Phishing Emails and Fake Logo Detection Using Browser Extension

A Project Report Submitted in complete Fulfillment of the Requirements for the Award of the Degree of

Bachelor of Technology

in

Computer Science and Engineering

By

I. Kavya Sai Sri	**(N200695)**
K. Teja Sri	**(N200375)**
M. Chandrika	**(N200150)**
Ch. Jyothirmai	**(N200320)**

Under the Guidance of

Mr. Kumar Anurupam
Assistant Professor
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Krishna District, Andhra Pradesh – 521202. July 2024

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
(A.P. Government Act 18 of 2008) RGUKT-NUZVID, Krishna Dist - 521202
Tel: 08656-235557 / 235150

CERTIFICATE OF COMPLETION

This is to certify that the work entitled **“Phishing Emails and Fake Logo Detection Using Browser Extension”** is the bonafide work of I. Kavya Sai Sri (N200695), K. Teja Sri (N200375), M. Chandrika (N200150), and Ch. Jyothirmai (N200320), carried out under my guidance and supervision for the Minor Project of the Bachelor of Technology in the Department of Computer Science and Engineering under RGUKT Nuzvid. This work was done during the academic session December 2024 – April 2025 under our guidance.

Mr. Kumar Anurupam
Assistant Professor
Department of CSE
RGUKT-Nuzvid

Mrs. S. Bhavani
Head of Department
Department of CSE
RGUKT-Nuzvid

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
(A.P. Government Act 18 of 2008) RGUKT-NUZVID, Krishna Dist - 521202
Tel: 08656-235557 / 235150

CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, **"Phishing Emails and Fake Logo Detection Using Browser Extension"** is the bonafide work of I. Kavya Sai Sri (N200695), K. Teja Sri (N200375), M. Chandrika (N200150), and Ch. Jyothirmai (N200320), and here by accord our approval of it as a study carried out and presented in a manner required for its acceptance in Minor Project of Bachelor of Technology for which it has been submitted. This approval does not necessarily endorse or accept swervy statement made, opinion expressed or conclusion drawn, as recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

Mr. Kumar Anurupam
Assistant Professor,
Department of CSE,
RGUKT Nuzvid.

Mr. Shiva Narayana
Assistant Professor
Department of CSE
RGUKT Nuzvid.

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
(A.P. Government Act 18 of 2008) RGUKT-NUZVID, Krishna Dist - 521202
Tel: 08656-235557 / 235150

DECLARATION

We I. Kavya Sai Sri (N200695), K. Teja Sri (N200375), M. Chandrika (N200150), and Ch. Jyothirmai (N200320), hereby declare that the project report entitled **"Phishing Emails and Fake Logo Detection Using Browser Extension"** done by us under the guidance of Mr. Kumar Anurupam is submitted in the complete fulfillment for the Minor Project of Bachelor of Technology in Computer Science and Engineering during the academic session December 2024 – April 2025 at RGUKT-Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Date: ___________
Place: ___________

ABSTRACT

Phishing attacks via email have been increasing day by day users are often getting mislead by fake urls and counterfeit logos posing a serious threat for sensitive information of users. This project presents a browser extension that helps in enhancing security within Gmail by detecting phishing emails which consist threatening URLs and fake logos that might lead users into traps. The extension works by scanning the entire content in the email , analyzing embedded urls and brand logos displayed within the email body. This extension integrates URL scanning techniques to classify them as safe and unsafe and uses a Trained Deep Learning model that compares the visible logos against the real brand logos and classifies them into real or fake. This extension reduces the rate of phishing combining cybersecurity with computer vision offering security to users.

TABLE OF CONTENTS

CHAPTER-1

1.1 INTRODUCTION

The Internet has become an indispensable part of our lives. Internet services are used daily by millions of people to communicate around the globe. However, it has also provided opportunities to perform malicious activities anonymously, such as phishing. Phishers attempt to deceive their victims through social engineering or by creating fake websites to steal sensitive information such as account IDs, usernames, and passwords from individuals and organizations.

Although many methods have been proposed to detect phishing websites, phishers continually evolve their techniques to bypass detection. Every day, a large volume of phishing emails floods users' inboxes. Traditional email filtering techniques have proven to be ineffective in curbing phishing attacks. The growing number of phishing emails poses serious challenges to both Internet Service Providers (ISPs) and end-users, threatening global security and the economy. Therefore, protecting users' mailboxes from phishing emails has become a critical issue today.

Solving this increasingly prevalent problem is not easy. Numerous approaches have been developed to filter phishing emails, yet a complete solution is still lacking. In our project, we address two major phishing threats:
1) Unsafe URLs, and
2) Impersonation of reputed organizations using fake logos as display pictures to deceive users.

We developed two deep learning models to detect such emails and integrated them into a browser extension for improved user interaction, enabling users to easily identify phishing emails.

1.2 OBJECTIVES

Objectives of Using a Chrome Extension for Phishing Email Detection

- **Early Detection and Prevention:**
  To identify phishing attempts at an early stage, allowing users to respond before they are tricked or compromised.

- **Mitigation of Harm:**
  By detecting and preventing phishing attacks, web extensions help individuals and organizations safeguard sensitive information, accounts, and systems from cybercrime.

- **User Interface and Experience:**
  The extension is designed to be user-friendly, providing clear alerts or warnings without disrupting the user experience.

- **Improved Security Posture:**
  Implementing phishing detection measures through web extensions strengthens overall cybersecurity by reducing the success rate of phishing attacks.

1.3 MOTIVATION

Although Gmail successfully filters most spam and phishing emails into the spam folder, there are still cases where phishing emails bypass these filters and land in the inbox. For instance, a phishing email was observed claiming that YouTube's terms and conditions had changed, urging the user to click a link, which ultimately led to a phishing attack.

Currently, users are expected to visit separate URL scanners to verify each suspicious link received in their Gmail. This is inconvenient and time-consuming. Hence, this browser extension was developed to address this issue and provide advanced, seamless protection within the Gmail environment.

1.4 Technical Details

PHISHING

Phishing is a cybercrime that involves tricking people into sharing sensitive information. It is the practice of sending fraudulent communications that appear to come from a legitimate and reputable source, usually through email and text messaging. The attacker's goal is to steal money, gain access to sensitive data and login information, or to install malware on the victim's device. Phishing works by luring a victim with legitimate-looking (but fraudulent) emails or other communication from a trusted (or sometimes seemingly desperate) sender who coaxes victims into providing confidential information often on what looks to be a convincingly legitimate website. Sometimes malware or ransomware is also downloaded onto the victim's computer.

Phishers frequently use tactics like fear, curiosity, a sense of urgency, and greed to compel recipients to open attachments or click on links. Phishing attacks are designed to appear to come from legitimate companies and individuals. Cybercriminals are continuously innovating and using increasingly sophisticated techniques, including spear phishing (an attack directed at a specific person or group) and other strategies.

Google Safe Browsing API

Google Safe Browsing API is used to identify and block unsafe or malicious URLs by checking them against Google's constantly updated lists of unsafe web resources.

ResNet-50 Model

In our project, the ResNet-50 model plays a crucial role in identifying fake logos in phishing emails, enhancing email security by providing users with real-time image authenticity verification. ResNet-50, a deep convolutional neural network, is used to identify counterfeit logos through deep learning. It learns intricate patterns from images and distinguishes between authentic and counterfeit logos. This model is mostly used in image classification, object detection, facial recognition, medical imageing.

1. Feature Extraction: ResNet-50, like other CNNs, excels at extracting important features from images. It learns to identify the unique characteristics of legitimate logos, such as color patterns, shapes, fonts, and fine details.
2. Classification: ResNet-50 is trained on a dataset of both genuine and counterfeit logos. During training, it learns to associate specific image features with each class (authentic or counterfeit).
3. Prediction and Probability Scores: When the model is given a new logo image, ResNet-50 analyzes its features and outputs a probability score for each class (authentic and counterfeit). This score indicates the model's confidence in the image being genuine or fake.
4. Detection: By setting a threshold on the probability scores, the system can automatically identify potential counterfeit logos.

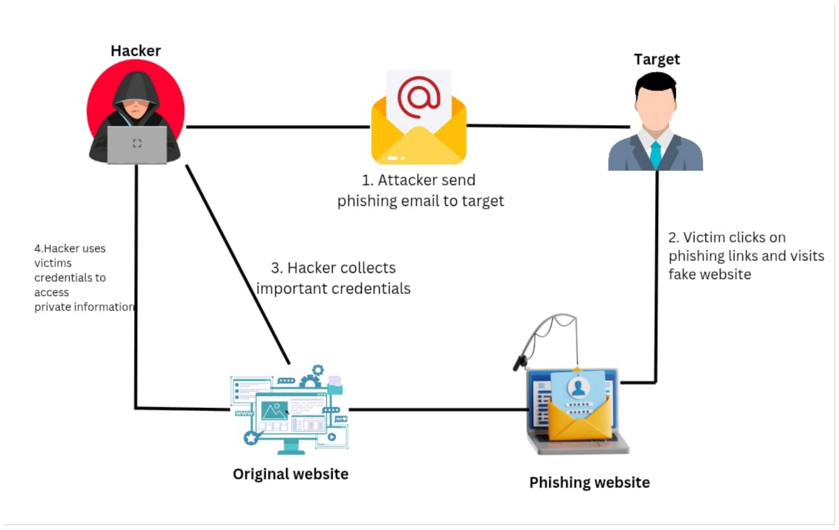

Hacker
Target
1. Attacker send phishing email to target
2. Victim clicks on phishing links and visits fake website
3. Hacker collects important credentials
4.Hacker uses victims credentials to access private information
Original website
Phishing website

Figure 1: Phishing reference

1.5 PHISHING CONSEQUENCES

Phishing attacks can lead to serious consequences, including..

- Identity Theft: Phishing attacks often aim to steal personal information like usernames, passwords, credit card details, and social security numbers, which can be used to open fraudulent accounts, take out loans, or commit other crimes in your name.
- Financial Loss: Successful phishing scams can lead to direct financial losses, such as unauthorized transactions, stolen funds, or the cost of recovering from identity theft.
- Compromised Personal Data: Phishing attacks can expose your personal data to unauthorized access, potentially leading to privacy violations, reputational damage, and emotional distress.
- Business Disruption: In a business context, phishing attacks can disrupt operations, lead to data breaches, and damage the company's reputation, potentially causing financial losses and loss of customer trust.
- Malware and Spyware: Phishing emails can contain malicious attachments or links that, when clicked, can install malware or spyware on your device, allowing hackers to gain control of your computer and steal information.
- Reputational Damage: A data breach caused by a phishing attack can severely damage an organization's reputation, leading to loss of customer trust and potential legal repercussions.

