Mini Project Report on



**Phishing URL Detection System** 

Submitted in partial fulfilment of the requirement for the award of the

degree of

BACHELOR OF COMPUTER APPLICATIONS

Submitted by: University Roll No.: Saloni Singh 2103037

*Under the Mentorship of*

Mr. Mukesh Bhandari

Assistant Professor



Department of Computer Applications

Graphic Era (Deemed to be University)

Dehradun, Uttarakhand

December-2024



CANDIDATE’S DECLARATION

I hereby certify that the work which is being presented in the project report entitled “**Phishing URL Detection System**” in partial fulfilment of the requirements for the award of the Degree of Bachelor of Computer Applications of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the under the mentorship of **Mr. Mukesh Bhandari**, Assistant Professor, Department of Computer Applications, Graphic Era (Deemed to be University), Dehradun.

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

**Name and Signature of Candidate**

This is to certify that the above mentioned statement in the candidate’s declaration is correct to the best of my knowledge.

**Date:::30/11/2024\_\_\_\_\_\_\_\_ Name :Mr.Mukesh Bhandari**

**Signature of Guide :**

**Signature of Supervisor Signature of External Examiner**

**HOD**

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing my deep sense of gratitude to Dr. Kamal Ghanshala, Chairman of Graphic Era University for providing necessary infrastructure and creating good environment.I express my gratitude to **Mr. Mukesh Bhandari**, Professor, Department of Computer Applications, Graphic Era (Deemed to be University), Dehradun., my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

Finally, a note of thanks to the teaching and non-teaching staff of Dept of Computer Application, for their cooperation extended to me, and my friends, who helped me directly or indirectly in the course of the project work.

Saloni Singh(2103037)

**ABSTRACT**

This project implements a phishing URL detection system using machine learning and feature extraction to classify URLs as either "Safe" or "Phishing." The system is built around two key components: a feature extraction module and a Flask-based application interface. The **feature extraction module** analyzes URLs to identify 30 distinct characteristics indicative of phishing behavior. These features include checking for IP address usage, URL length, suspicious symbols like "@", subdomain count, HTTPS usage, WHOIS data validation, and favicon location

The **application interface** integrates the feature extraction module with a user-friendly web application. Users can submit URLs through a form, and the system processes them using a pre-trained machine learning model to provide predictions on whether the URL is safe or phishing. It also calculates confidence scores, offering a probability-based explanation of the results. The Flask application displays the predictions and confidence levels clearly for user interpretation.

This project demonstrates a robust, adaptable, and efficient solution for real-time phishing detection. It effectively mitigates risks posed by common phishing tactics, such as URL shortening, subdomain exploitation, and WHOIS data manipulation, ensuring high accuracy and reliability. Designed for scalability, the system enhances cybersecurity by helping users and organizations identify and avoid phishing threats.

#### Table of Contents

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Description** | **Page No.** |
| Chapter 1 | Introduction | 07-11 |
| Chapter 2 | System Analysis and Requirements Specifications | 12-18 |
| Chapter 3 | System Design and Architecture | 19-35 |
| Chapter 4 | Planning Management | 36-40 |
| Chapter 5 | Features and Output | 41-42 |
| Chapter 6 | Summary and Future Scope | 43-45 |
| Chapter 7 | Methodology | 46-54 |
| Chapter 8 | Working and Discussion | 55-58 |
| Chapter 9 | Conclusion | 59 |
| Chapter 10 | Reference | 60 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure name** | **Page No.** |
| Figure 1.1 | Detection | 11 |
| Figure 3.1 | Flowchart | 19 |
| Figure 3.2 | Technology Used | 19 |
| Figure 3.3 | Directory Tree | 20 |
| Figure 3.4 | ER Diagram | 21 |
| Figure 5.1 | Result | 42 |
| Figure 6.1 | Machine Learning based Url | 45 |
| Figure 7.1 | Data Structure | 47 |
| Figure 7.2 | Dataset | 51 |
| Figure 7.3 | Gradient Boosting | 53 |
| Figure 7.4 | Accuracy score | 54 |
| Figure 8.1 | Workflow | 56 |
| Figure 8.2 | Code | 57 |
| Figure 8.3 | Input | 57 |
| Figure 8.4 | Input | 58 |
| Figure 8.5 | Output | 58 |

**Chapter 1 Introduction**

**1.1 Introduction**

Phishing attacks are a prevalent cybersecurity threat, designed to deceive users into divulging sensitive information by imitating legitimate websites. To address this, the **Phishing URL Detection System** combines machine learning with feature-based URL analysis to identify and classify websites as either safe or phishing. The system's core functionality is divided into two components: feature extraction and detection. The feature extraction module analyzes URLs using heuristics such as the presence of IP addresses, URL length, subdomains, HTTPS usage, and domain registration details. It examines 30 unique features, including suspicious symbols like @ or -, and external favicon links, to evaluate the trustworthiness of a URL. The detection module employs a pre-trained Gradient Boosting Classifier (GBC) to classify URLs and provide confidence levels for predictions. With a user-friendly Flask-based interface, the system allows real-time detection by accepting URLs as input and displaying classification results with associated confidence scores. This robust tool empowers users to mitigate phishing risks effectively and ensures safer online interactions.

**1.2 Problem Statement**

Phishing remains a significant cybersecurity challenge, where attackers deceive users into providing sensitive information by impersonating trusted websites. The primary problem lies in the difficulty of identifying phishing URLs, which are often designed to closely resemble legitimate ones. Manual detection is both time-consuming and prone to errors, while traditional automated systems may suffer from high false positive rates, particularly when legitimate domains use subdomains or other complex structures. The provided code aims to address this issue by developing a robust phishing URL detection system. It uses feature-based analysis to extract 30 critical URL attributes, such as the presence of IP addresses, suspicious symbols (@, -), HTTPS usage, and domain registration details. These features are analyzed using a machine learning model (Gradient Boosting Classifier) to classify URLs as either safe or phishing. The system further incorporates a whitelist for trusted domains to reduce false positives and offers real-time predictions through a Flask-based web interface. This approach ensures an efficient and reliable solution to enhance online security and mitigate phishing attacks.

**1.3 Scope**

The scope of the **Phishing URL Detection System** project:

1. **Real-time URL Classification**: The system provides real-time analysis and classification of URLs as safe or phishing, making it useful for both individual users and organizations to prevent phishing attacks on the fly.
2. **Comprehensive Feature Extraction**: By extracting and analyzing 30 critical URL features, the project ensures an in-depth understanding of a URL’s structure and behavior, improving the accuracy of phishing detection.
3. **Machine Learning Integration**: The project leverages machine learning (Gradient Boosting Classifier) to automate the detection process, enhancing scalability and adaptability to evolving phishing tactics.
4. **Whitelisting for Trusted Domains**: The inclusion of a whitelist for trusted domains, such as educational institutions, reduces false positives, ensuring legitimate URLs are not misclassified as phishing attempts.
5. **User-Friendly Interface**: With a Flask-based web application, the system allows users to interact easily by simply entering a URL, making phishing detection accessible to non-technical users while providing confidence scores for transparency.

**1.4 Significance**

The significance of the **Phishing URL Detection System** lies in its ability to provide a reliable and efficient solution to one of the most pressing cybersecurity threats: phishing attacks. Here are some key aspects that highlight its importance:

1. **Enhanced Online Security**: Phishing attacks are a major threat to individuals, organizations, and financial institutions. By automatically detecting phishing URLs, this system helps protect users from falling victim to scams that can result in data breaches, financial loss, and identity theft.
2. **Scalability and Automation**: Traditional manual methods of detecting phishing websites are slow, labor-intensive, and error-prone. This system automates the detection process, making it scalable and efficient for large volumes of URLs, benefiting both individual users and enterprises with high traffic websites.
3. **Reduced False Positives**: The incorporation of a whitelist for trusted domains and the extraction of 30 distinct URL features ensures that legitimate websites, even those with subdomains or complex structures, are not misclassified. This reduces false positives, improving the system’s reliability.
4. **Proactive Defense**: The system provides real-time URL classification, enabling users to proactively check websites before interacting with them. This helps in preventing phishing attacks before they can cause damage, particularly for users who are less tech-savvy.
5. **Awareness and Empowerment**: By offering a simple web-based interface, the system empowers users to be more vigilant about their online safety. It helps raise awareness about phishing tactics and encourages safer internet practices among the general public.

**1.5 Overview of Machine Learning and NLP in the Phishing URL Detection System**

In the **Phishing URL Detection System**, **Machine Learning (ML)** and **Natural Language Processing (NLP)** play key roles in classifying URLs and analyzing their content. While the primary focus of this project is on **machine learning**, some aspects of NLP could potentially be integrated. Here’s an overview of both in the context of this project:

### ****Machine Learning in Phishing URL Detection****

Machine Learning (ML) is central to the functionality of this project. The system uses a **Gradient Boosting Classifier (GBC)** to classify URLs as either "safe" or "phishing." Here's how machine learning is applied:

1. **Feature Extraction**: The system extracts 30 key features from the URL, such as domain name, length, use of HTTPS, and the presence of suspicious symbols (@, -, etc.). These features are then converted into numerical values that the machine learning model can process.
2. **Training a Model**: The GBC algorithm is trained on a dataset containing both phishing and legitimate URLs. The model learns to identify patterns in the features that distinguish phishing sites from legitimate ones.
3. **Prediction**: Once the model is trained, it can predict whether a new URL is phishing or safe based on the features extracted. The machine learning model also calculates probabilities, offering confidence levels in its predictions.
4. **Improvement**: As new data is gathered, the model can be retrained or fine-tuned to adapt to new phishing tactics and improve its accuracy over time.

### ****Natural Language Processing (NLP) in Phishing URL Detection****

While NLP is not the primary technique in this project, it could complement the system in the future. Here’s how NLP could be utilized:

1. **Content Analysis**: If URLs contain web page content or if the system interacts with the page to extract text (using BeautifulSoup), NLP can help analyze the textual content of a website. For example, NLP techniques can be used to detect **phishing-specific language**, suspicious keywords, or misleading text that tries to trick users into providing sensitive information.
2. **Sentiment Analysis**: NLP can be employed to detect emotionally charged or manipulative language commonly used in phishing emails or websites to induce urgency or panic, prompting users to click malicious links.
3. **Text Classification**: NLP could enhance the classification by analyzing the content of URLs (such as meta descriptions, headers, or body text) for patterns that are typically seen in phishing websites.

**1.6 Future Directions**

The future direction of the **Phishing URL Detection System** involves several key improvements and expansions to enhance its accuracy, functionality, and applicability. First, integrating **deep learning models**, such as neural networks, could enable the system to learn more complex patterns in URLs and web content, improving detection rates, particularly for sophisticated phishing attacks. Additionally, incorporating **Natural Language Processing (NLP)** techniques for content analysis could allow the system to examine the text on web pages, identifying deceptive language or phishing-specific content that machine learning alone might miss. The system could also be enhanced by using **real-time data** from phishing databases and threat intelligence feeds, allowing it to stay updated with the latest phishing tactics. Another potential improvement is the development of a **browser extension** or **mobile application** to offer real-time phishing detection for users as they browse, providing an added layer of protection. Finally, expanding the whitelist to include a broader range of legitimate domains, along with continuous retraining of the machine learning model on a larger dataset, would reduce false positives and make the system more adaptable to new types of phishing attempts.

