also antivirus detection which increases their current footprint.

Our application is using pattern recognition based on Deep Learning models as against the other commercial solutions which are using database lookup which is a more than 100 MB lookup table in certain scenarios.

The benchmarked results show a maximum of 22% increase in greater accuracy in identifying phishing websites.

Below is the summary of AI User Safety Application performance:

Table 4: Summary of Benchmarked Performance Report

Resources	Bit Defender	Kaspersky	Avast	McAfee	GMV AI User Safety App
	Total Security 2021	Total Security 2021	Antivirus Service 2021 + Browser	Total Protection 2021	Version 12
Mean Idle CPU utilization	0%	0%	0%	0%	0%
Mean Idle Memory Utilization (MB)	96	111	142	120	31.7
Mean Prediction CPU Utilization	44%	54%	55%	42%	29.60%
Mean Time to predict (seconds)	11	8	12	14	3
Prediction Memory Utilization (MB)	128	144	224	138	97.8
Number of websites correctly categorized as phishing (120 links from PhishTank)	94%	93%	75%	81%	87%
AI model at the backend detection	No	No	No	No	Yes
Prediction mode	Local Database lookup	Local Database lookup	Online Database lookup	Online Database lookup	Pattern Recognition

The AI User Safety Application is faster in the prediction of phishing the webpage, it is more efficient and optimized than other commercial solutions present in the market.

11.2. Recommendations

Currently, the application can detect only phishing based on URL validation, Domain Validation, and Logo detection. The application scope will be increased with the following features added as part of the Google Chrome extension application:

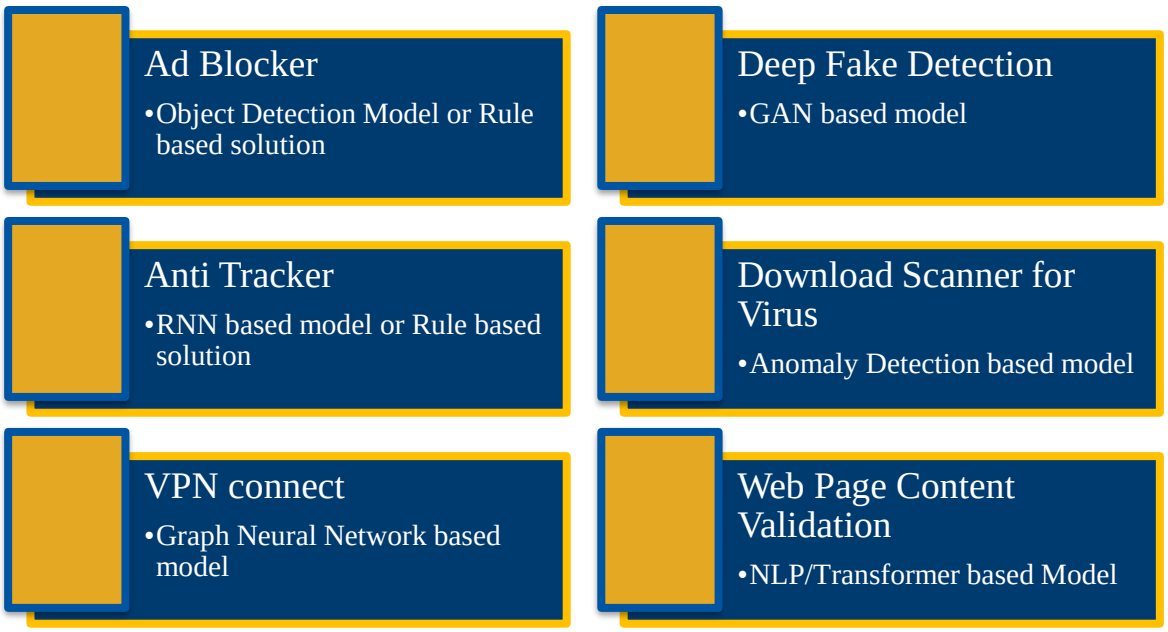


Figure 53: Features to be Integrated into Future Products

References

1. Y. Huang, Q. Yang, J. Qin, and W. Wen, "Phishing URL Detection via CNN and Attention-Based Hierarchical RNN," *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug. 2019.

2. H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," *arXiv.org*, 02-Mar-2018. [Online]. Available: https://arxiv.org/abs/1802.03162. [Accessed: 06-May-2021].

3. Desai, J. Jatakia, R. Naik, and N. Raul, "Malicious web content detection using machine leaning," *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, May 2017.

4. S. Bozkir and M. Aydos, "LogoSENSE: A companion HOG based logo detection scheme for phishing web page and E-mail brand recognition," *Computers & Security*, vol. 95, p. 101855, Apr. 2020.

5. S. Abdelnabi, K. Krombholz, and M. Fritz, "VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity," *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

6. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

7. Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 12993–13000, 2020.

8. Y. Kim, "Convolutional Neural Networks for Sentence Classification," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sep. 2014.

9. M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model Cards for Model Reporting," *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019.

10. Says: A. Says: O. Says: D. Says: T. Says: B. Says: H. M. Says: M. O. Says: 720p says: I. Says: Y. Says: and E. Says: "The beginner's guide to implementing YOLOv3 in TensorFlow 2.0 (part-2)," *Machine Learning Space*, 19-Mar-2020. [Online]. Available: https://machinelearningspace.com/yolov3-tensorflow-2-part-2/. [Accessed: 06-May-2021].

11. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, vol. 2. San Jose, California: O'Reilly, 2020.

12. R. Khandelwal, “Different IoU Losses for Faster and Accurate Object Detection,” *Medium*, 09-Mar-2021. [Online]. Available: https://medium.com/analytics-vidhya/different-iou-losses-for-faster-and-accurate-object-detection-3345781e0bf. [Accessed: 06-May-2021].

13. S. Yohanandan, “mAP (mean Average Precision) might confuse you!” *Medium*, 09-Jun-2020. [Online]. Available: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2?gi=da5373e8c4a6. [Accessed: 06-May-2021].

14. D. Ulevitch, “Join the fight against phishing,” *PhishTank*, 2006. [Online]. Available: http://phishtank.org/index.php. [Accessed: 06-May-2021].

15. *Majestic*. [Online]. Available: https://majestic.com/reports/majestic-million. [Accessed: 02-May-2021].

16. S. Marchal, J. Francois, R. State, and T. Engel, “PhishStorm: Detecting Phishing With Streaming Analytics,” *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 458–471, 2014.