SYSTEM REQUIREMENTS

2.1 Hardware Requirements

- **Processor:** 64-bit, Core i5, 2.5 GHz minimum per core
- **RAM:** 8 GB or more
- **HDD:** 20 GB of available space or more
- **Display:** Dual XGA (1024 × 768) or higher resolution monitors
- **Keyboard:** A standard keyboard

2.2 Software Requirements

- **Operating System:** Windows OS 64-bit
- **Python:** Version 3.11.4
- **Python Libraries:**
  - **Flask:** For creating RESTful API endpoints
  - **Flask-CORS:** To enable Cross-Origin Resource Sharing
  - **torch:** PyTorch deep learning library
  - **torchvision:** Image transformation and pretrained model utilities
  - **Pillow (PIL):** For image processing and conversion
  - **NumPy:** Tools for numerical computations
  - **Pandas:** Data manipulation and analysis
  - **scikit-learn:** Metrics and evaluation tools
- **Development Platform:** Kaggle Notebook (for .ipynb files)
  Vscode for browser extension

2.3 Functional Requirements

1.Browser Extension

- **Email Content Submission:** The browser extension must send the email content to the backend API via a POST request, including any relevant metadata, such as email headers and body content.
- **Classification Results Display:** The extension must handle responses from the backend API and display the classification results correctly, including the classification label (safe or unsafe) and any relevant confidence scores or probabilities.

2.Model Training

- **Convolutional Neural Network (CNN) Training:** Train a CNN on the preprocessed images using a suitable optimizer (e.g., Adam, SGD) and loss function (e.g., cross-entropy).

3.Image Preprocessing

- **Image Transformation:** Transform images to fit the model requirements, including resizing, normalization, and data augmentation.
- **Edge Enhancement and Blur Correction:** Apply techniques such as edge enhancement, blur correction, and noise reduction to improve image quality.

4.Model Evaluation

- **Performance Metrics:** Evaluate model performance using relevant metrics, including:
  - Image classification accuracy
  - Classification precision and recall
  - Softmax probabilities of the classification

5.Email Classification

- **Classification:** Classify emails as safe or unsafe based on the observations, using the trained CNN model.
- **Confidence Scores:** Provide confidence scores or probabilities for each classification label.

6.URL Scanning via Google Safe Browsing API

- **URL Extraction:** Extract all URLs from the email body and metadata.
- **Threat Detection:** Use the Google Safe Browsing API to check each URL for phishing, malware, or other threats.
- **Safe Browsing Results Display:** Clearly indicate which URLs were flagged as unsafe in the browser extension UI.

2.4 Non-Functional Requirements

1.Scalability

- **Algorithm Optimization:** Optimize algorithms to handle large and varied image inputs efficiently using batching or parallel processing.
- **Concurrent User Support:** Ensure the backend can handle a large number of concurrent users without performance degradation.

2.Usability

- **User-Friendly UI:** Provide a simple and intuitive interface in the browser extension that requires minimal training.
- **Extension Installation:** Ensure the extension is easy to install on Chrome and optionally on other browsers.

3.Compatibility

- **Browser Support:** Ensure compatibility with Chrome, Firefox, and Edge.
- **Operating System Support:** Ensure the system works on Windows, macOS, and Linux environments.

4.Maintainability

- **Modular Code:** Follow clean architecture principles and use modular code for easier maintenance and scalability.
- **Error Handling:** Implement robust error handling with proper logging and meaningful error messages for both frontend and backend.

5.Performance

- **Response Time:** The system must respond within 2–3 seconds of email submission, including image classification and URL scanning.
- **Fallback Mechanism:** If Google Safe Browsing API is unreachable, proceed with classification and notify the user about the partial result.
- **Caching Safe URLs:** Cache previously verified safe URLs temporarily to reduce redundant API calls and improve response time.

6.Security and Accuracy

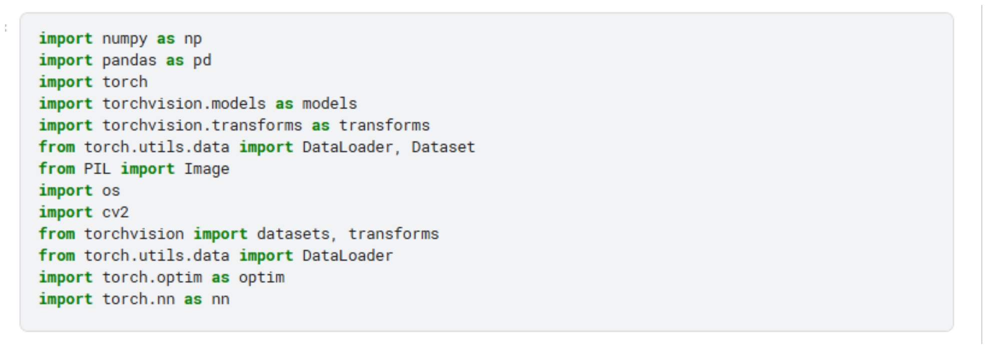
- **Secure API Usage:** Store the Google Safe Browsing API key securely and avoid exposing it to the frontend.
- **Data Privacy:** Ensure email content and URLs are processed securely, without storing any personal user data.

SYSTEM DESIGN

3.1 Environment Setup

We have used Kaggle environment foe the development of Fake logo detection model. we have made sure all the required libraries are installed.

- `numpy`
- `pandas`
- `torch`
- `torchvision.models`
- `torchvision.transforms`
- `torch.utils.data.DataLoader`
- `torch.utils.data.Dataset`
- `PIL.Image`
- `os`
- `cv2`
- `torch.optim`
- `torch.nn`



```
import numpy as np
import pandas as pd
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
from PIL import Image
import os
import cv2
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.nn as nn
```

Figure 2: Required libraries are imported

3.2 Dataset

We loaded the dataset in the Kaggle environment.
We created our dataset from scratch by manually collecting a diverse set of genuine and fake logos of various brands from the internet. After gathering the images, we organized them into appropriate categories based on their authenticity. The entire dataset was then uploaded to the Kaggle input folder, making it easily accessible for both training and testing purposes throughout the project.
The dataset is divided into two main subsets:

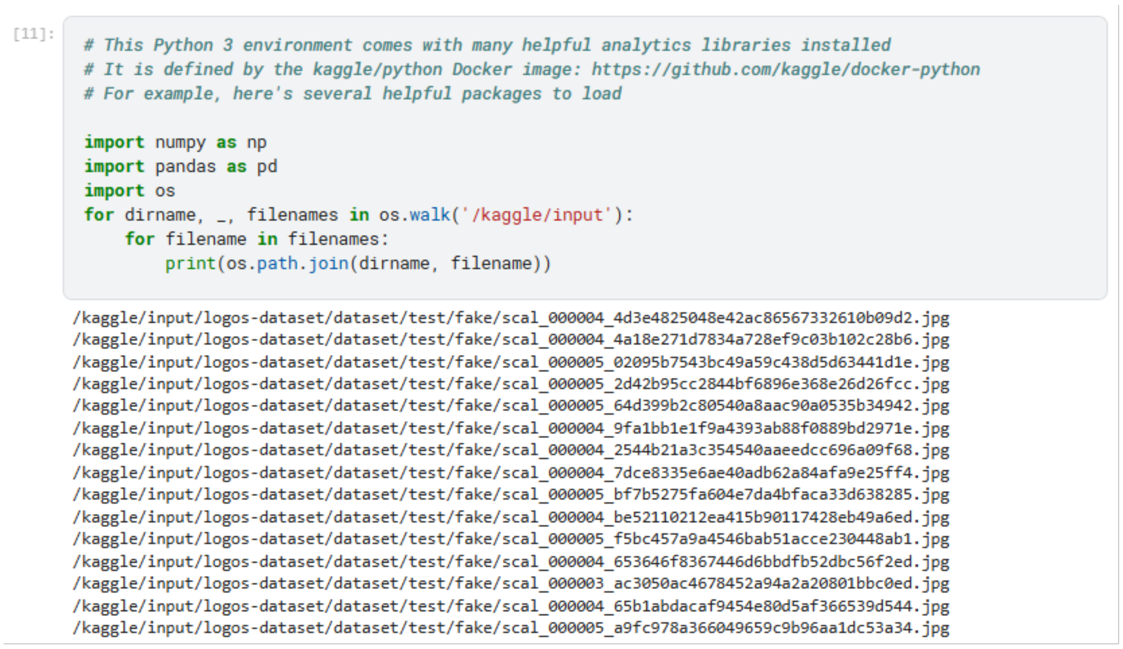
Training Dataset:
This subset is used to train the ResNet-50 model. It contains a total of **970 images**, with:

- **464 images of genuine brand logos**
- **416 images of fake logos**

Testing Dataset:
This subset is used to evaluate the performance of the trained model. It consists of **322 images**, including:

- **162 images of genuine brand logos**
- **160 images of fake logos**

The dataset we used is in the required format and is divided into training and testing subsets. The test dataset consists of two separate unlabelled folders: one for fake images and one for genuine images. By carefully organizing and balancing the number of images in each category, we ensured that the model receives a fair representation of both classes during training and testing. This helps improve the model's ability to distinguish between genuine and counterfeit logos, thereby increasing its reliability and accuracy in real-world scenarios.



```
[11]:
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_4d3e4825048e42ac8656733261 0b09d2.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_4a18e271d7834a728ef9c03b102c28b6.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_02095b7543bc49a59c438d5d63441d1e.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_2d42b95cc2844bf6896e368e26d26fcc.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_64d399b2c80540a8aac90a0535b34942.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_9fa1bb1e1f9a4393ab88f0889bd2971e.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_2544b21a3c354540aaeedcc696a09f68.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_7dce8335e6ae40adb62a84afa9e25ff4.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_bf7b5275fa604e7da4bfaca33d638285.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_be52110212ea415b90117428eb49a6ed.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_f5bc457a9a4546bab51acce230448ab1.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_653646f8367446d6bbdfb52dbc56f2ed.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000003_ac3050ac4678452a94a2a20801bbc0ed.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000004_65b1abdacaf9454e80d5af366539d544.jpg
/kaggle/input/logos-dataset/dataset/test/fake/scal_000005_a9fc978a366049659c9b96aa1dc53a34.jpg
```

Figure 3: Loading dataset

Similarly, the training dataset contains two separate unlabelled folders for fake and genuine images. We labelled this data, performed preprocessing, and trained a ResNet-50 model on it.