**1.7 Purpose**

The purpose of the **Phishing URL Detection System** is to provide an automated and efficient solution to identify and protect users from phishing attacks. Phishing remains a significant threat to cybersecurity, as malicious actors often create deceptive websites that appear legitimate to steal sensitive information, such as login credentials and financial data. This project aims to address this issue by analyzing the characteristics of URLs and using machine learning techniques to classify them as either safe or phishing.

The system’s purpose is to:

1. **Enhance Online Security**: By detecting phishing websites in real-time, the system helps prevent users from falling victim to scams that could lead to financial losses, identity theft, or data breaches.
2. **Automate URL Classification**: It automates the process of identifying phishing URLs, making it scalable and suitable for both individuals and organizations.
3. **Minimize False Positives**: The inclusion of a trusted domain whitelist ensures legitimate websites are not misclassified as phishing, improving the system’s reliability.
4. **Provide User Confidence**: By offering confidence scores for URL safety, the system empowers users to make informed decisions about their online safety, particularly in high-risk environments like online banking or e-commerce.
5. **Raise Awareness**: The project aims to raise awareness about the dangers of phishing and encourage safer online practices among users, especially those who may not have technical knowledge about how phishing works.

Overall, the system provides a proactive approach to detecting and mitigating phishing attacks, contributing to a safer digital environment for all users.



Figure 1.1

**Chapter 2: System Analysis and Requirements Specifications**

**2.1 Business Requirements**

The **Phishing URL Detection System** is designed to address the following business requirements:

• **Accurate Phishing Detection**: The primary goal is to develop a system capable of accurately classifying URLs as either safe or phishing. This involves extracting key features from the URL, such as domain names, subdomains, HTTPS usage, and suspicious symbols, and using machine learning algorithms to analyze these features for phishing indicators.

• **Real-time URL Classification**: The system must provide real-time classification of URLs, ensuring that users can instantly check whether a website is safe or a phishing attempt before interacting with it. This will help businesses offer immediate protection to their customers, particularly for online banking, e-commerce, and other sensitive online activities.

• **Scalability**: The system must be designed to scale, ensuring it can handle large volumes of URL checks and provide quick, accurate results even as the number of users and web traffic increases. It should efficiently support businesses and organizations with high-traffic websites or large user bases.

• **Whitelisting of Trusted Domains**: To reduce false positives, the system will include a whitelist of trusted domains. This feature ensures that legitimate websites, such as educational or governmental domains, are not misclassified as phishing sites, thereby maintaining the accuracy and reliability of the detection system.

• **Integration with Existing Security Systems**: The system should be easily integrable with existing business security infrastructures, such as web applications, email filtering systems, or enterprise security solutions. This will enable businesses to incorporate phishing protection seamlessly into their existing cybersecurity frameworks.

• **User-Friendly Interface**: The system will feature an intuitive web-based interface where users can input URLs and quickly receive classification results. It should display the classification status (safe or phishing), confidence levels, and any relevant information about the classification to help users make informed decisions.

• **Data Privacy and Security**: The system will ensure the security of user data by implementing encryption techniques for secure data transmission and storage. It will also incorporate access controls to protect sensitive user data, ensuring compliance with industry standards and regulations for data privacy.

These business requirements focus on providing a reliable, scalable, and easy-to-use system that helps businesses and users protect themselves from phishing attacks and safeguard their online security.

**2.2 System Objectives**

The **system objectives** of the **Phishing URL Detection System** are as follows:

1. **Accurate Phishing URL Classification**: To develop a system capable of accurately classifying URLs as either safe or phishing based on an analysis of their structure, domain, and key features, using machine learning techniques. This ensures that phishing URLs are reliably identified to prevent user interactions with malicious websites.
2. **Real-time Detection and Analysis**: To enable real-time URL classification, allowing users to instantly determine whether a website is safe or a phishing attempt. This provides immediate protection, particularly for users involved in sensitive online activities such as online banking, shopping, or entering personal information.
3. **Machine Learning Model Optimization**: To continuously improve the machine learning model by training it on updated datasets, ensuring it can detect evolving phishing tactics and accurately classify new types of phishing URLs.
4. **Reduce False Positives and Negatives**: To minimize false positives (legitimate websites flagged as phishing) and false negatives (phishing websites classified as safe), through the use of a whitelist for trusted domains and the inclusion of comprehensive URL features for analysis.
5. **Seamless Integration with Existing Systems**: To design the system so that it can easily integrate with existing business platforms, such as websites, security systems, and enterprise tools, providing phishing protection without requiring significant infrastructure changes.
6. **User-Friendly Interface for Easy Interaction**: To create an intuitive and accessible interface where users can input URLs, receive instant phishing classifications, and view detailed confidence scores and reasoning behind the classification, making it easy for both technical and non-technical users to protect themselves.
7. **Enhanced Security and Privacy**: To ensure the security and privacy of user data by using industry-standard encryption methods and implementing strict access control measures to safeguard sensitive information.

These objectives aim to provide an effective, scalable, and user-friendly solution for detecting and mitigating phishing threats, helping businesses and individuals protect themselves from the growing risks of phishing attacks.

**2.3 Hardware and Software Requirements**

Hardware Requirements

* Development Machine:
  + Processor: Multi-core processor (Intel i5 or AMD equivalent) for efficient model training and testing.
  + RAM: Minimum 8GB, recommended 16GB or higher for handling large datasets and concurrent user requests.
  + Storage: Solid-state drive (SSD) with at least 256GB of storage capacity for storing datasets, model files, and system logs.
  + Graphics: Integrated graphics or dedicated GPU for visualizing data and model performance.

**Software Requirements**

1. **Python:**
   1. Version: 3.6 or higher (Python 3.8 or 3.9 recommended)
2. **Flask:**
   1. A lightweight WSGI web application framework.
   2. Installation: pip install Flask
   3. Version: 1.1.2 or higher
3. **Scikit-learn:**
   1. A machine learning library for Python.
   2. Installation: pip install scikit-learn
   3. Version: 0.22 or higher
4. **Pandas:**
   1. A data analysis and manipulation library.
   2. Installation: pip install pandas
   3. Version: 1.0.0 or higher
5. **Matplotlib:**
   1. A plotting library for Python.
   2. Installation: pip install matplotlib
   3. Version: 3.1.0 or higher
6. **Seaborn:**
   1. A data visualization library based on Matplotlib.
   2. Installation: pip install seaborn
   3. Version: 0.10.0 or higher
7. **Pickle:**
   1. Python's built-in module for serializing and de-serializing Python objects (typically does not require a separate installation as it's part of the standard library).
8. **Pre-trained Model:**
   1. A serialized model saved in a file named model.pkl.
   2. This file should be generated beforehand using the pickle module.
9. **CSV File:.**
   1. The dataset file news.csv containing the news articles and their labels.

**2.4 Functional Requirements**

The **functional requirements** for the **Phishing URL Detection System** are as follows:

1. **URL Input and Submission**:
   * The system must allow users to input a URL for analysis through a web-based interface or application.
   * The system should accept URLs in various formats, including HTTP, HTTPS, and domain names with or without subdomains.
2. **Feature Extraction**:
   * The system must extract 30 key features from the input URL. These features may include:
     + Domain name
     + Subdomain count
     + Length of the URL
     + Use of HTTPS
     + Presence of suspicious characters like @, -, or //
     + Domain registration details (WHOIS)
     + Favicon analysis
3. **Phishing Detection and Classification**:
   * The system must use a **machine learning model** (such as Gradient Boosting Classifier) to classify the URL as either safe or phishing based on the extracted features.
   * The model should provide a binary classification result (Safe or Phishing) and a confidence score for each prediction.
   * The system should handle a large dataset of URLs to improve its accuracy over time by training on new phishing data.
4. **Whitelisting of Trusted Domains**:
   * The system should allow the inclusion of a list of trusted domains (e.g., educational institutions, government websites) that are automatically classified as safe to reduce false positives.
   * Whitelisted domains should be excluded from further phishing analysis.
5. **Real-time Processing**:
   * The system must provide real-time classification results. As soon as the user submits a URL, the system should process the URL and return the classification result and confidence score within a short time frame.
6. **User Interface**:
   * The system should provide a user-friendly interface where users can:
     + Enter URLs for analysis
     + View the classification result (safe or phishing)
     + See confidence levels (percentage probability of being safe or phishing)
     + Understand the reasoning behind the classification through explanations of key features (if possible).
7. **Reporting and Logs**:
   * The system should log all URL submissions, including the date/time of submission, URL, classification result, confidence score, and any relevant metadata for future audits and tracking.
   * Reports should be available to administrators for monitoring and improving system performance.
8. **Integration with Existing Systems**:
   * The system must support integration with other security systems, such as email filters, web application firewalls, or enterprise security platforms, allowing businesses to incorporate phishing URL detection within their existing infrastructure.
9. **Continuous Learning and Model Updating**:
   * The system should allow for the continuous retraining of the machine learning model using new phishing data to adapt to evolving phishing techniques.
   * There should be an automated process for updating the model with the latest phishing URL datasets to improve detection accuracy.
10. **Security and Privacy**:
    * The system should ensure the privacy and security of user-submitted URLs and data, employing encryption methods for secure data transmission.
    * Access controls should be implemented to restrict unauthorized access to sensitive data, ensuring compliance with data privacy standards.

These functional requirements aim to ensure that the Phishing URL Detection System is effective, accurate, scalable, and secure, while providing an easy-to-use interface for end users.

**2**.**5 Non-Functional Requirements**

1. **Performance**
   * The system should be capable of processing and analyzing news articles within a reasonable timeframe, typically within seconds.
   * Analysis results should be delivered to users in real-time or near-real-time to ensure a seamless user experience.
   * Scalability measures should be in place to handle fluctuations in user traffic and article submission rates without compromising performance.
2. **Security**
   * Data confidentiality and integrity should be ensured throughout the article submission and analysis process.
   * Access controls should be implemented to prevent unauthorized access to sensitive user data and analysis results.
   * Measures should be taken to mitigate the risk of adversarial attacks aimed at manipulating analysis outcomes or compromising model integrity.
3. **Reliability**
   * The system should be highly available, with minimal downtime or service disruptions.
   * Failover mechanisms should be in place to handle server failures or network interruptions gracefully.
   * Regular backups of data and model parameters should be performed to safeguard against data loss and ensure system resilience.
4. **Usability**
   * The user interface should be intuitive and accessible to users with varying levels of technical expertise.
   * Clear instructions and guidance should be provided to users on how to submit articles, interpret analysis results, and provide feedback.
   * The system should adhere to established design principles and accessibility standards to enhance usability for all users.
5. **Maintainability**
   * The system architecture should be modular and well-documented to facilitate maintenance and future enhancements.
   * Codebase should follow best practices in software development, including version control, code review, and automated testing.
   * Continuous monitoring and logging should be implemented to track system
   * performance, identify issues, and support troubleshooting efforts.