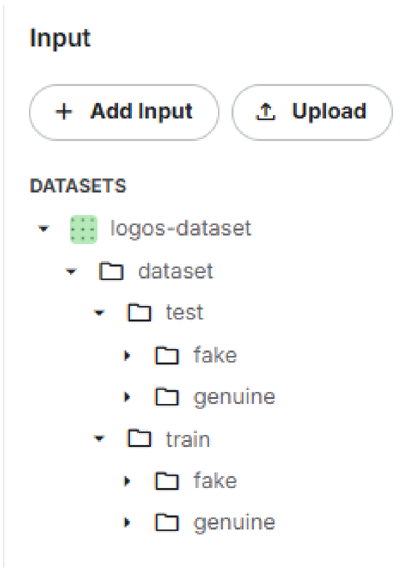
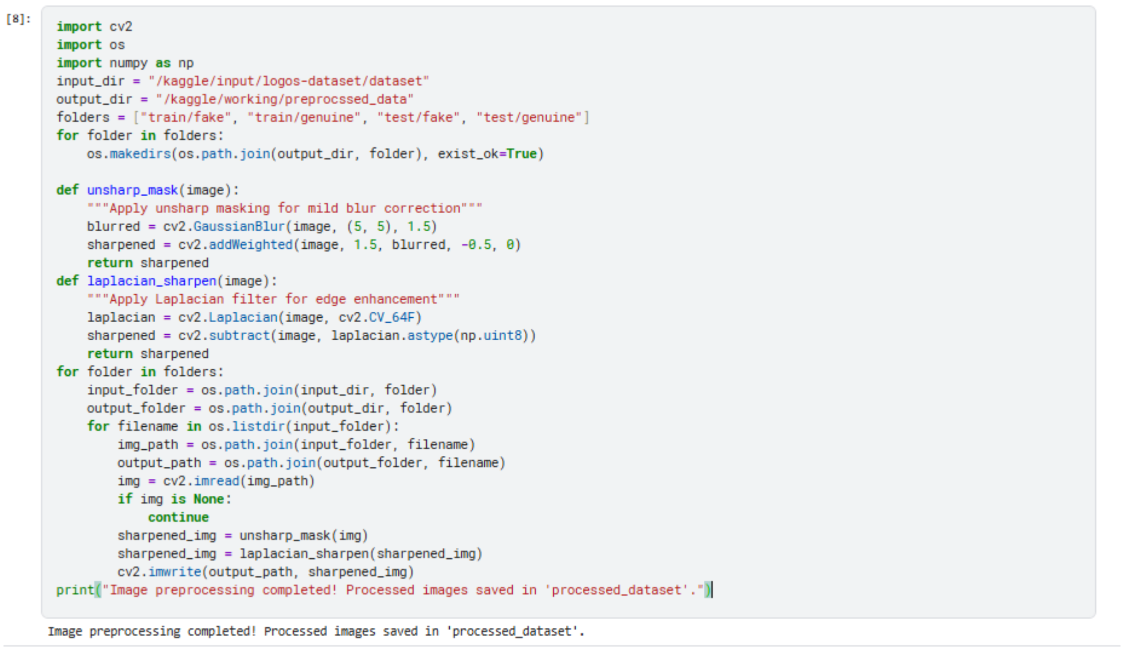


Figure 4: Dataset Structure

3.3 Model Implementation

- Image Preprocessing

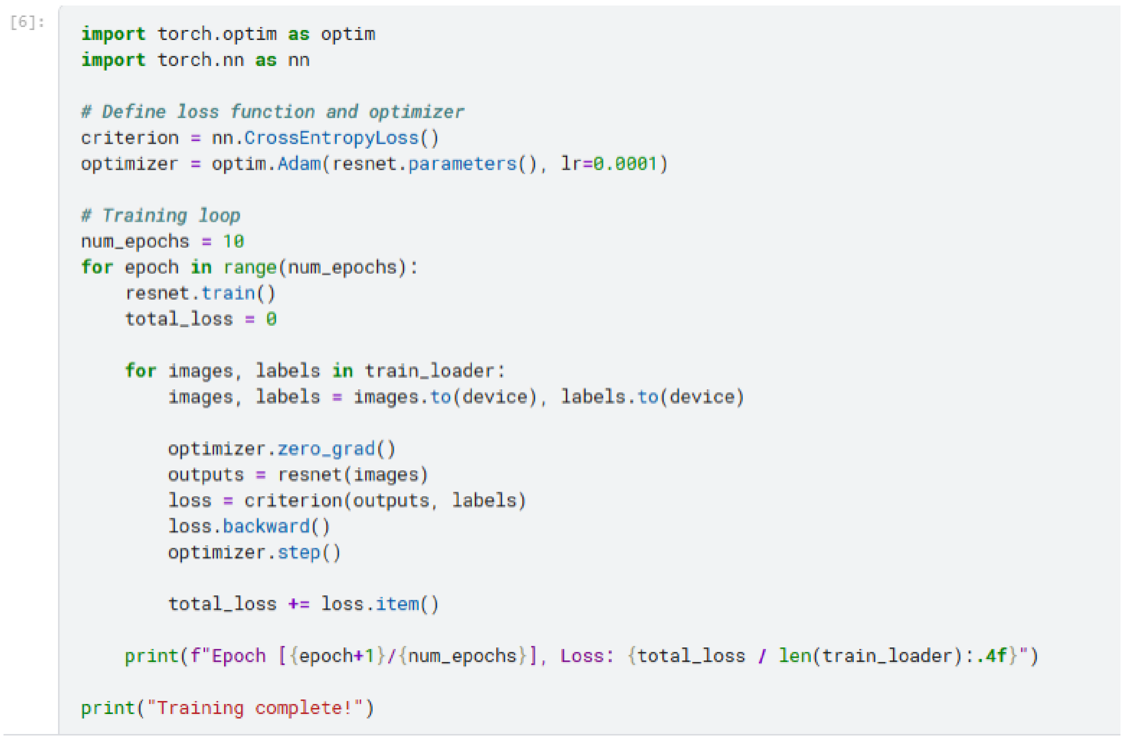


```
[8]:
import cv2
import os
import numpy as np
input_dir = "/kaggle/input/logos-dataset/dataset"
output_dir = "/kaggle/working/preprocssed_data"
folders = ["train/fake", "train/genuine", "test/fake", "test/genuine"]
for folder in folders:
    os.makedirs(os.path.join(output_dir, folder), exist_ok=True)

def unsharp_mask(image):
    """Apply unsharp masking for mild blur correction"""
    blurred = cv2.GaussianBlur(image, (5, 5), 1.5)
    sharpened = cv2.addWeighted(image, 1.5, blurred, -0.5, 0)
    return sharpened
def laplacian_sharpen(image):
    """Apply Laplacian filter for edge enhancement"""
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    sharpened = cv2.subtract(image, laplacian.astype(np.uint8))
    return sharpened
for folder in folders:
    input_folder = os.path.join(input_dir, folder)
    output_folder = os.path.join(output_dir, folder)
    for filename in os.listdir(input_folder):
        img_path = os.path.join(input_folder, filename)
        output_path = os.path.join(output_folder, filename)
        img = cv2.imread(img_path)
        if img is None:
            continue
        sharpened_img = unsharp_mask(img)
        sharpened_img = laplacian_sharpen(sharpened_img)
        cv2.imwrite(output_path, sharpened_img)
print("Image preprocessing completed! Processed images saved in 'processed_dataset'.")
```

```
Image preprocessing completed! Processed images saved in 'processed_dataset'.
```

Figure 5: Image preprocessing step

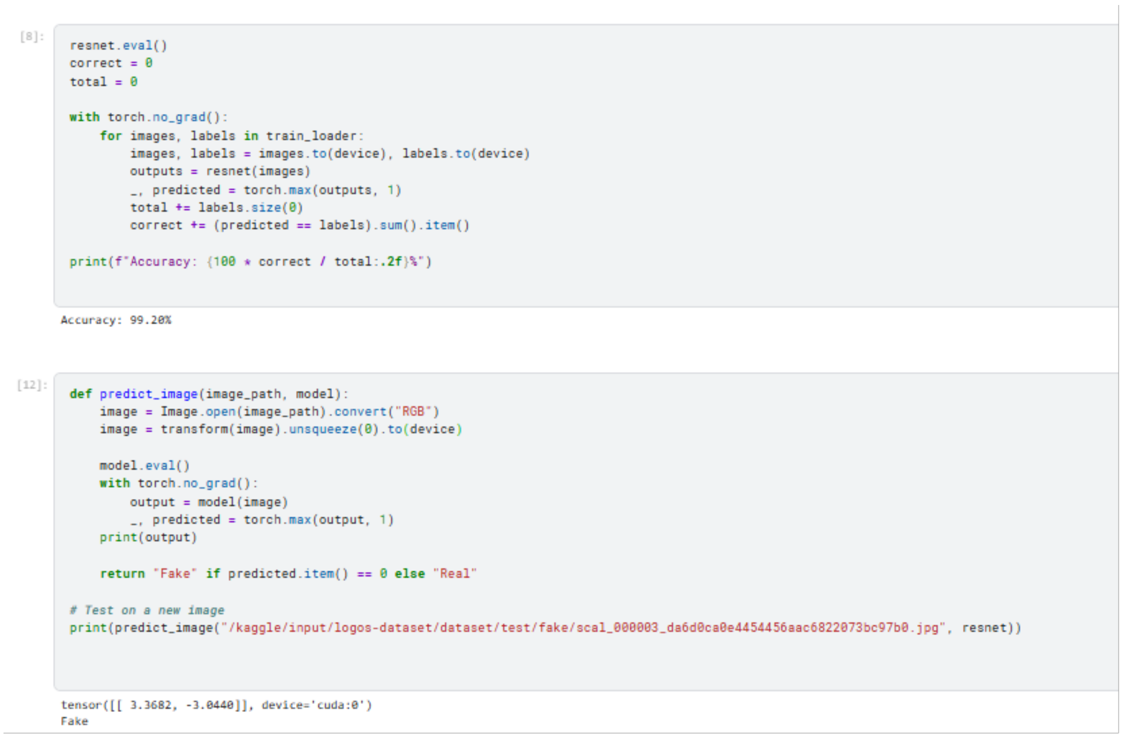
- Model Training



```
[6]:
import torch.optim as optim
import torch.nn as nn

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet.parameters(), lr=0.0001)

# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    resnet.train()
    total_loss = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = resnet(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss / len(train_loader):.4f}")

print("Training complete!")
```

Figure 6: Model training process

- Model Prediction



```
[8]:
resnet.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = resnet(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy: {100 * correct / total:.2f}%")
```

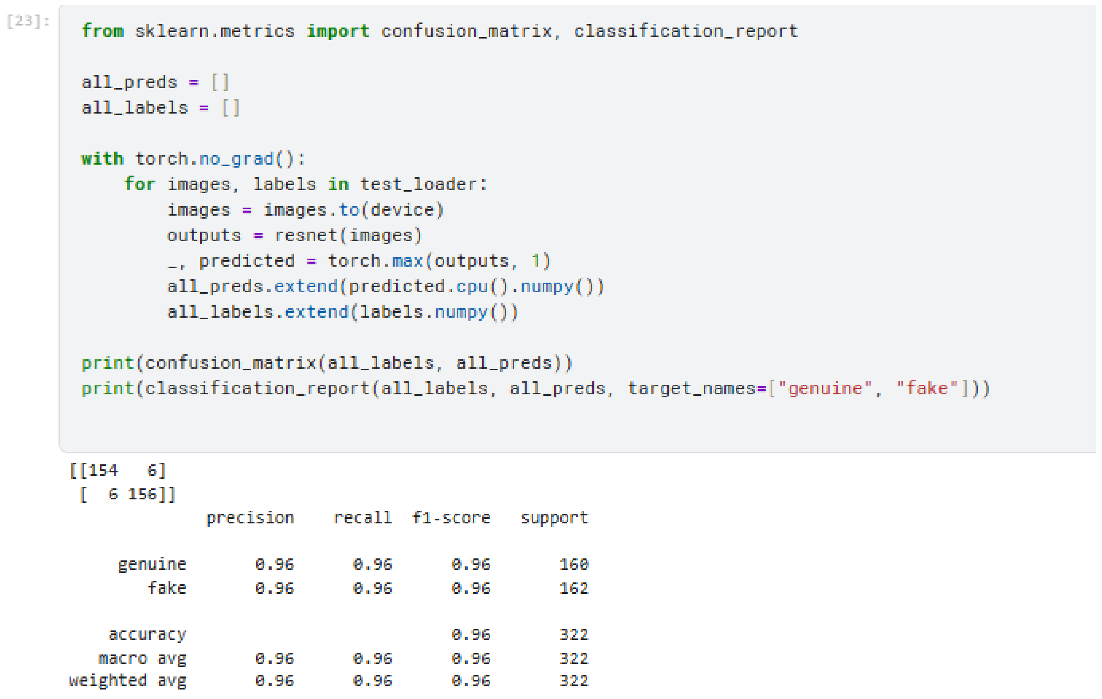
```
Accuracy: 99.20%
```

```
[12]:
def predict_image(image_path, model):
    image = Image.open(image_path).convert("RGB")
    image = transform(image).unsqueeze(0).to(device)

    model.eval()
    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)
    print(output)

    return "Fake" if predicted.item() == 0 else "Real"

# Test on a new image
print(predict_image("/kaggle/input/logos-dataset/dataset/test/fake/scal_000003_da6d0ca0e4454456aac6822073bc97b0.jpg", resnet))
```

```
tensor([[ 3.3682, -3.0440]], device='cuda:0')
Fake
```

Figure 7: Model prediction process

- Model Evaluation



```
[23]:
from sklearn.metrics import confusion_matrix, classification_report

all_preds = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = resnet(images)
        _, predicted = torch.max(outputs, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.numpy())

print(confusion_matrix(all_labels, all_preds))
print(classification_report(all_labels, all_preds, target_names=["genuine", "fake"]))
```

```
[[154   6]
 [  6 156]]
              precision    recall  f1-score   support

     genuine       0.96      0.96      0.96       160
        fake       0.96      0.96      0.96       162

    accuracy                           0.96       322
   macro avg       0.96      0.96      0.96       322
weighted avg       0.96      0.96      0.96       322
```

Figure 8: Model evaluation results

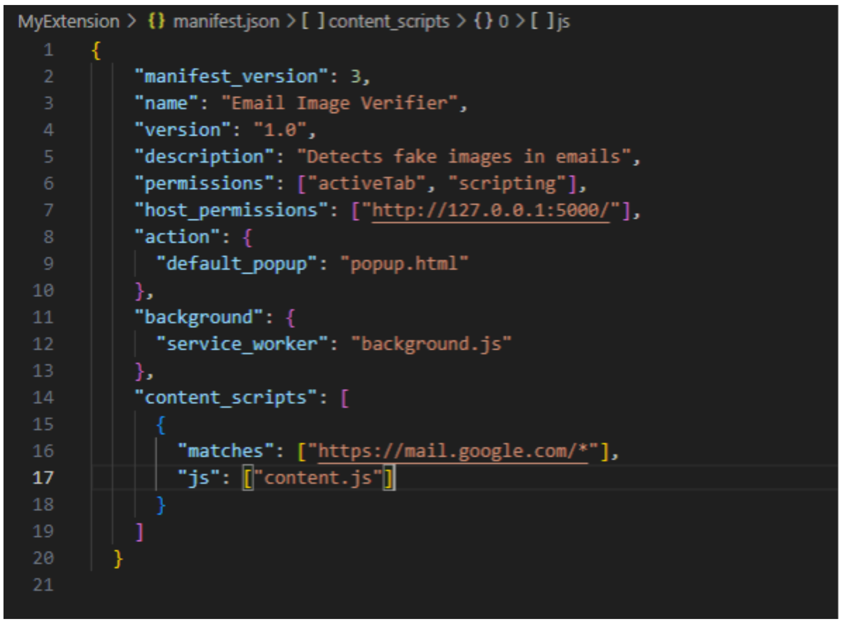
3.4 Extension Implementation

Chrome extension when activated and loaded it shows a popup on the web page, which can be included with our own requirements and design. Basically these Chrome or any browser extensions are made with simple HTML, CSS, JavaScript and Json files.
Any Browser extension contains some similar files names Manifest.json,popup.js, popup.html, content.js, background.js

Components and Their Roles

- **manifest.json** - This file defines the browser metadata, permissions, and the extension's blueprint.
- **content.js** - This allows the extension to read, modify, and analyze the webpage elements.
- **popup.html** - This file creates the user interface for the extension, which is what the user sees when they open the extension.
- **popup.js** - This file handles the functionality and interactivity of the extension.
- **app.py** - This handles the Python scripts that run a Flask server and processes the images for model prediction.

- Manifest.json

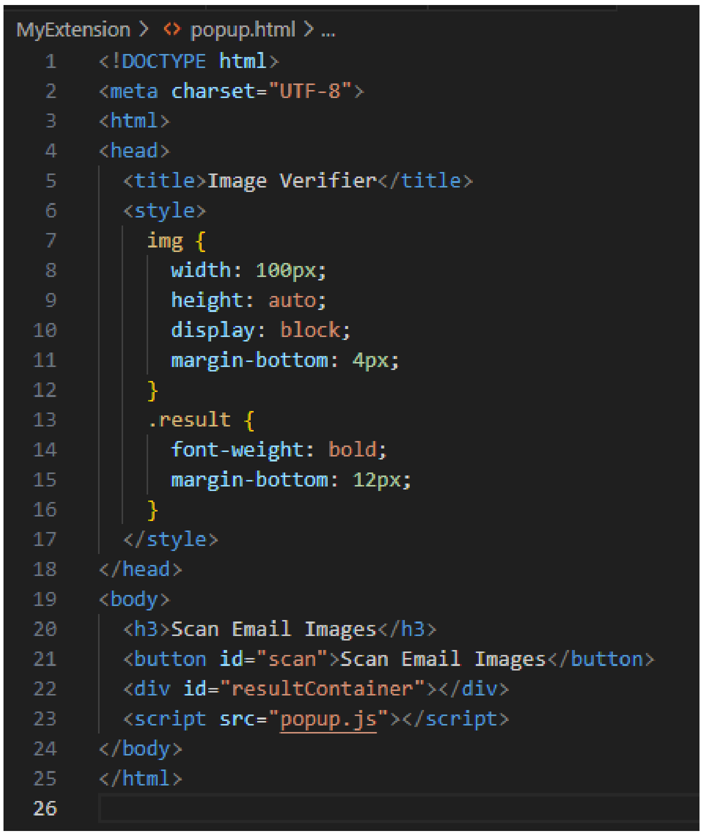


MyExtension > {} manifest.json > [ ] content_scripts > {} 0 > [ ] js

```json
{
    "manifest_version": 3,
    "name": "Email Image Verifier",
    "version": "1.0",
    "description": "Detects fake images in emails",
    "permissions": ["activeTab", "scripting"],
    "host_permissions": ["http://127.0.0.1:5000/"],
    "action": {
      "default_popup": "popup.html"
    },
    "background": {
      "service_worker": "background.js"
    },
    "content_scripts": [
      {
        "matches": ["https://mail.google.com/*"],
        "js": ["content.js"]
      }
    ]
  }

```

Figure 9: Manifest.json file structure

- popup.html

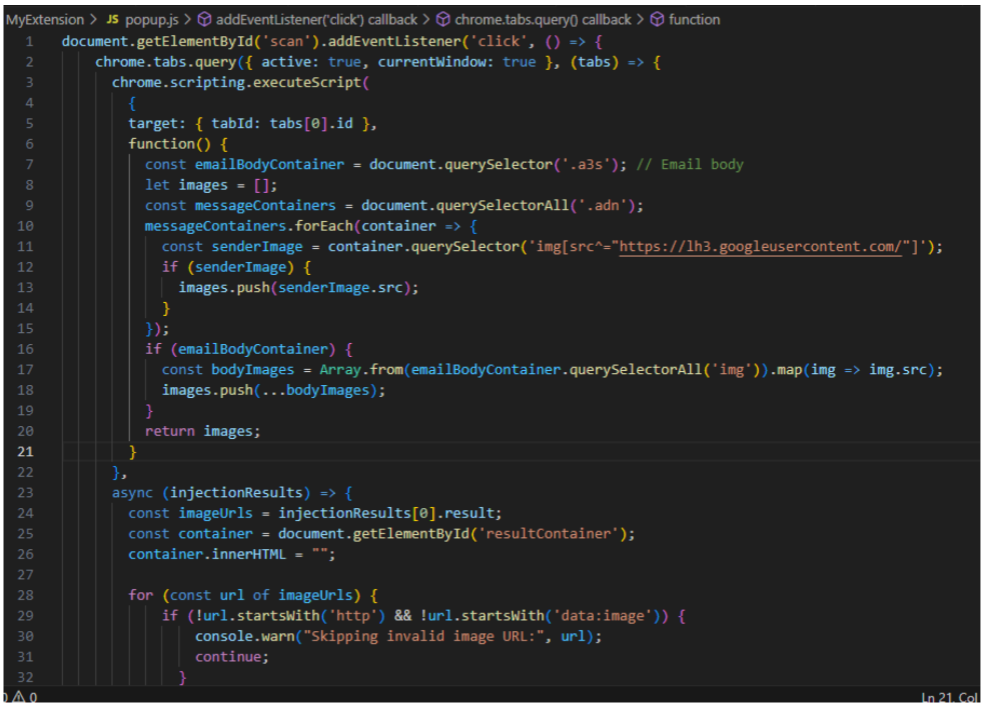


MyExtension > popup.html > ...

```html
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<head>
  <title>Image Verifier</title>
  <style>
    img {
      width: 100px;
      height: auto;
      display: block;
      margin-bottom: 4px;
    }
    .result {
      font-weight: bold;
      margin-bottom: 12px;
    }
  </style>
</head>
<body>
  <h3>Scan Email Images</h3>
  <button id="scan">Scan Email Images</button>
  <div id="resultContainer"></div>
  <script src="popup.js"></script>
</body>
</html>