**Chapter 3: System Design and Architecture**

**3.1 Flowchart**

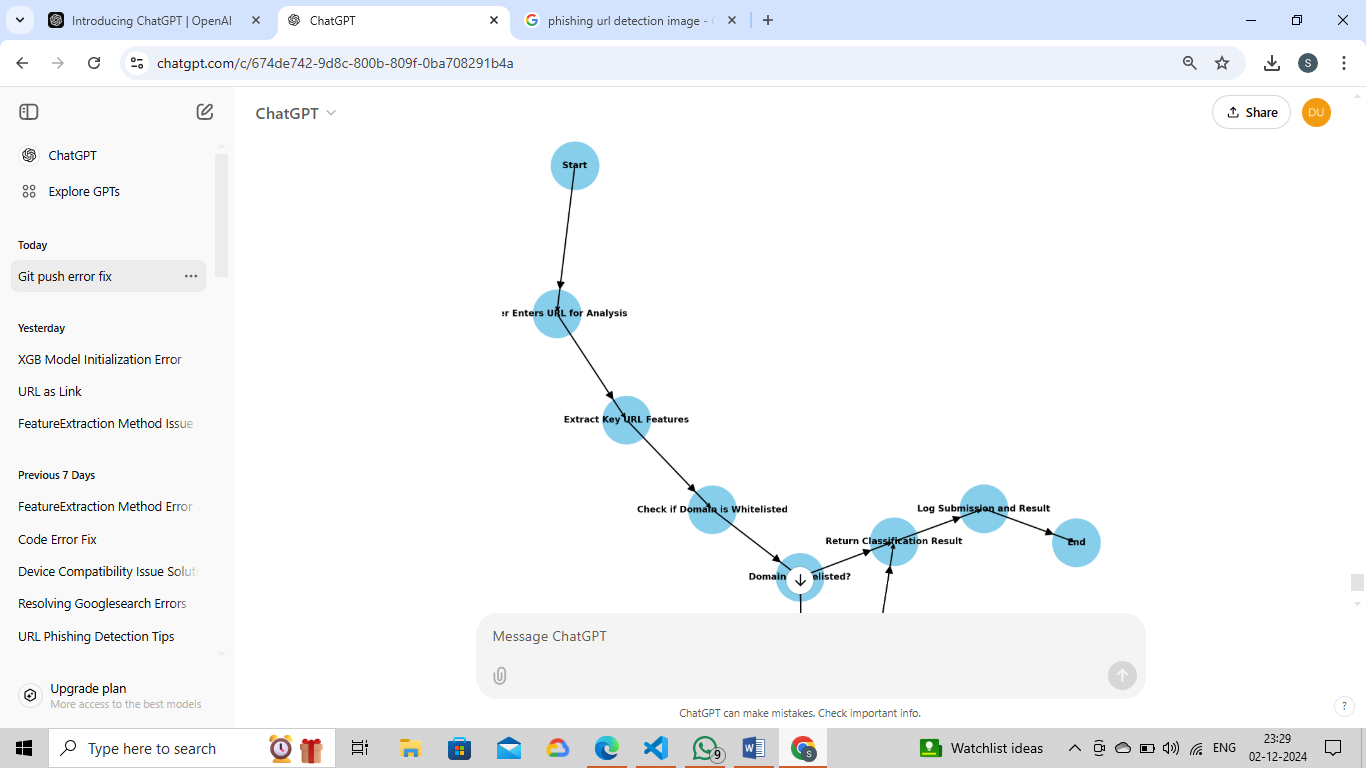
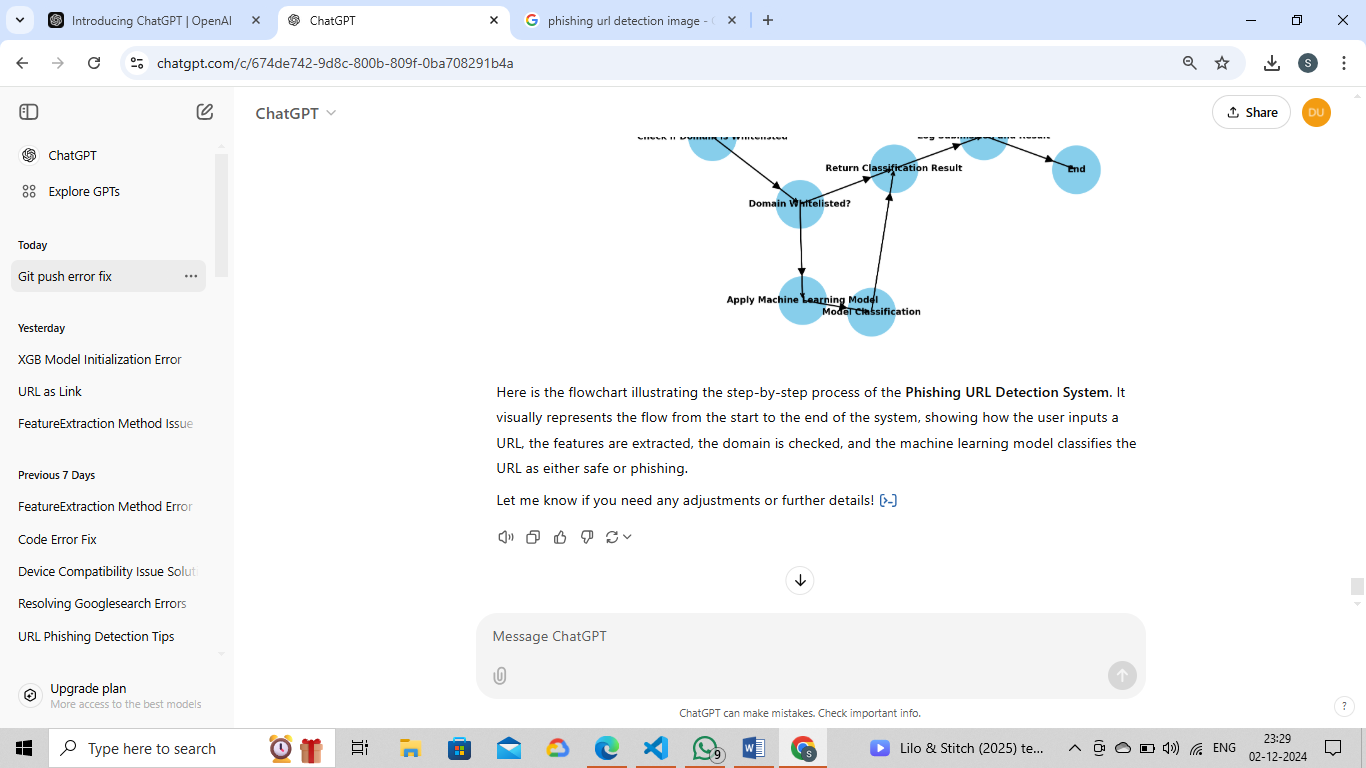


Figure 3.1

## Technologies Used

Figure 3.2

**Directory Tree**

├── pickle

│   ├── model.pkl

├── static

│   ├── styles.css

├── templates

│   ├── index.html

├── Phishing URL Detection.ipynb

├── Procfile

├── README.md

├── app.py

├── feature.py

├── phishing.csv

├── requirements.txt

**Figure 3.3**

**ALGORITHM**

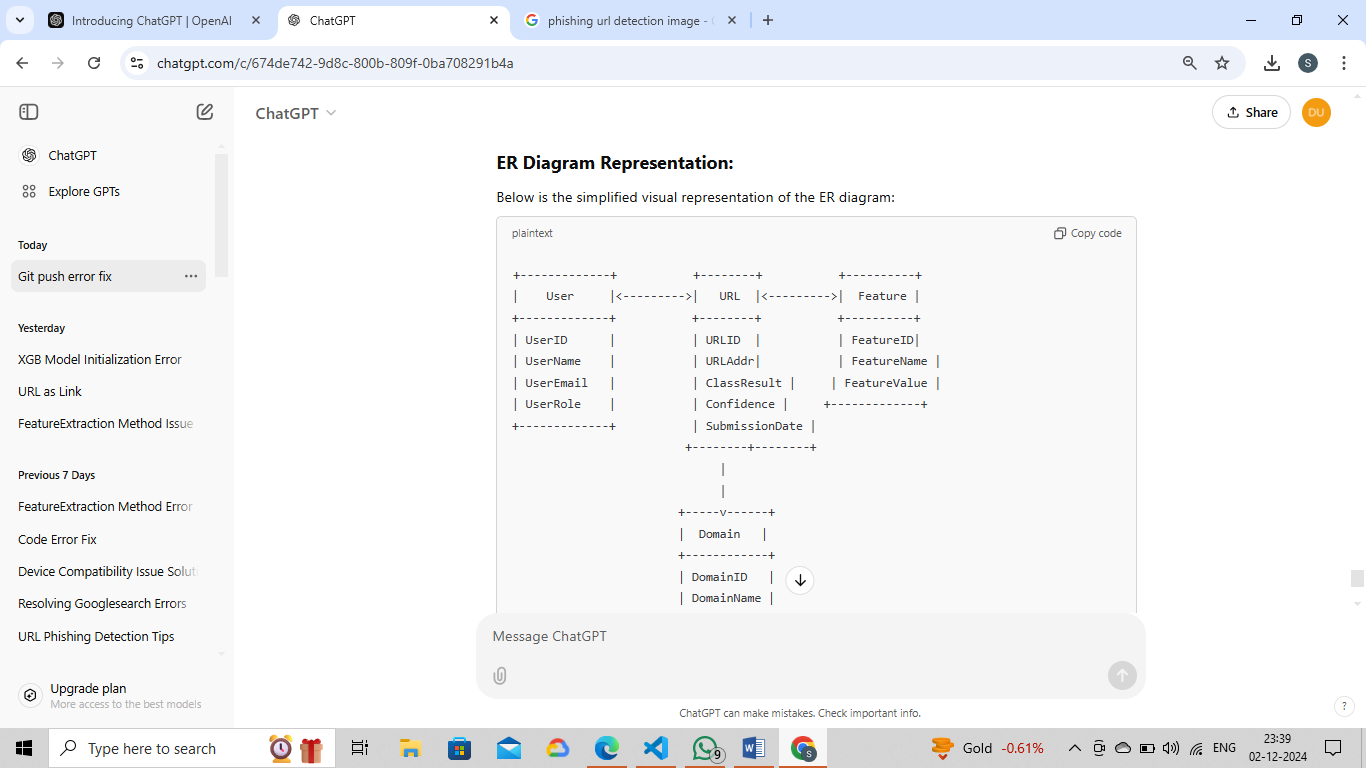
The **Phishing URL Detection Algorithm** operates as follows:

1. **User Input**: The user provides a URL for analysis.
2. **Feature Extraction**: Key features (such as domain name, URL length, HTTPS usage, suspicious symbols, subdomains, and WHOIS data) are extracted from the URL.
3. **Whitelisted Domain Check**: The system checks if the domain belongs to a trusted whitelist. If it does, the URL is classified as **Safe**.
4. **Machine Learning Classification**: If the domain is not whitelisted, a machine learning model (e.g., Gradient Boosting Classifier) classifies the URL as either **Safe** or **Phishing** based on the extracted features.
5. **Return Result**: The system displays the classification result (Safe/Phishing) along with a confidence score.
6. **Log Data**: The URL, result, and confidence score are logged for auditing and model improvement.
7. **End**: The process concludes after providing the result to the user.