```

Figure 10: User Interface Layout

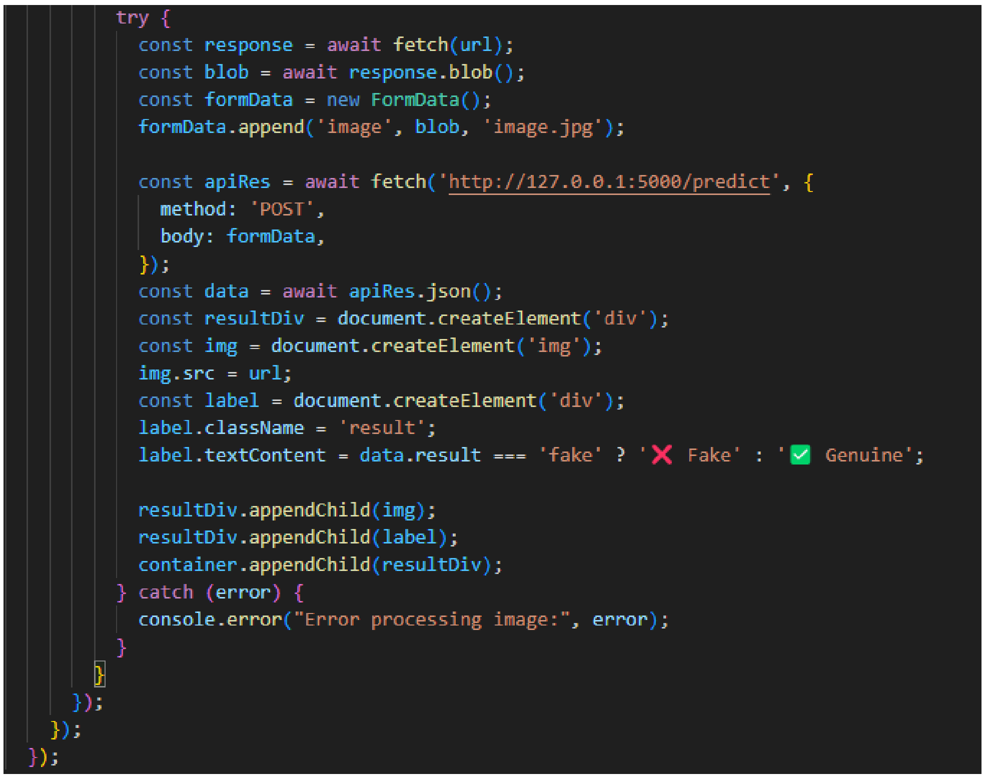
- popup.js



MyExtension > JS popup.js > addEventListener('click') callback > chrome.tabs.query() callback > function

```
document.getElementById('scan').addEventListener('click', () => {
    chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
      chrome.scripting.executeScript(
        {
        target: { tabId: tabs[0].id },
        function() {
          const emailBodyContainer = document.querySelector('.a3s'); // Email body
          let images = [];
          const messageContainers = document.querySelectorAll('.adn');
          messageContainers.forEach(container => {
            const senderImage = container.querySelector('img[src^="https://lh3.googleusercontent.com/"]');
            if (senderImage) {
              images.push(senderImage.src);
            }
          });
          if (emailBodyContainer) {
            const bodyImages = Array.from(emailBodyContainer.querySelectorAll('img')).map(img => img.src);
            images.push(...bodyImages);
          }
          return images;
        }
      },
      async (injectionResults) => {
        const imageUrls = injectionResults[0].result;
        const container = document.getElementById('resultContainer');
        container.innerHTML = "";

        for (const url of imageUrls) {
            if (!url.startsWith('http') && !url.startsWith('data:image')) {
                console.warn("Skipping invalid image URL:", url);
                continue;
              }
```

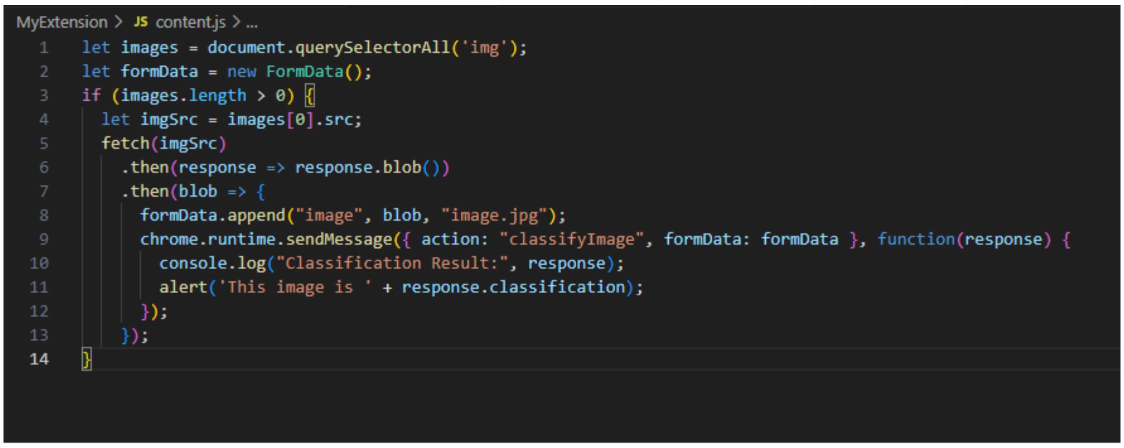
Figure 11: popup.js: Part 1



```
            try {
              const response = await fetch(url);
              const blob = await response.blob();
              const formData = new FormData();
              formData.append('image', blob, 'image.jpg');

              const apiRes = await fetch('http://127.0.0.1:5000/predict', {
                method: 'POST',
                body: formData,
              });
              const data = await apiRes.json();
              const resultDiv = document.createElement('div');
              const img = document.createElement('img');
              img.src = url;
              const label = document.createElement('div');
              label.className = 'result';
              label.textContent = data.result === 'fake' ? '❌ Fake' : '✅ Genuine';

              resultDiv.appendChild(img);
              resultDiv.appendChild(label);
              container.appendChild(resultDiv);
            } catch (error) {
              console.error("Error processing image:", error);
            }
          }
        });
      });
    });
```

Figure 12: popup.js: Part 2

- content.js

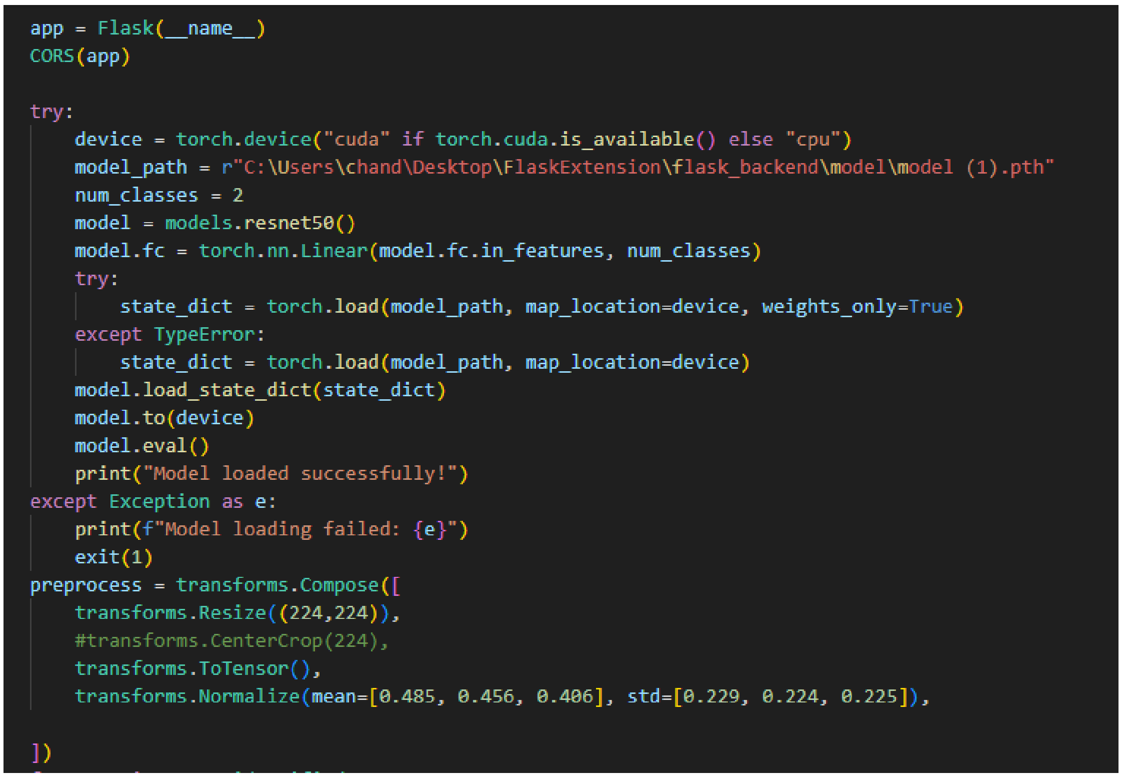


MyExtension > JS content.js > ...

```javascript
let images = document.querySelectorAll('img');
let formData = new FormData();
if (images.length > 0) {
  let imgSrc = images[0].src;
  fetch(imgSrc)
    .then(response => response.blob())
    .then(blob => {
      formData.append("image", blob, "image.jpg");
      chrome.runtime.sendMessage({ action: "classifyImage", formData: formData }, function(response) {
        console.log("Classification Result:", response);
        alert('This image is ' + response.classification);
      });
    });
}
```

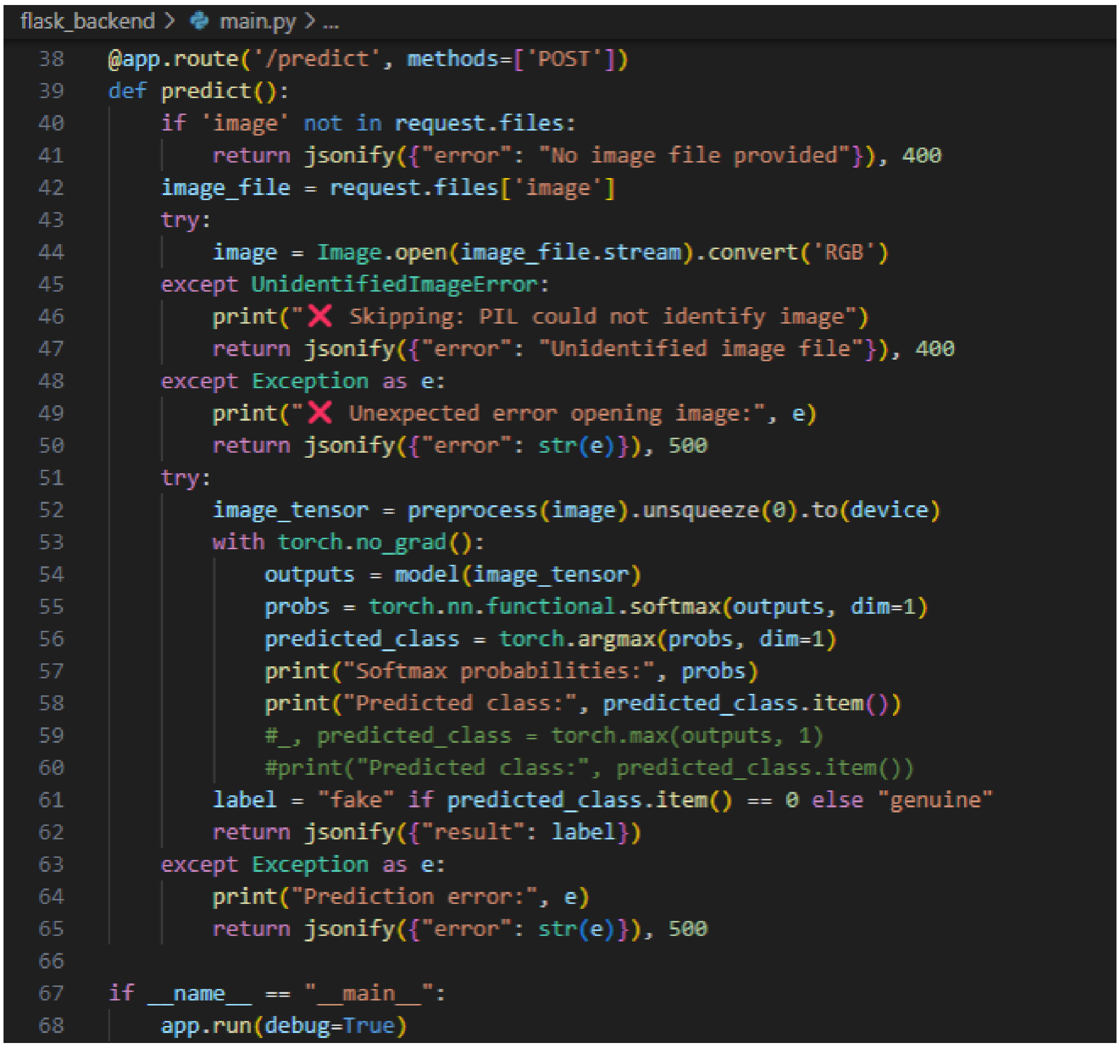
Figure 13: content.js Script handling webpage elements

- app.py



```python
app = Flask(__name__)
CORS(app)

try:
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model_path = r"C:\Users\chand\Desktop\FlaskExtension\flask_backend\model\model (1).pth"
    num_classes = 2
    model = models.resnet50()
    model.fc = torch.nn.Linear(model.fc.in_features, num_classes)
    try:
        state_dict = torch.load(model_path, map_location=device, weights_only=True)
    except TypeError:
        state_dict = torch.load(model_path, map_location=device)
    model.load_state_dict(state_dict)
    model.to(device)
    model.eval()
    print("Model loaded successfully!")
except Exception as e:
    print(f"Model loading failed: {e}")
    exit(1)
preprocess = transforms.Compose([
    transforms.Resize((224,224)),
    #transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])
```

Figure 14: app.py Flask Server Code: Part 1

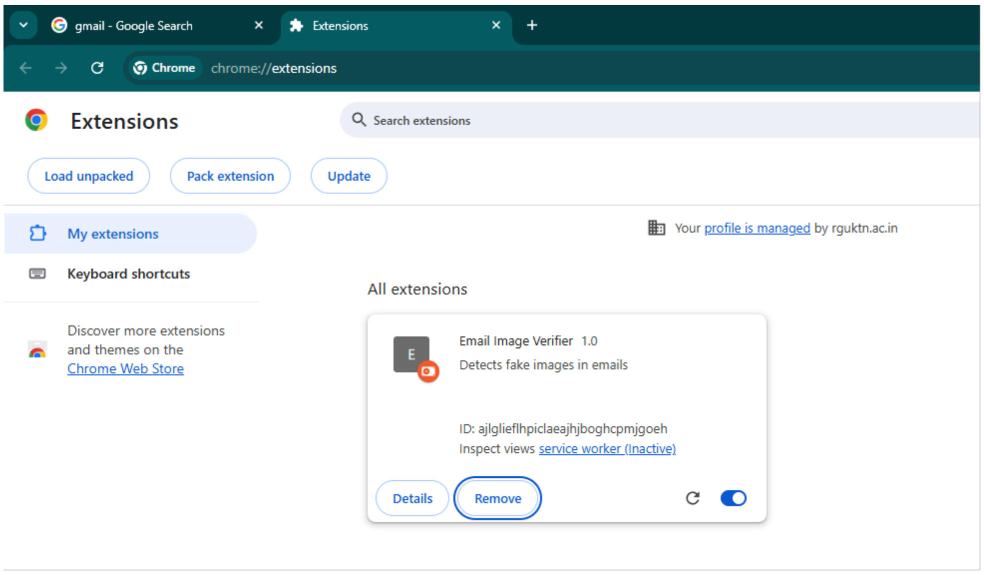

flask_backend > main.py > ...

```python
@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({"error": "No image file provided"}), 400
    image_file = request.files['image']
    try:
        image = Image.open(image_file.stream).convert('RGB')
    except UnidentifiedImageError:
        print("❌ Skipping: PIL could not identify image")
        return jsonify({"error": "Unidentified image file"}), 400
    except Exception as e:
        print("❌ Unexpected error opening image:", e)
        return jsonify({"error": str(e)}), 500
    try:
        image_tensor = preprocess(image).unsqueeze(0).to(device)
        with torch.no_grad():
            outputs = model(image_tensor)
            probs = torch.nn.functional.softmax(outputs, dim=1)
            predicted_class = torch.argmax(probs, dim=1)
            print("Softmax probabilities:", probs)
            print("Predicted class:", predicted_class.item())
            #_, predicted_class = torch.max(outputs, 1)
            #print("Predicted class:", predicted_class.item())
        label = "fake" if predicted_class.item() == 0 else "genuine"
        return jsonify({"result": label})
    except Exception as e:
        print("Prediction error:", e)
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    app.run(debug=True)
```

Figure 15: app.py Flask Server Code: Part 2

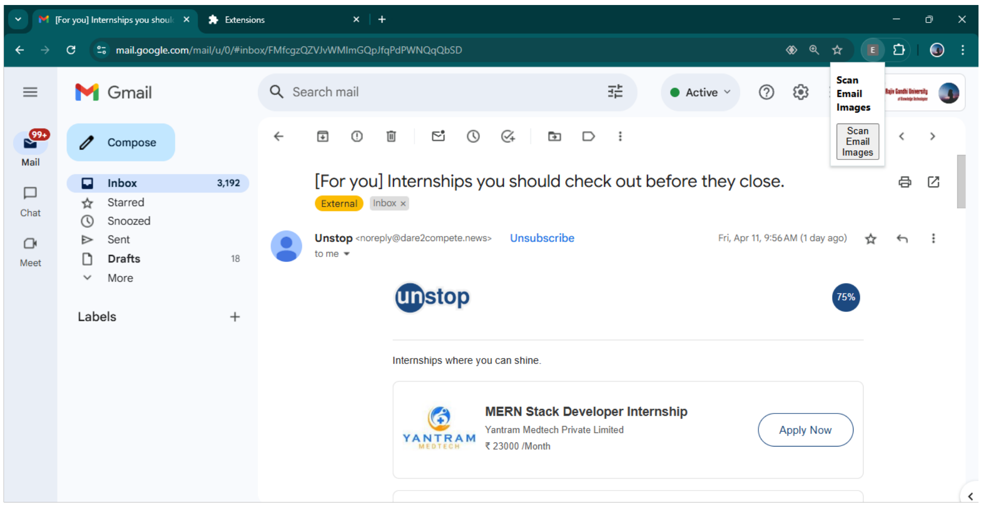
3.5 Input-Output

Input



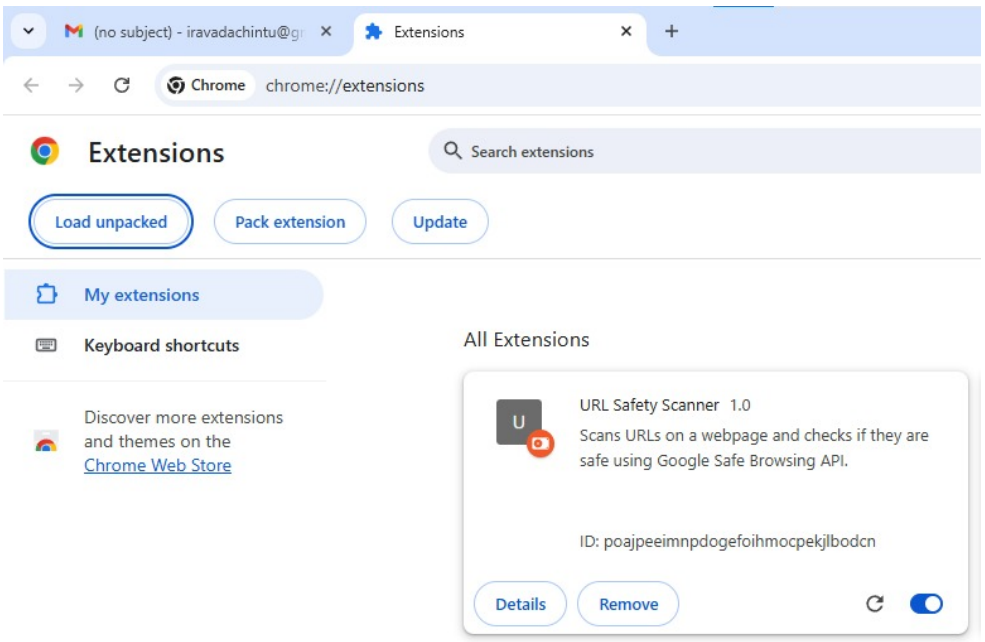
Email Image Verifier 1.0
Detects fake images in emails
ID: ajlglieflhpiclaejhjboghcpmjgoeh
Inspect views service worker (Inactive)