This algorithm efficiently detects phishing URLs by combining feature extraction and machine learning classification, providing reliable protection for users.

**3.2 ER-Diagram**

**Traditional**



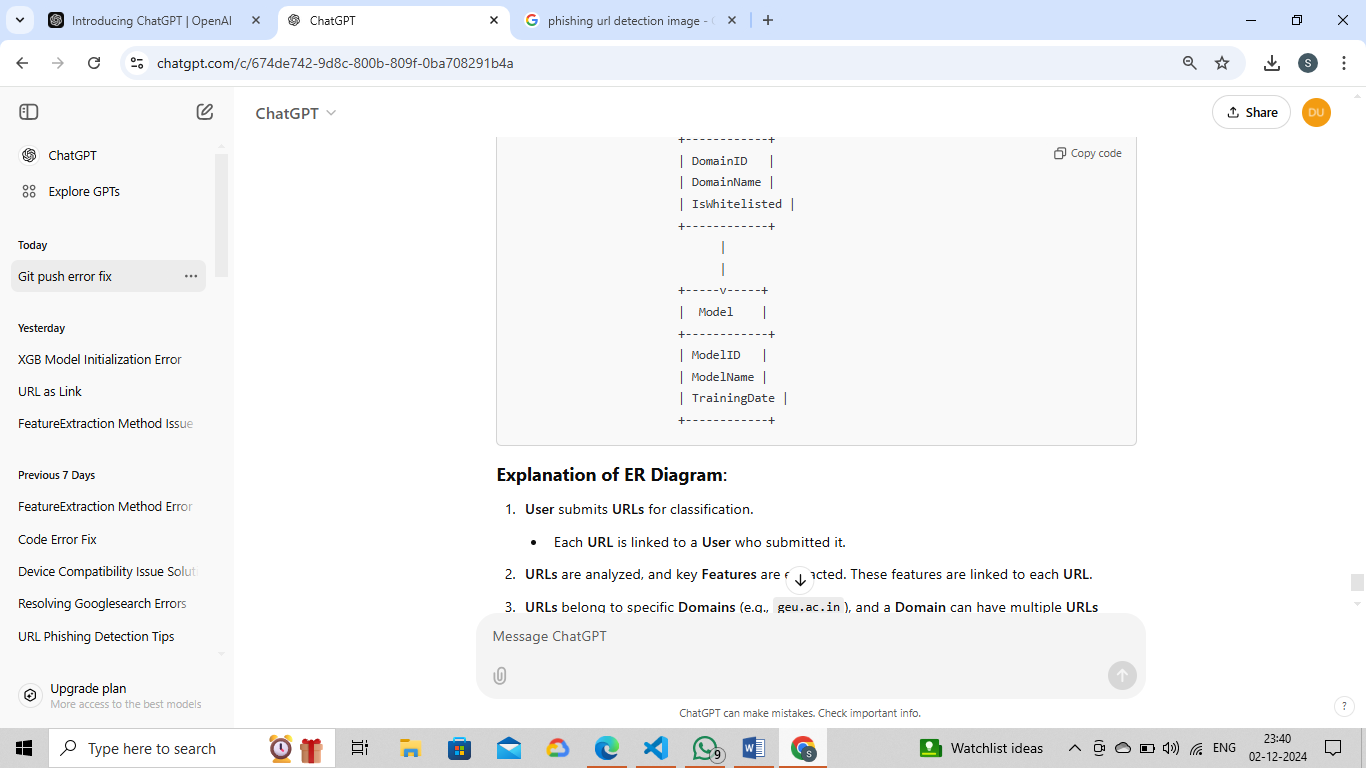


Figure 3.4

Creating an Entity-Relationship (ER) diagram for the Phishing Url Detection System involves identifying the key entities and relationships involved in the system. Below is a textual representation of the ER diagram components for this application:

**Entities**

### ****ER Diagram for Phishing URL Detection System****

An **Entity-Relationship (ER) Diagram** is used to represent the data model of a system. Below is the conceptual ER diagram for the **Phishing URL Detection System**, along with explanations for each entity and its relationships.

#### ****Entities and Attributes:****

1. **User**
   * **Attributes**:
     + UserID (Primary Key)
     + UserName
     + UserEmail
     + UserRole (Admin/User)

**Explanation**: This entity stores the details of the user who interacts with the system to submit URLs for classification.

1. **URL**
   * **Attributes**:
     + URLID (Primary Key)
     + URLAddress (URL entered by the user)
     + ClassificationResult (Safe/Phishing)
     + ConfidenceScore (Percentage indicating the certainty of the classification)
     + SubmissionDate (Date and time the URL was submitted)

**Explanation**: This entity stores the details of the URLs submitted by users for classification. It includes the result of the classification and the confidence score generated by the machine learning model.

1. **Feature**
   * **Attributes**:
     + FeatureID (Primary Key)
     + FeatureName (e.g., DomainName, URLLength, SuspiciousSymbols)
     + FeatureValue (Value of the feature for a specific URL)

**Explanation**: The **Feature** entity stores the individual features extracted from a URL. These features are used by the machine learning model to classify the URL as phishing or safe.

1. **Domain**
   * **Attributes**:
     + DomainID (Primary Key)
     + DomainName (e.g., "geu.ac.in")
     + IsWhitelisted (True/False)

**Explanation**: This entity stores the details of domains that are checked against a whitelist. Trusted domains are marked as "Whitelisted", and any URL under these domains is classified as **Safe** without further analysis.

1. **Model**
   * **Attributes**:
     + ModelID (Primary Key)
     + ModelName (e.g., GradientBoostingClassifier)
     + TrainingDate (Date when the model was last trained)

**Explanation**: This entity stores information about the machine learning model used for classification. It tracks which model is used and when it was last trained or updated.

#### ****Relationships****:

1. **User Submits URL**:
   * **Relationship**: A user can submit multiple URLs for classification, but each URL is submitted by exactly one user.
   * **Cardinality**: One-to-Many (One user can submit many URLs).
2. **URL Has Features**:
   * **Relationship**: Each URL has multiple extracted features (e.g., domain, URL length, etc.).
   * **Cardinality**: One-to-Many (Each URL will have many features associated with it).
3. **URL Belongs to Domain**:
   * **Relationship**: Each URL belongs to one domain. A domain can have multiple URLs associated with it (e.g., multiple URLs from the same domain).
   * **Cardinality**: One-to-Many (Each domain can have many URLs).
4. **URL Classified by Model**:
   * **Relationship**: Each URL is classified by a machine learning model.
   * **Cardinality**: Many-to-One (Many URLs are classified by the same model).

### ****ER Diagram Representation:****

Below is the simplified visual representation of the ER diagram:

+-------------+ +--------+ +----------+

| User |<--------->| URL |<--------->| Feature |

+-------------+ +--------+ +----------+

| UserID | | URLID | | FeatureID|

| UserName | | URLAddr| | FeatureName |

| UserEmail | | ClassResult | | FeatureValue |

| UserRole | | Confidence | +-------------+

+-------------+ | SubmissionDate |

+--------+--------+

|

|

+-----v------+

| Domain |

+------------+

| DomainID |

| DomainName |

| IsWhitelisted |

+------------+

|

|

+-----v-----+

| Model |

+------------+

| ModelID |

| ModelName |

| TrainingDate |

+------------+

### ****Explanation of ER Diagram****:

1. **User** submits **URLs** for classification.
   * Each **URL** is linked to a **User** who submitted it.
2. **URLs** are analyzed, and key **Features** are extracted. These features are linked to each **URL**.
3. **URLs** belong to specific **Domains** (e.g., geu.ac.in), and a **Domain** can have multiple **URLs** associated with it.
4. A **Model** is used to classify **URLs**. The **Model** stores the details of the machine learning model that processes URLs for phishing detection.

### ****Conclusion:****

This **ER Diagram** provides a structured view of how the various components of the **Phishing URL Detection System** interact. It covers user interaction with the system, the classification of URLs, and the extraction of features, while also tracking the domains and machine learning model used for classification.

**3.3 Data Flow Diagrams**

### ****Data Flow Diagram (DFD) for Phishing URL Detection System****

A **Data Flow Diagram (DFD)** visually represents the flow of data through a system. It shows how data is input, processed, and output within the system. For the **Phishing URL Detection System**, the DFD will outline the major processes involved, including user interaction, feature extraction, machine learning classification, and output generation.

### ****Level 0 DFD (Context Diagram)****

At the highest level (Level 0), the **Phishing URL Detection System** is represented as a single process. It interacts with external entities, such as the **User** and **External Databases** (like the whitelist of trusted domains and machine learning model data).

+-----------------------+

| |

| Phishing URL | <--- User ---> [Submit URL]

| Detection System |

| |

| | ---> [Classify URL] ---> [Result (Safe/Phishing)]

| |

+-----------------------+

^

|

|

[Whitelist Domains] <---> [Model Data]

### ****Explanation of Level 0 DFD****:

* **External Entities**:
  + **User**: The user provides the URL for analysis.
  + **Whitelist Domains**: A list of trusted domains (e.g., educational or governmental websites). The system checks if a domain belongs to this list to reduce false positives.
  + **Model Data**: The machine learning model that is used for classifying the URL based on its extracted features.
* **Process**: The **Phishing URL Detection System** processes the input URL by checking it against the whitelist, extracting features, applying the machine learning model, and generating a classification result.

### ****Level 1 DFD (Detailed View)****

Level 1 of the DFD breaks the **Phishing URL Detection System** into its major internal processes: **URL Submission**, **Feature Extraction**, **Whitelisted Domain Check**, **Model Classification**, and **Output Generation**.

+------------------------+

| |

| User Inputs URL |

| |

+-----------+------------+

|

v

+--------------+

| |

| Extract URL |

| Features | <----- Extracted Features

| |

+------+-------+

|

v

+------------------------+

| |

| Check Against Whitelist |

| |

+----------+-------------+

|

+-----------------v--------------------+

| |

+-------+--------+ +------v-------+

| Whitelisted | | Not in |

| Domain? | | Whitelist |

+-------+--------+ +------+-------+

| |

+------v------+ +--v------------+

| Safe URL | | Extracted |

| Classification| | Features Processed|

+------+-------+ +------v-----------+

|

v

+--------------+

| |

| Apply ML |

| Model |

| Classification|

+------+-------+

|

v

+------------------+

| |

| Return Result |

| (Safe/Phishing) |

+------------------+

### ****Explanation of Level 1 DFD****:

1. **User Inputs URL**:
   * The user submits a URL to the system for analysis.
2. **Extract URL Features**:
   * Key features of the URL are extracted (e.g., domain name, URL length, presence of suspicious symbols, HTTPS usage, etc.).
3. **Check Against Whitelist**:
   * The system checks if the domain of the submitted URL exists in the **whitelist** of trusted domains.
     + If the domain is in the whitelist, it is **classified as Safe** without further processing.
     + If not, the system proceeds with feature extraction and machine learning classification.
4. **Apply Machine Learning Model**:
   * If the URL is not in the whitelist, the system uses a machine learning model (e.g., **Gradient Boosting Classifier**) to classify the URL. The model is trained on historical data to determine if the URL is phishing or safe based on its features.
5. **Return Result (Safe/Phishing)**:
   * The system outputs the classification result (either **Safe** or **Phishing**) along with a confidence score that indicates the model's certainty.