Figure 16: Activating browser extension



[For you] Internships you should check out before they close.
Unstop <noreply@dare2compete.news> to me
Fri, Apr 11, 9:56 AM (1 day ago)
unstop
Internships where you can shine.
MERN Stack Developer Internship
Yantram Medtech Private Limited
₹ 23000 /Month

Figure 17: Click Scan Images button



Discover more extensions and themes on the Chrome Web Store

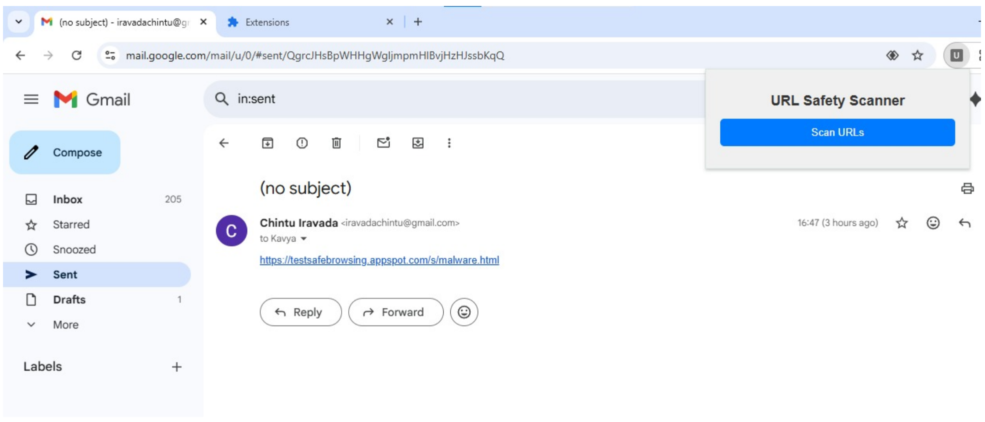
All Extensions

URL Safety Scanner 1.0

Scans URLs on a webpage and checks if they are safe using Google Safe Browsing API.

ID: poajpeeimnpdogefoihmocpekjlbodcn

Figure 18: Activate Browser Extension



mail.google.com/mail/u/0/#sent/QgrcJHsBpWHHgWgljmpmHIBvjHzHJssbKqQ

in:sent

(no subject)

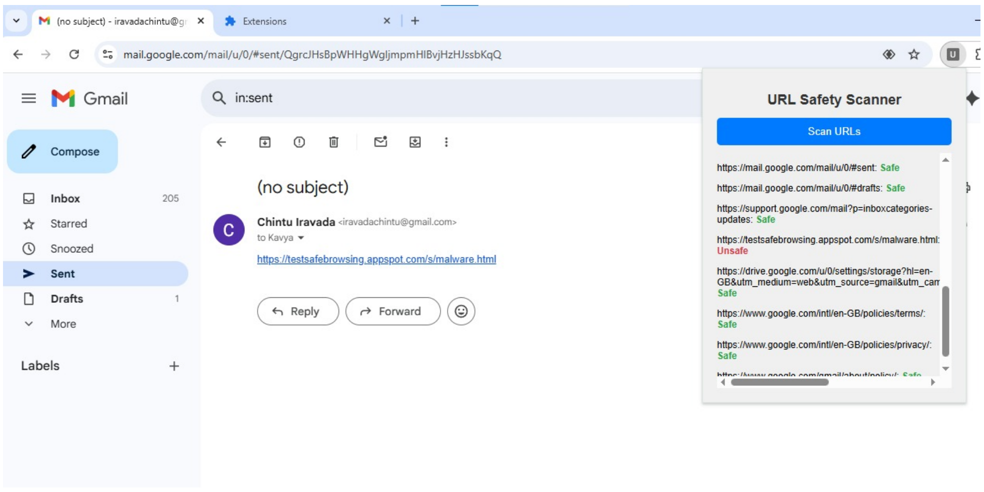
Chintu Iravada <iravadachintu@gmail.com>
to Kavya

16:47 (3 hours ago)

https://testsafebrowsing.appspot.com/s/malware.html

Figure 19: Click Scan URLs

Output



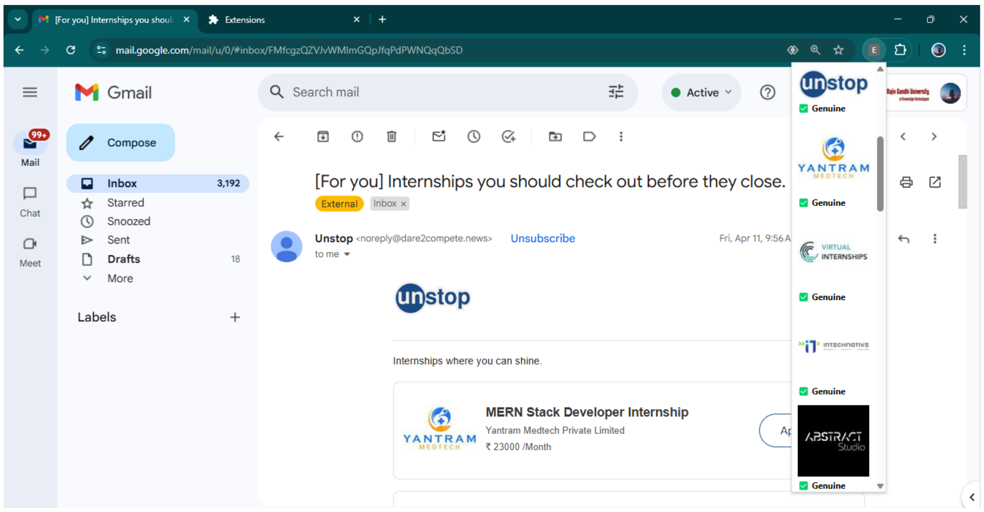
```
URL Safety Scanner
Scan URLs

https://mail.google.com/mail/u/0/#sent: Safe
https://mail.google.com/mail/u/0/#drafts: Safe
https://support.google.com/mail?p=inboxcategories-
updates: Safe
https://testsafebrowsing.appspot.com/s/malware.html:
Unsafe
https://drive.google.com/u/0/settings/storage?hl=en-
GB&utm_medium=web&utm_source=gmail&utm_cam
Safe
https://www.google.com/intl/en-GB/policies/terms/:
Safe
https://www.google.com/intl/en-GB/policies/privacy/:
Safe
```

(no subject)

Chintu Iravada <iravadachintu@gmail.com>
to Kavya

https://testsafebrowsing.appspot.com/s/malware.html

Figure 20: Displaying the output



[For you] Internships you should check out before they close.

Unstop <noreply@dare2compete.news> Unsubscribe
to me

Fri, Apr 11, 9:56 A

Internships where you can shine.

MERN Stack Developer Internship
Yantram Medtech Private Limited
₹ 23000 /Month

```
unstop
Genuine
YANTRAM MEDTECH
Genuine
VIRTUAL INTERNSHIPS
Genuine
intechnative
Genuine
ABSTRACT Studio
Genuine
```

Figure 21: Displaying the output

VISUALIZATION

4.1 Flowchart

The following flowchart presents a comprehensive overview of the entire project, outlining all the steps involved to facilitate a clear and thorough understanding.