### ****Level 2 DFD (Further Detailing)****

At Level 2, each of the internal processes can be broken down further. For example, the **Feature Extraction** process can be expanded to show how individual features like domain name, URL length, and other characteristics are extracted from the URL.

+---------------------------+

| |

| Extract Domain Name |

| |

+--------------+------------+

|

v

+------------------+------------------+

| |

+------v-------+ +-----v------+

| URL Length | | HTTPS Check |

+--------------+ +------------+

| |

v v

+----------+-----------+ +---------------v---------------+

| | | |

+------v-------+ +--------v--------+ +---------------+--+

| Suspicious | | Subdomain Count | | WHOIS Data |

| Symbols | | | | (Registration) |

+--------------+ +------------------+ +------------------+

### ****Explanation of Level 2 DFD****:

* **Extract Domain Name**: The domain name is parsed from the URL (e.g., www.example.com).
* **URL Length**: The length of the URL is measured and recorded.
* **HTTPS Check**: The system checks if the URL uses HTTPS or HTTP.
* **Suspicious Symbols**: The system scans the URL for symbols like @, -, or multiple slashes //, which are often used in phishing URLs.
* **Subdomain Count**: The number of subdomains in the URL is extracted (e.g., blog.example.com has one subdomain).
* **WHOIS Data**: WHOIS registration information is retrieved for the domain to determine the registration age, owner, etc.

### ****Conclusion:****

The **Data Flow Diagram (DFD)** for the **Phishing URL Detection System** shows how data moves through the system, from the user's input of a URL, to feature extraction, checking against a whitelist, applying machine learning for classification, and finally outputting a result. Each process ensures that the system can efficiently and accurately classify URLs as safe or phishing, providing users with protection against online threats.

**3.4 High-Level Architecture**

### ****High-Level Architecture for Phishing URL Detection System****

The **High-Level Architecture** for the **Phishing URL Detection System** represents the major components of the system and how they interact with each other to detect phishing URLs and provide classification results. Below is a description of the architecture along with its components:

### ****1. User Interface (UI)****

* **Role**: The user interface is the entry point for the system, where users can input the URL they want to analyze.
* **Technology**: Web-based interface (built using frameworks like Flask, HTML, CSS, JavaScript) or desktop applications.
* **Interaction**:
  + The user submits a URL for analysis via a simple form.
  + Displays the result (safe or phishing) and confidence score after processing.

### ****2. Front-End (Client-Side)****

* **Role**: It serves as the layer where the user interacts with the system. It communicates with the backend system to send and receive data (i.e., the URL and classification result).
* **Technology**: HTML, CSS, JavaScript, ReactJS, or Angular.
* **Interaction**:
  + Sends the URL input from the user to the backend via API requests (RESTful API).
  + Receives classification results (Safe/Phishing) along with confidence scores.

### ****3. API Gateway****

* **Role**: The API Gateway acts as an intermediary between the front-end and the backend services. It is responsible for receiving user requests, routing them to the appropriate backend services, and returning the responses to the user.
* **Technology**: Flask (for web apps), ExpressJS (for Node.js applications), or FastAPI.
* **Interaction**:
  + Receives the URL input from the UI and forwards it to the URL processing and classification service.
  + Sends back the classification result and confidence score to the user interface.

### ****4. URL Processing and Feature Extraction Service****

* **Role**: This service is responsible for extracting key features from the input URL that will be used for classification. These features include the domain, subdomains, URL length, HTTPS usage, suspicious symbols, WHOIS data, etc.
* **Technology**: Python (with libraries like re, whois, beautifulsoup4, etc.).
* **Interaction**:
  + Receives the URL from the API Gateway.
  + Extracts features from the URL for further classification by the machine learning model.

### ****5. Phishing Detection Engine (Machine Learning Model)****

* **Role**: This engine uses a machine learning model (e.g., Gradient Boosting Classifier) to analyze the extracted features and classify the URL as either **Safe** or **Phishing**.
* **Technology**: Python (with libraries like scikit-learn, pickle for model deployment).
* **Interaction**:
  + The model is trained on a dataset of phishing and legitimate URLs.
  + The model classifies the extracted features and returns a prediction (Safe or Phishing) along with a confidence score.
  + The classification result is sent back to the URL processing service.

### ****6. Domain Whitelisting Service****

* **Role**: This service stores and checks the URL domain against a whitelist of trusted domains (e.g., educational institutions, government websites) to ensure that known, safe URLs are not flagged incorrectly.
* **Technology**: Database (e.g., SQL or NoSQL) to store whitelisted domains.
* **Interaction**:
  + Before sending the URL to the machine learning model, the domain is checked against the whitelist.
  + If the domain is found in the whitelist, the URL is classified as **Safe** without requiring further analysis.
  + If not, the URL is passed to the classification engine for further processing.

### ****7. Logging and Reporting Service****

* **Role**: The logging service records all incoming URL submissions, classification results, and confidence scores. This data is important for auditing, monitoring system performance, and retraining the machine learning model.
* **Technology**: Python logging or third-party services like Elasticsearch, Logstash, and Kibana (ELK stack).
* **Interaction**:
  + Logs each URL submission with the result (safe/phishing), confidence score, and timestamp.
  + Stores logs for auditing, security, and performance evaluation

### ****8. Database (Optional for Storing URLs and Results)****

* **Role**: A database can be used to store historical data for URLs, including user input, classification results, and machine learning model details.
* **Technology**: SQL (e.g., MySQL, PostgreSQL) or NoSQL (e.g., MongoDB).
* **Interaction**:
  + Stores details of URL submissions, their classification results, confidence scores, and related features.
  + Allows for easy retrieval of data for reporting, model retraining, and system analysis.

### ****High-Level Architecture Diagram****

+---------------------------------------+

| User Interface (UI) |

| (Web Application or Mobile App) |

+---------------------------------------+

|

v

+-------------------+

| API Gateway |

| (Handles requests) |

+-------------------+

|

v

+-------------------------------+

| URL Processing and Feature |

| Extraction Service |

+-------------------------------+

|

v

+---------------------------+

| Domain Whitelisting Service|

+---------------------------+

|

v

+-------------------------------+

| Phishing Detection Engine |

| (Machine Learning Model) |

+-------------------------------+

|

v

+-------------------------------+

| Logging and Reporting Service|

+-------------------------------+

|

v

+-------------------------------+

| Database (Optional) |

| (Stores URL, Results, Logs) |

+-------------------------------+

### ****Explanation of Architecture****:

1. **User Interface (UI)**:
   * The user interacts with the system via the UI to input a URL for phishing detection.
2. **API Gateway**:
   * Receives the URL input from the UI, sends it to the backend services for processing, and returns the classification result.
3. **URL Processing and Feature Extraction**:
   * This service extracts essential features from the provided URL for classification (e.g., domain, URL length, etc.).
4. **Domain Whitelisting Service**:
   * Checks if the domain of the URL is in a predefined list of trusted domains. If whitelisted, the URL is classified as safe.
5. **Phishing Detection Engine (Machine Learning Model)**:
   * If the domain is not whitelisted, the system uses a machine learning model to classify the URL as phishing or safe based on extracted features.
6. **Logging and Reporting Service**:
   * Logs every URL submission and classification result for auditing, monitoring, and performance purposes.
7. **Database** (optional):
   * Stores URL history, classification results, and additional data for future reference or model retraining.

**Table Structure**

**Database Table Structure for Phishing URL Detection System**

Below is the **table structure** for the **Phishing URL Detection System**, including the necessary entities and relationships required for storing and managing the data processed by the system.

**1. User Table**

This table stores information about the users who interact with the system.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| UserID | INT | Primary Key, unique identifier for each user |
| UserName | VARCHAR(255) | Name of the user |
| UserEmail | VARCHAR(255) | Email address of the user |
| UserRole | VARCHAR(50) | Role of the user (Admin/User) |

**2. URL Table**

This table stores information about each URL submitted by users for phishing detection.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| URLID | INT | Primary Key, unique identifier for each URL |
| UserID | INT | Foreign Key, references UserID in the **User** table |
| URLAddress | VARCHAR(2048) | The URL submitted by the user |
| ClassificationResult | VARCHAR(50) | Classification result (Safe/Phishing) |
| ConfidenceScore | DECIMAL(5,2) | Confidence score (percentage) for the classification result |
| SubmissionDate | DATETIME | Date and time when the URL was submitted |

**3. Feature Table**

This table stores the extracted features from the URLs for analysis and classification.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Column Name** | **Data Type** | **Description** | | FeatureID | INT | Primary Key, unique identifier for each feature | | URLID | INT | Foreign Key, references URLID in the **URL** table | | FeatureName | VARCHAR(255) | Name of the feature (e.g., Domain, URLLength, HTTPS, SuspiciousSymbols) | | FeatureValue | VARCHAR(255) | The value of the feature (e.g., "example.com", "True", "45") | |

**4. Domain Table**

This table stores information about domains, including whether they are whitelisted.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| DomainID | INT | Primary Key, unique identifier for each domain |
| DomainName | VARCHAR(255) | The name of the domain (e.g., geu.ac.in) |
| IsWhitelisted | BOOLEAN | Whether the domain is in the whitelist (True/False) |

**5. Model Table**

This table stores information about the machine learning model used for phishing detection.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| ModelID | INT | Primary Key, unique identifier for the model |
| ModelName | VARCHAR(255) | Name of the model (e.g., Gradient Boosting Classifier) |
| TrainingDate | DATETIME | Date when the model was last trained or updated |

**6. Logs Table**

This table stores logs for auditing and monitoring purposes, including classification results and confidence scores.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| LogID | INT | Primary Key, unique identifier for each log entry |
| URLID | INT | Foreign Key, references URLID in the **URL** table |
| ClassificationResult | VARCHAR(50) | Classification result (Safe/Phishing) |
| ConfidenceScore | DECIMAL(5,2) | Confidence score of the classification |
| Timestamp | DATETIME | Date and time of the log entry |

**Relationships Between Tables:**

* **User Table**:
  + A user can submit many URLs, but each URL is submitted by only one user.
* **URL Table**:
  + A URL can have many features (stored in the **Feature Table**).
  + A URL belongs to one user (through **UserID**).
  + A URL can be linked to one domain (through **DomainID**).
* **Feature Table**:
  + A feature belongs to a specific URL.
* **Domain Table**:
  + A domain can have multiple URLs, but each URL belongs to only one domain.
* **Logs Table**:
  + Each log entry references a URL to store the classification result and confidence score for auditing purposes.

**Summary:**

The **Phishing URL Detection System** requires several key tables to store data efficiently:

* **User**: Stores user details.
* **URL**: Stores information about each URL submission.
* **Feature**: Stores extracted features from URLs.
* **Domain**: Stores trusted domains and their whitelist status.
* **Model**: Stores details about the machine learning model used for classification.
* **Logs**: Stores logs for auditing and monitoring system performance.

This table structure ensures that all relevant data (URL submissions, classification results, features, domains, and logs) are stored efficiently for easy retrieval and analysis.

**Chapter 4: Planning Management**

**4.1 Project Planning and Scheduling**

**4.1.1 Project Scope**

#### ****1. User Input and Prediction:****

* A web interface where users can input a URL for analysis.
* Backend processing to classify the URL as either **Safe** or **Phishing** using a pre-trained machine learning model.
* Display of the prediction result (Safe/Phishing) along with the confidence score (probability).

#### ****2. Model Integration:****

* Utilize a pre-trained **Gradient Boosting Classifier** model stored in a pickle file (model.pkl).
* Extract relevant features (e.g., domain, URL length, HTTPS usage, suspicious symbols) from the URL.
* Use the extracted features as input to the machine learning model for classification.

#### ****3. Accuracy Metrics:****

* Calculation of the model's accuracy based on test data (e.g., phishing and legitimate URLs).
* Display of the accuracy and confusion matrix to evaluate the model's performance in classifying phishing URLs correctly.

#### ****4. Visual Representation:****

* Use **matplotlib** and **seaborn** to create a visual representation of the confusion matrix.
* Display the confusion matrix as an image file on the web interface for easier interpretation by the user.

### ****Technical Components:****

#### ****1. Backend:****

* **Flask** framework for web application development.
* **Scikit-learn** for machine learning tasks, including URL feature extraction and model prediction.
* **Pickle** for storing and loading the pre-trained model.
* **Matplotlib** and **Seaborn** for generating and displaying the confusion matrix.

#### ****2. Frontend:****

* **HTML** templates rendered by **Flask** to create a user-friendly interface.
* Forms for URL input submission and displaying prediction results.

#### ****3. Data Handling:****

* Reading and processing of URLs submitted by users for phishing detection.
* Extracting features (e.g., domain name, URL length, HTTPS status, suspicious symbols).
* Preprocessing URL features to be input into the machine learning model.

### ****Workflow:****

#### ****1. Data Preparation:****

* Extract relevant features from the input URL (e.g., domain name, HTTPS usage).
* Ensure the model is ready to classify the URL with the preprocessed data.

#### ****2. Model Prediction:****

* Input the extracted features into the pre-trained **Gradient Boosting Classifier** model.
* The model predicts whether the URL is **Safe** or **Phishing**, along with a confidence score.

#### ****3. Metrics Calculation:****

* Use the model to evaluate test data (URLs labeled as phishing or safe).
* Calculate the **accuracy** of the model and generate a **confusion matrix** for model evaluation.

#### ****4. Result Display:****

* Display the classification result (Safe or Phishing) on the web interface.
* Show the confidence score and provide a confusion matrix for users to evaluate the model's performance.

### ****Limitations and Assumptions:****

* **Model Dependency**: The accuracy of the phishing detection is reliant on the quality of the pre-trained model and the features extracted from the URLs.
* **Language Limitation**: The system assumes the URLs are in standard text format and does not handle multi-language URLs or highly obfuscated phishing URLs.
* **Whitelist Dependence**: Trusted domains (e.g., educational, governmental) are whitelisted to prevent false positives, but this may miss sophisticated phishing domains.
* **Pre-trained Model**: The system depends on the availability of the pre-trained machine learning model (model.pkl), which must be updated regularly to detect new phishing tactics.

### ****Future Enhancements:****

* **Improve the Model**: Train the model on a larger and more diverse dataset of phishing and legitimate URLs to improve detection accuracy.
* **Multi-Language Support**: Add support for URLs in different languages, allowing the system to detect phishing across global domains.
* **User Interface Enhancement**: Improve the user interface with more advanced features, such as real-time suggestions and browser extensions for phishing detection.
* **Mobile App Support**: Develop a mobile application for users to easily check URLs on their smartphones.
* **Real-Time Updates**: Integrate with external databases or threat intelligence sources for real-time phishing URL detection and updates to the model.

This scope outlines the main functionalities, technical components, and potential future improvements for the **Phishing URL Detection System**, ensuring a reliable, scalable, and user-friendly solution to combat phishing threats.

**4.2 Risk Management**

**4.2.1 Risk Identification**

Technical Risks:

1. Integration Challenges
2. Scalability Issues
3. Data Security Concerns

Project Management Risks: 4. Scope Creep 5. Resource Constraints 6. Schedule Overruns

External Risks: 7. Third-Party Service Dependency 8. Regulatory Changes

**4.2.2 Risk Analysis and Mitigation**

Potential Risks and Mitigation:

1. Integration Challenges
   * Allocate extra time for research and development.
   * Seek external expertise if needed.
2. Scope Creep
   * Establish a change control process.
   * Conduct impact assessments for proposed changes.
3. Resource Constraints
   * Cross-train team members.
   * Maintain a buffer of additional resources.

**4.2.3 Risk Management Plan**

* Identify Risks: Continuously identify potential risks throughout the project lifecycle.
* Assess Risks: Evaluate the likelihood and impact of identified risks.
* Plan Responses: Develop mitigation strategies and contingency plans for high-priority risks.
* Monitor and Control: Regularly monitor risks and implement response plans as needed.

**4.3 Work Breakdown Structure (WBS)**

Level 1: Project Phases

1. Project Initiation and Planning
2. Requirements Analysis
3. System Design
4. Development
5. Testing and Quality Assurance
6. Deployment and Final Review

Level 2: Tasks *(Example tasks under each phase)*

1. Project Initiation and Planning
   * Define project scope
   * Identify stakeholders
   * Create project plan
   * Develop schedule
   * Set up development environment
2. Requirements Analysis
   * Gather functional requirements
   * Document non-functional requirements
   * Create use case diagrams
   * Review and approve requirements
3. System Design
   * Design system architecture
   * Create ER diagrams
   * Develop data flow diagrams
   * Design front-end components
   * Design back-end components
4. Development
   * Implement user authentication
   * Develop chat room management features
   * Integrate real-time messaging functionality
5. Testing and Quality Assurance
   * Perform unit testing
   * Conduct integration testing
   * Execute user acceptance testing
   * Fix bugs and optimize performance
6. Deployment and Final Review
   * Deploy system to production
   * Conduct final review
   * Handover project to stakeholders

**Chapter 5: Features and Output**

### ****Features and Output of the Phishing URL Detection System****

The **Phishing URL Detection System** offers a set of essential features designed to efficiently detect phishing URLs and provide accurate classification results:

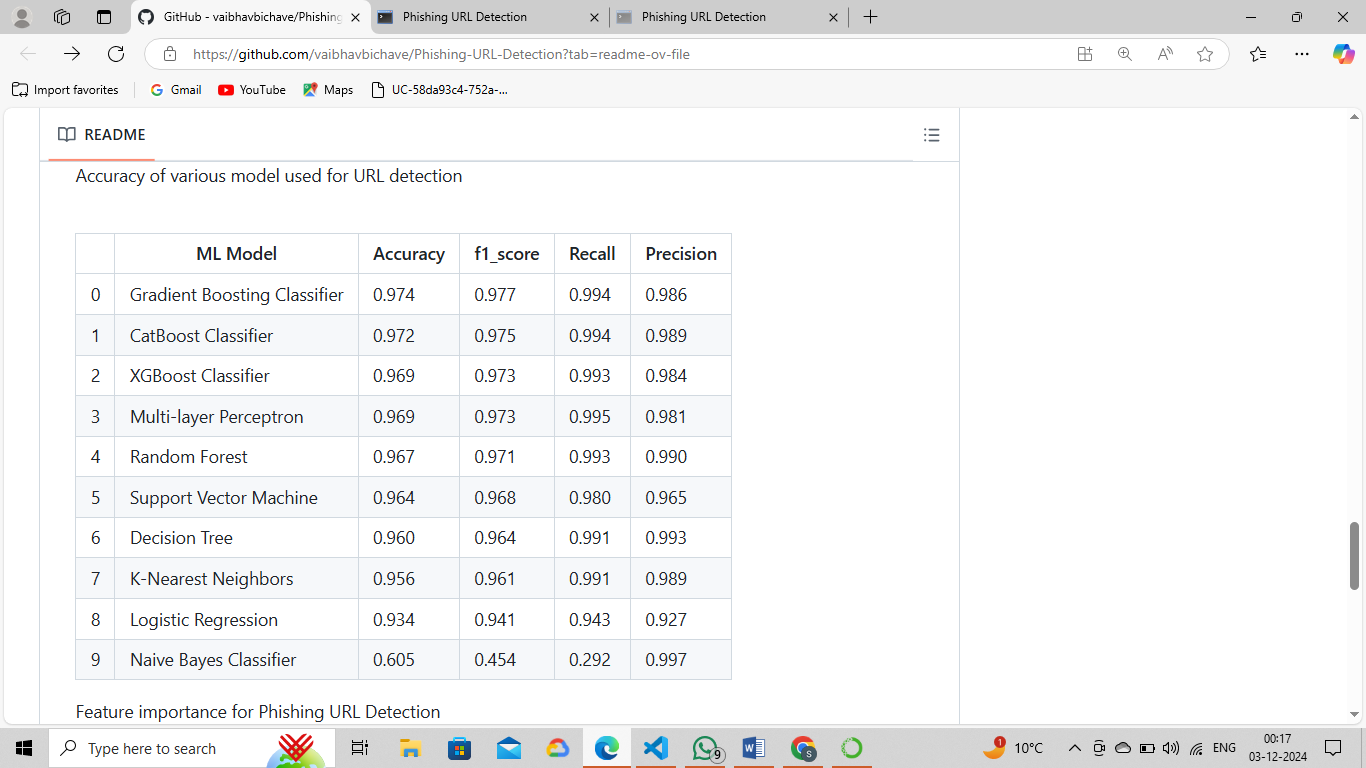
* **URL Input and Submission**: Users can input URLs into the web interface to be analyzed for phishing attempts.
* **Feature Extraction**: The system extracts key features from the URL, such as domain name, URL length, use of HTTPS, presence of suspicious symbols, subdomain count, and WHOIS registration data.
* **Whitelisted Domain Check**: The system checks whether the submitted URL belongs to a trusted, pre-defined whitelist of domains. If it matches, the URL is automatically classified as safe.
* **Machine Learning Classification**: If the URL is not whitelisted, it is passed to the machine learning model (Gradient Boosting Classifier) for classification. The model uses the extracted features to predict whether the URL is safe or phishing.
* **Confidence Scoring**: The system generates a confidence score, indicating the certainty of the classification result. A higher score reflects greater confidence in the URL’s classification.
* **Real-Time Processing**: URLs are analyzed and classified in real-time, providing users with quick results.
* **Logging and Reporting**: All URL submissions and their results are logged for future auditing, retraining of the model, and system performance monitoring.
* **User-Friendly Interface**: The web-based interface is simple and intuitive, allowing users to easily input URLs and view the results without requiring technical knowledge.