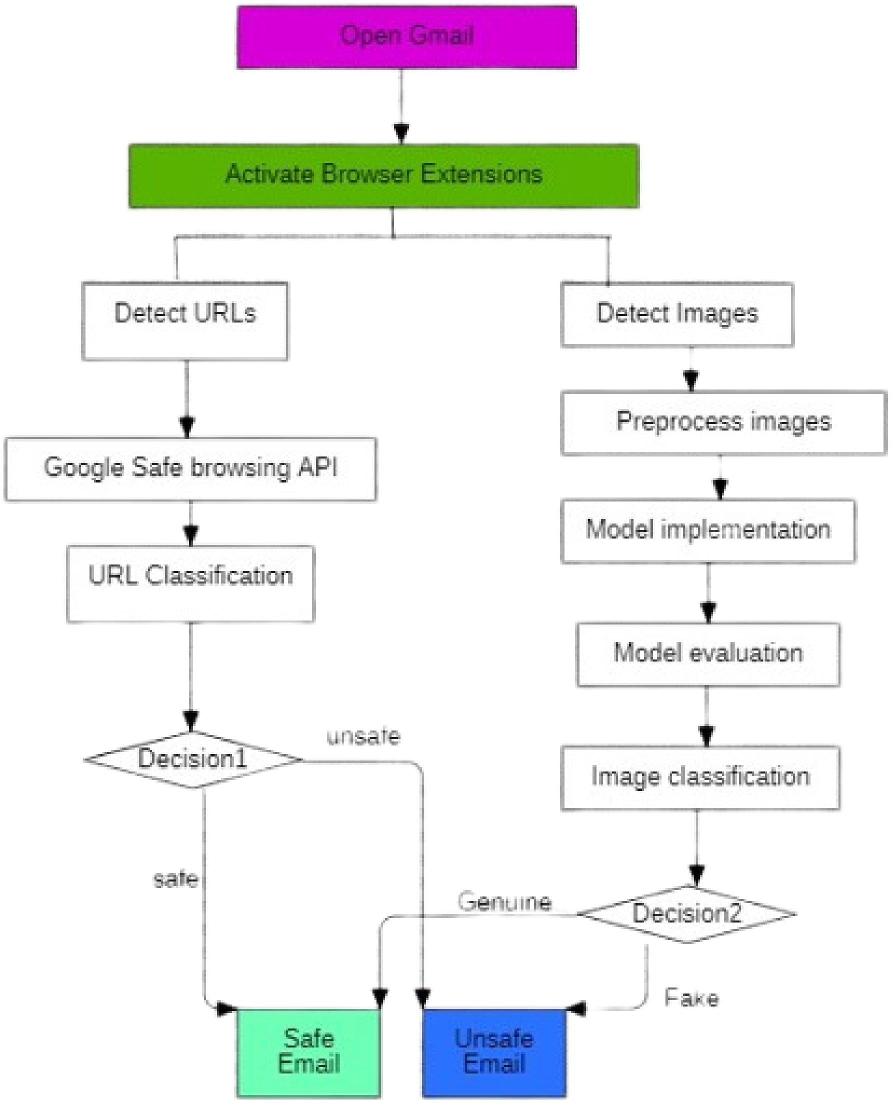


Figure 22: Complete Project Flowchart

4.2 Use Case Diagram

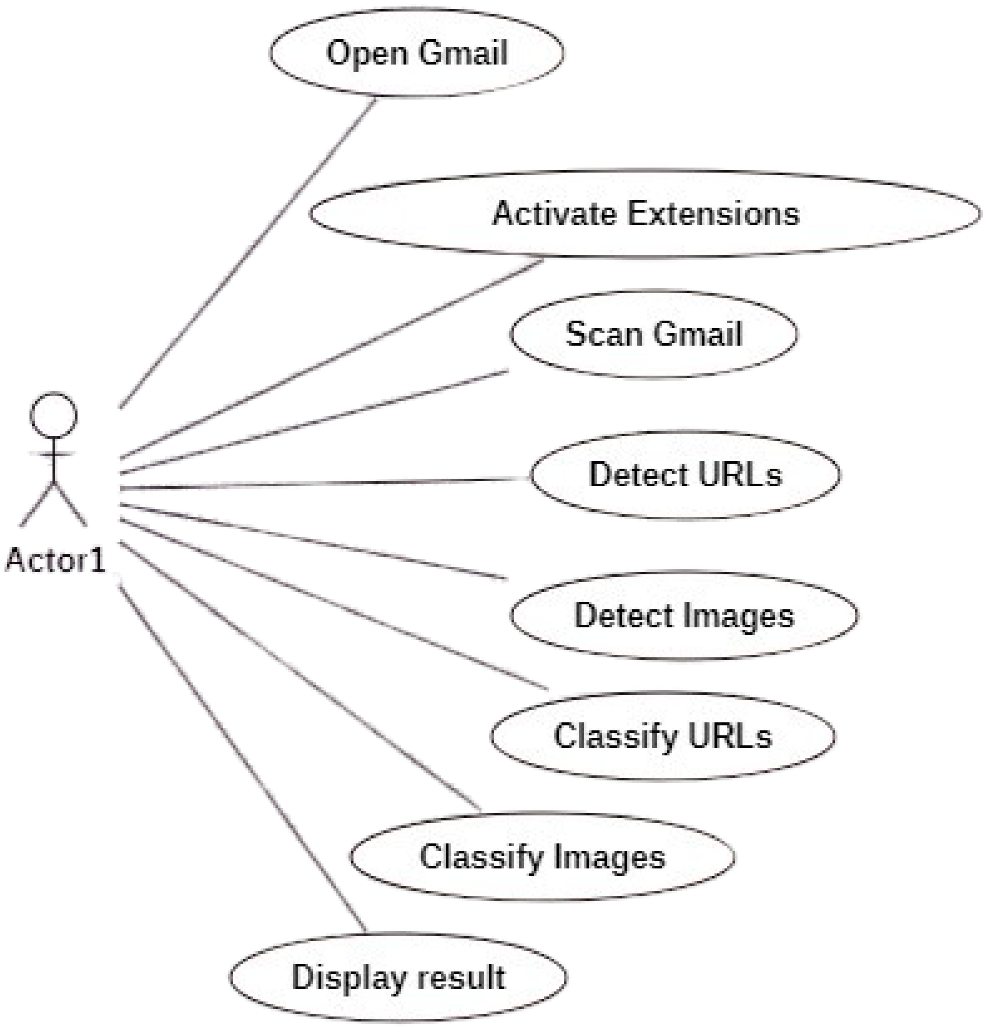


Figure 23: Use Case Diagram for the Browser Extension

The Use Case Diagram illustrates how the **User** interacts with the system, capturing the high-level functionalities provided by the browser extension.

Actors:

- **User:** The person using Gmail and interacting with the extension.

Use Cases:

- **Open Gmail:** The user logs into Gmail to check emails.
- **Activate Browser Extensions:** The user enables the two browser extensions for URL and image phishing detection.

- **Scan Email:** Once activated, the extension scans the email's content.
  - **Detect URLs:** The system identifies all URLs in the email.
  - **Detect Images:** The system extracts images (such as logos) from the email.
- **URL Classification:** URLs are classified as *Safe* or *Unsafe* using a trained model and the Google Safe Browsing API.
- **Image Preprocessing:** Prepares images for model evaluation.
- **Image Classification:** Classifies logos/images as *Genuine* or *Fake* using the ResNet-50 model.
- **Display Result:** Based on the classifications, the system marks the email as either a *Safe Email* or a *Phishing Email*.

This diagram captures the interaction between the user and the system, focusing on what actions are performed and who initiates them.

4.3 Class Diagram

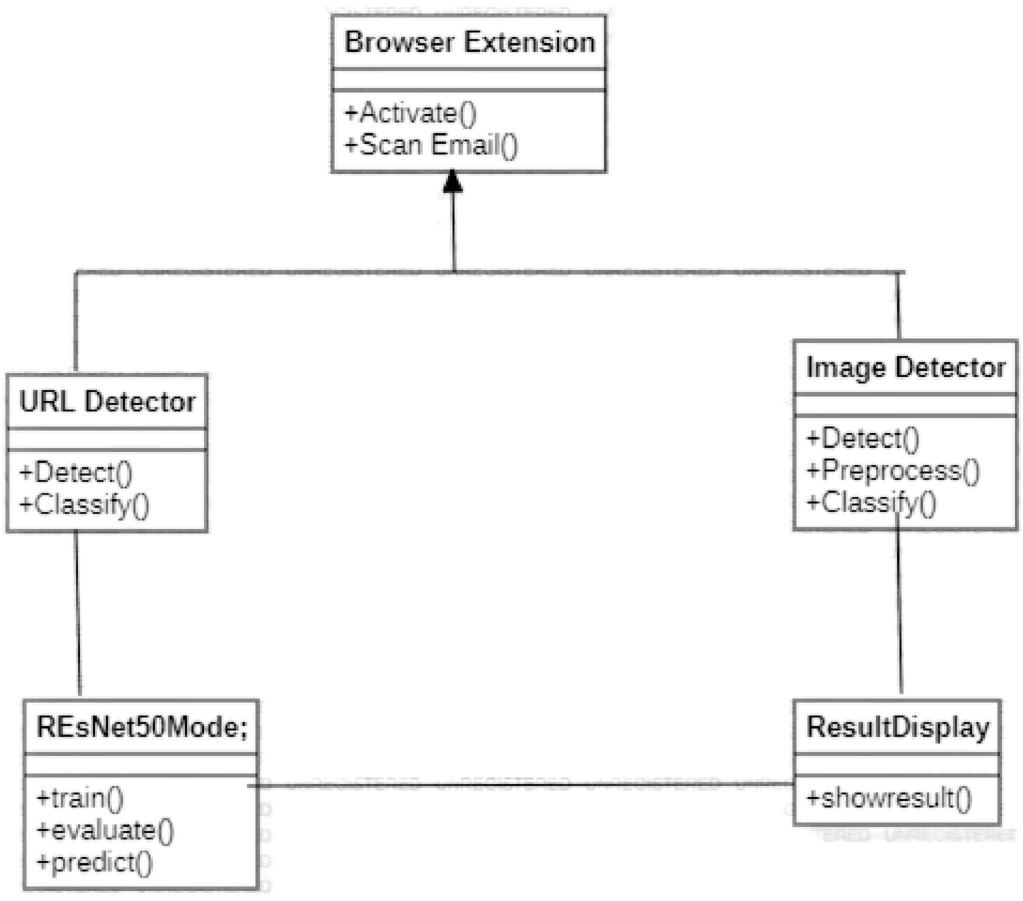


Figure 24: Class Diagram

The Class Diagram describes the static structure of the system by showing the key classes, their attributes, methods, and interactions.

Classes:

- **BrowserExtension**
  - `+ activate()`
  - `+ scanEmail()`
- **URLDetector**
  - `+ detect()`

  - + `classify()`

- **ImageDetector**
  - + `detect()`
  - + `preprocess()`
  - + `classify()`
- **ResNet50Model**
  - + `train()`
  - + `evaluate()`
  - + `predict()`
- **ResultDisplay**
  - + `showResult()`

This diagram demonstrates the modular design of the system, separating responsibilities for URL detection, image detection, classification, and result display.

5.1 CONCLUSION

Email is an inexpensive, effective, and fast way to exchange messages using the internet. Spam email is annoying to end-users, financially damaging, and can be a security risk. The objective of spam email is to collect sensitive personal information about users. The majority of emails in internet traffic contain spam. This work uses deep learning method such as CNN to classify Spam and Not-Spam messages.
This project demonstrates the powerful capabilities of convolutional Neural Networks (CNNs) integrated with Web browser for email spam detection. Traditional intrusion detection systems often rely on manual feature extraction, which can miss crucial relationships and patterns.The usage of loss function and optimizers, such as adam and cross entropy, plays a vital role in enhancing model performance .This approach leads to improved classification accuracy, allowing the system to more effectively distinguish between normal and spam emails

5.2 FUTURE SCOPE

In actuality we have made two different browser extensions, where one is used to detect threat URLS and the other is for detecting fake logos. Our URL extension works on Google safe browsing API and the fake logo detection extension is based on RESNET-50 pretrained model. We have an idea of integrating both extensions into one and some extra features where we can be able to detect deepfake images and threatening text along with threat URLs.
To make this happen we have thought of two approaches

- Either using parallel implementation of models.
- Pipeline of the extensions to integrate all models into one.

We would also like to integrate some more features that come under phishing and make the extensions useful in the real world.

5.3 REFERENCES

References

[1] A. Li-Lian, ”Image classification with ResNet (PyTorch),” 2022. [Online]. Available: https://medium.com/@anglilian/image-classification-with-resnet-pytorch-1e48a4c33905

[2] D. Sarwinda, R. H. Paradisa, A. Bustamam, and P. Anggia, ”Deep learning in image classification using residual network (ResNet) variants for detection of colorectal cancer,” *Procedia Computer Science*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921000284?ref=pdf%20download&fr=RR-2&rr=92f546b41ff67f47

[3] A. Sheneamer, ”Comparison of deep and traditional learning methods for email spam filtering,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 12, no. 1, 2021. [Online]. Available: https://thesai.org/Downloads/Volume12No1/Paper%2064-Comparison%20of%20Deep%20and%20Traditional%20Learning%20Methods.pdf