### ****Output****:

* The system provides a clear classification result of the submitted URL, indicating whether it is **Safe** or **Phishing**.
* Along with the classification, the system displays a **confidence score** (e.g., 85% confident the URL is safe).
* A **confusion matrix** can be generated and displayed to show the model’s performance in distinguishing between phishing and safe URLs.
* The results are displayed promptly on the user interface, allowing users to take immediate action based on the classification.

This system provides users with a reliable and efficient method for phishing URL detection, leveraging machine learning to offer high accuracy and confidence in the results.

**Results**



#### 

Figure 5.1

**Chapter 6: Summary and Future Scope**

#### Summary

The **Phishing URL Detection System** serves as a crucial tool in protecting users from malicious phishing websites, offering an advanced platform to detect and classify potentially harmful URLs. By leveraging machine learning algorithms and feature extraction techniques, the system provides reliable classifications while ensuring a user-friendly experience. Key functionalities include URL submission, real-time analysis, model integration, and continuous improvement through feedback.

### ****Key Features****

1. **URL Submission and Analysis:**
   * Users can easily submit URLs for analysis, triggering the system to assess whether the URL is safe or a phishing attempt.
   * The system extracts relevant features from the URL (e.g., domain name, URL length, suspicious symbols) and classifies it using a pre-trained machine learning model.
2. **User Interface:**
   * The intuitive web-based interface allows users to input URLs for instant phishing detection and receive detailed results.
   * The interface provides the classification result (Safe/Phishing) along with confidence scores, ensuring transparency and user engagement.
3. **Model Management:**
   * The system integrates a pre-trained **Gradient Boosting Classifier** model, trained on a large dataset of phishing and legitimate URLs for accurate classification.
   * The model is continuously improved by retraining with new data, ensuring it adapts to emerging phishing techniques and threats.
4. **Logging and Reporting:**
   * Each URL submission, classification result, and confidence score is logged for auditing, tracking, and improving model accuracy over time.
   * Users are able to view a history of their submissions for easy reference.

### ****Development Process****

1. **Backend Development:**
   * **Flask** serves as the backend framework, handling user requests, feature extraction, and classification.
   * The **Scikit-learn** library is used for machine learning tasks, including model training, feature extraction, and classification.
   * **Pickle** is used to store and load the pre-trained machine learning model.
2. **Frontend Development:**
   * The web interface is developed using **HTML**, **CSS**, and **JavaScript**, providing a clean and user-friendly experience.
   * **Flask** renders the HTML templates to display the results of the phishing detection process.

### ****Testing and Maintenance****

#### ****Testing Plan****

1. **Unit Testing:**
   * Test individual components, such as model loading, URL feature extraction, and prediction functions, to ensure they work as expected.
2. **Integration Testing:**
   * Test the integration between the front-end and back-end to verify smooth interaction between components (e.g., form submissions, result generation).
3. **End-to-End Testing:**
   * Simulate user interactions and test various scenarios (e.g., valid/invalid URLs) to ensure the system behaves as expected.
4. **Performance Testing:**
   * Measure the system’s response time and scalability under load to ensure reliable performance with many simultaneous users.
5. **Security Testing:**
   * Conduct security audits and test for vulnerabilities (e.g., XSS, CSRF) to protect against malicious attacks.

#### ****Maintenance Plan****

1. **Regular Updates:**
   * Keep the system updated with security patches, bug fixes, and library updates to maintain reliability and security.
2. **Model Maintenance:**
   * Monitor the quality of the data and retrain the model periodically with new phishing data to improve its accuracy and adapt to evolving threats.
3. **Monitoring and Error Handling:**
   * Implement performance monitoring tools to track system health and set up alerts for critical issues. Log errors for troubleshooting and improve system resilience.
4. **Backup and Recovery:**
   * Ensure regular backups of system data and configurations to prevent data loss, and verify recovery procedures.
5. **User Feedback and Iterative Improvement:**
   * Collect user feedback to identify areas for improvement, and iterate on the application to enhance user experience and model performance.

This project provides a scalable, efficient, and user-friendly solution for detecting phishing URLs, leveraging advanced machine learning models and continuous user engagement to enhance its capabilities over time. The system ensures a high level of accuracy, security, and adaptability, making it an essential tool for online safety.

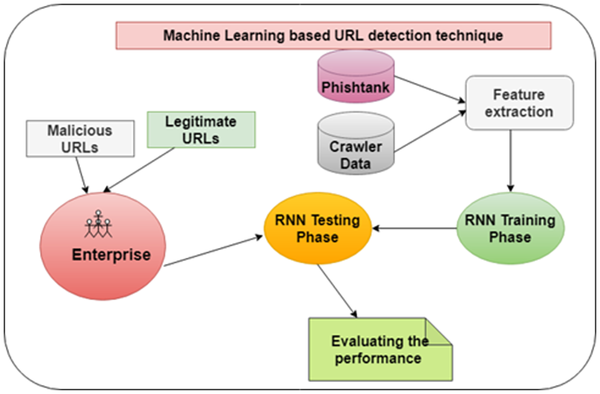


Figure 6.1

**Chapter 7 Methodology**

The **Phishing URL Detection System** employs a systematic methodology to ensure accurate and efficient detection of phishing URLs. The process is divided into several key stages, each designed to address a specific aspect of the system's functionality, from data collection to prediction and result interpretation.

**7.1 Dataset**

### Dataset for Phishing URL Detection System

The dataset plays a critical role in training and validating the machine learning model for detecting phishing URLs. Below is an outline of the dataset structure, sources, and key components:

### ****1. Dataset Overview****

* The dataset should contain URLs labeled as either **Phishing** or **Legitimate (Safe)**.
* It must be comprehensive, covering various types of phishing URLs to improve the model's robustness.
* The dataset can be divided into two subsets:
  + **Training Set**: Used to train the machine learning model.
  + **Testing Set**: Used to evaluate the model's accuracy and performance.

### ****2. Data Sources****

* **Phishing URLs**:
  + Public phishing URL repositories like [PhishTank](https://www.phishtank.com/) or [OpenPhish](https://openphish.com/).
  + Datasets from cybersecurity research platforms or competitions (e.g., Kaggle, UCI Machine Learning Repository).
* **Legitimate URLs**:
  + Trusted websites scraped from directories like Alexa Top Sites.
  + Educational, governmental, and commercial websites from whitelists.

### ****3. Dataset Structure****

The dataset should be in a **CSV** or **JSON** format with the following columns:

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| URL | The full URL (e.g., https://example.com/login) |
| Label | The classification label: Phishing or Legitimate |
| Domain | The domain name of the URL (e.g., example.com) |
| URL\_Length | The length of the URL in characters |
| HTTPS\_Status | Indicates if the URL uses HTTPS (e.g., 1 for Yes, 0 for No) |
| Suspicious\_Symbols | Presence of symbols like @ or - (1 for Yes, 0 for No) |
| Subdomains | The count of subdomains in the URL |
| WHOIS\_Registration | Age of domain registration (in months or years) |
| Non\_Standard\_Port | Indicates the use of non-standard ports (1 for Yes, 0 for No) |
| Favicon\_Linked | Whether the favicon links to an external source (1 for Yes, 0 for No) |

Figure 7.1

### ****4. Dataset Split****

* **Training Set**: 80% of the data for training the machine learning model.
* **Testing Set**: 20% of the data for evaluating model performance.
* **Validation Set (Optional)**: 10% of the training data for hyperparameter tuning and avoiding overfitting.

### ****5. Dataset Requirements****

* **Balanced Classes**: Ensure the dataset contains a roughly equal number of phishing and legitimate URLs to avoid bias in the model.
* **Diversity**: Include a wide variety of phishing tactics and URL structures to improve the model’s generalizability.
* **Label Accuracy**: Labels (Phishing/Legitimate) should be verified to ensure the dataset’s reliability.

### ****6. Dataset Preprocessing****

* **URL Cleaning**: Remove duplicate entries and irrelevant data.
* **Feature Engineering**:
  + Extract key features like URL length, domain age, HTTPS status, and suspicious symbols.
  + Encode categorical data into numerical format for machine learning compatibility.
* **Normalization**: Scale features like URL length and WHOIS registration age to a standard range (e.g., 0 to 1).

### ****7. Tools for Dataset Handling****

* **Python Libraries**:
  + pandas for data manipulation.
  + numpy for numerical operations.
  + scikit-learn for splitting the dataset into training and testing sets.
* **Dataset Sources**:
  + Kaggle: [Phishing Website Dataset](https://www.kaggle.com/competitions/phishing-websites/data).
  + UCI Repository: Phishing dataset with labeled URLs.

**7.2 Data Preprocessing**

Data preprocessing is a critical step in the development of the **Phishing URL Detection System**, as it ensures the data is clean, structured, and ready for feature extraction and model training. Below is the step-by-step process for data preprocessing:

### ****1. Load the Dataset****

* Use Python libraries like pandas to load the dataset (usually in CSV or JSON format).
* Example:
* import pandas as pd
* data = pd.read\_csv('phishing\_dataset.csv')

### ****2. Handle Missing Values****

* Check for missing or null values in the dataset.
* Impute missing values where applicable or remove rows/columns with excessive missing data.
* Example:
* data.isnull().sum() # Check for missing values
* data = data.dropna() # Drop rows with missing values

### ****3. Remove Duplicates****

* Identify and remove duplicate entries to ensure unique data points.
* Example:
* data = data.drop\_duplicates()

### ****4. Extract Features****

* Extract key features from the URLs for analysis and classification:
  + **Domain Name**: Extract the main domain.
  + **URL Length**: Calculate the total length of the URL.
  + **Subdomains**: Count the number of subdomains in the URL.
  + **Suspicious Symbols**: Check for symbols like @, -, or //.
  + **HTTPS Status**: Check if the URL uses HTTPS.
  + **WHOIS Data**: Retrieve domain registration details (e.g., age, expiration).
* Example:
* from urllib.parse import urlparse
* def extract\_features(url):
* parsed\_url = urlparse(url)
* return {
* 'Domain': parsed\_url.netloc,
* 'URL\_Length': len(url),
* 'Subdomains': parsed\_url.netloc.count('.'),
* 'Has\_HTTPS': int(parsed\_url.scheme == 'https'),
* 'Has\_Symbols': int(any(c in url for c in ['@', '-', '//']))
* }
* data['Features'] = data['URL'].apply(extract\_features)

### ****5. Encode Labels****

* Convert the target labels (e.g., "Phishing" and "Legitimate") into numerical values for model compatibility.
* Example:
* data['Label'] = data['Label'].map({'Phishing': 1, 'Legitimate': 0})

### ****6. Normalize Features****

* Scale numerical features to a uniform range (e.g., 0 to 1) to improve model performance.
* Use libraries like scikit-learn for normalization.
* Example:
* from sklearn.preprocessing import MinMaxScaler
* scaler = MinMaxScaler()
* numeric\_features = ['URL\_Length', 'Subdomains', 'WHOIS\_Age']
* data[numeric\_features] = scaler.fit\_transform(data[numeric\_features])

### ****7. Handle Categorical Features****

* Convert categorical features (e.g., Domain, Has\_HTTPS) into numerical or encoded values.
* Use one-hot encoding or label encoding as appropriate.
* Example:
* data = pd.get\_dummies(data, columns=['Domain'], drop\_first=True)

### ****8. Split the Dataset****

* Divide the dataset into training, testing, and optionally validation sets.
* Use an 80-20 split for training and testing.
* Example:
* from sklearn.model\_selection import train\_test\_split
* X = data.drop('Label', axis=1) # Features
* y = data['Label'] # Target
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

### ****9. Save the Processed Dataset****

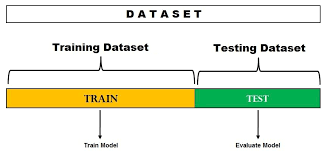
* Save the preprocessed data for later use in training and evaluation.
* Example:
* data.to\_csv('preprocessed\_dataset.csv', index=False)

### ****10. Validate the Data****

* Perform exploratory data analysis (EDA) on the processed dataset to ensure the features are correctly extracted and balanced.
* Check the distribution of labels to avoid class imbalance issues.

**7.3 Train-Test Split of Dataset**

The train-test split divides the dataset into training and testing sets, typically in an 80-20 ratio. The training set, comprising 80% of the data, is used to train the model, while the remaining 20% is reserved for evaluating model performance.



**Figure 7.2**

**7.4 Feature Extraction**

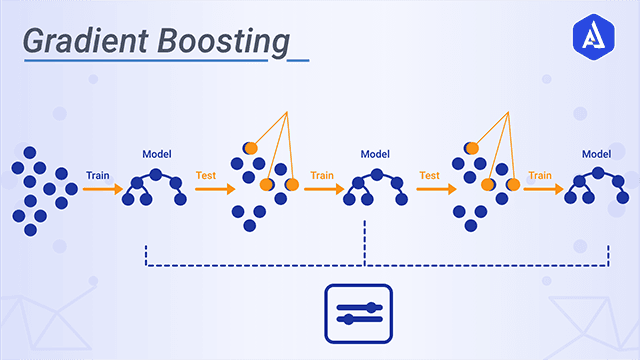
Feature extraction involves deriving new features from existing ones, relevant to the model's objectives. This process eliminates irrelevant data to enhance model accuracy. In this study, features are extracted to translate textual data into numerical values for training the classifier.

### 7.6 ****Gradient Boosting Classifier****

The Gradient Boosting Classifier (GBC) is a powerful ensemble machine learning algorithm widely used for classification tasks. It builds a robust model by combining multiple weak learners, typically decision trees, to iteratively improve performance. Its flexibility and high accuracy make it a preferred choice for solving problems like phishing URL detection.

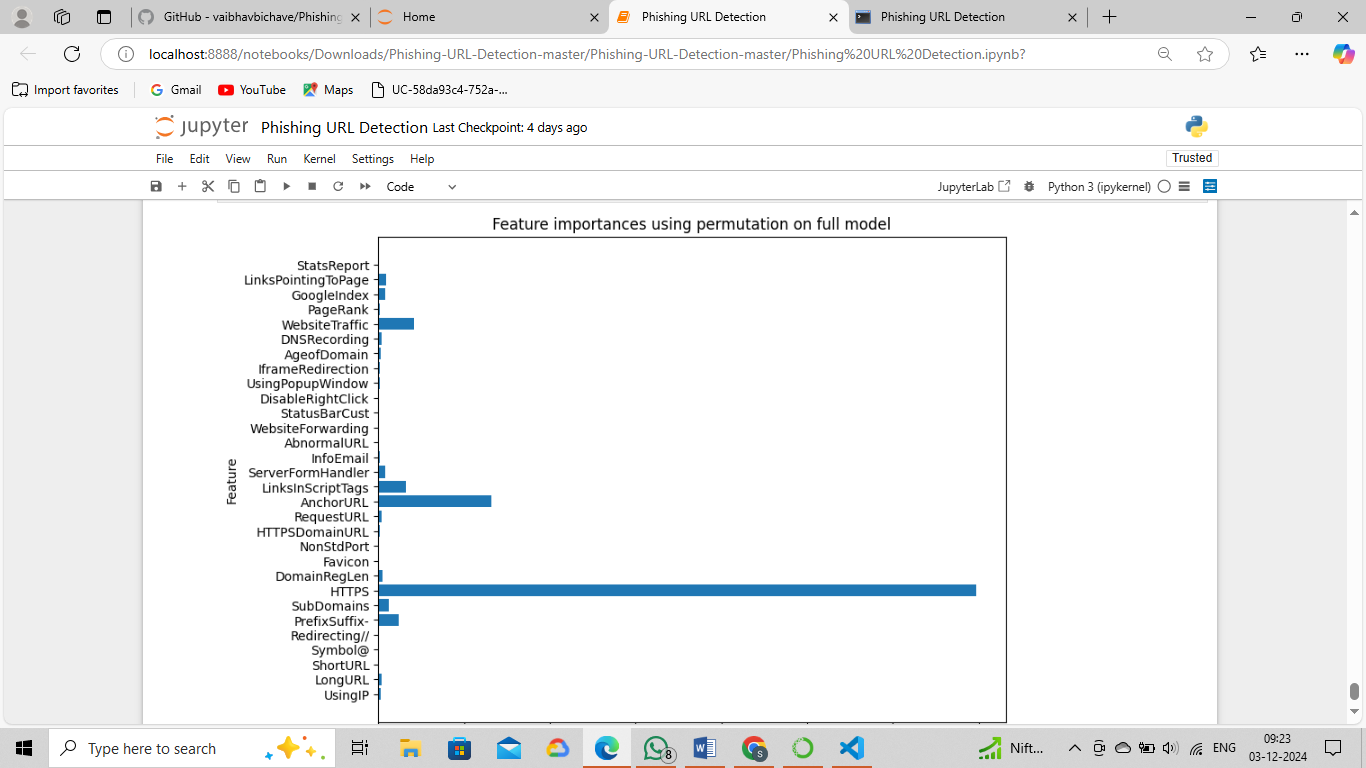
1. **Ensemble Learning**:  
   The Gradient Boosting Classifier uses an ensemble learning approach, where multiple weak learners (e.g., shallow decision trees) are combined to form a strong predictive model. Each weak learner corrects the errors of its predecessor, resulting in a progressively better model.
2. **Boosting Technique**:  
   GBC employs a boosting technique to sequentially train models. At each iteration, a new model is trained to minimize the errors made by the previous models. This iterative process improves the overall prediction accuracy.
3. **Handling Imbalanced Data**:  
   The algorithm is well-suited for imbalanced datasets, where phishing URLs might be far fewer than legitimate ones. By focusing on hard-to-classify examples, GBC effectively balances predictions across both classes.
4. **Feature Importance**:  
   GBC provides insights into feature importance, allowing the identification of which URL features (e.g., domain name, URL length, HTTPS usage) are most critical for detecting phishing. This interpretability helps refine feature engineering for better results.
5. **Flexibility**:  
   The Gradient Boosting Classifier can handle various types of data and is adaptable to different feature spaces, making it highly suitable for complex tasks like phishing detection, where feature interactions are significant.
6. **Regularization**:  
   Regularization techniques like shrinkage (learning rate) and subsampling are incorporated into GBC to prevent overfitting. These methods control the complexity of the model and improve its generalization performance.
7. **Scalability**:  
   Despite its iterative nature, GBC is computationally efficient and scales well with large datasets. Parallelization techniques, like those in XGBoost or LightGBM implementations, further enhance its scalability.
8. **Multi-class and Binary Classification**:  
   While Gradient Boosting Classifier is often used for binary classification tasks like phishing URL detection, it can also be extended to handle multi-class problems, making it versatile for various domains.
9. **Hyperparameter Tuning**:  
   GBC requires careful tuning of hyperparameters, such as the number of estimators, learning rate, and maximum depth of trees, to achieve optimal performance. Grid search or random search techniques are commonly used for this purpose.
10. **Applications in Phishing Detection**:  
    The Gradient Boosting Classifier is particularly suited for phishing detection as it can handle tabular data with extracted features (e.g., URL length, subdomain count, suspicious symbols). Its ability to learn complex decision boundaries ensures high detection accuracy and low false positives.

By leveraging these characteristics, the Gradient Boosting Classifier offers an efficient and reliable approach to phishing URL detection, making it a valuable algorithm in the cybersecurity domain. Its ensemble-based learning framework, interpretability, and adaptability ensure robust performance even in challenging real-world scenarios.



**Figure 7.3**

**7.7 Accuracy Score**

The accuracy score measures the proportion of correct predictions in the test dataset, providing insight into model performance compared to previous iterations.

**Figure 7.4**

**Chapter 8 Working and Discussion**

**8.1 Libraries Used:**

1. **Data Manipulation**:
   * **pandas**: For handling datasets, cleaning, and transforming data.
   * **numpy**: For numerical computations and feature scaling.
2. **Feature Extraction**:
   * **re**: To detect patterns in URLs (e.g., suspicious symbols like @, -).
   * **whois**: To fetch domain registration details (e.g., age, expiration).
   * **urllib.parse**: To extract URL components like domain and subdomains.
   * **beautifulsoup4**: For scraping HTML elements like favicons.
3. **Machine Learning**:
   * **scikit-learn**: For training and evaluating models (e.g., Gradient Boosting Classifier) and metrics calculation.
   * **pickle/joblib**: For saving and loading the pre-trained machine learning model.
4. **Data Visualization**:
   * **matplotlib**: For creating static visualizations (e.g., bar charts).
   * **seaborn**: For advanced plots (e.g., heatmaps for confusion matrices).
5. **Web Development**:
   * **Flask**: To build the web application for URL input and result display.
6. **Others**:
   * **requests**: For fetching URL content.
   * **os**: For managing file paths and environment variables.

These libraries streamline the entire project workflow, from data preprocessing and model training to real-time URL classification and result visualization.

**8.2 Workflow:**

1. **User Input**:
   * The user submits a URL through a web interface for analysis.
2. **Feature Extraction**:
   * Extract key features from the URL, such as domain name, URL length, presence of HTTPS, suspicious symbols, subdomain count, and WHOIS data.
3. **Whitelist Check**:
   * Check if the URL's domain is in a trusted whitelist. If found, classify the URL as **Safe** directly.
4. **Model Prediction**:
   * If not whitelisted, pass the extracted features to the pre-trained **Gradient Boosting Classifier**.
   * The model classifies the URL as **Safe** or **Phishing** and provides a confidence score.
5. **Result Display**:
   * Show the classification result (Safe/Phishing) and confidence score on the web interface.
6. **Logging**:
   * Log the URL, extracted features, classification result, and confidence score for auditing and future analysis.
7. **Model Updating (Optional)**:
   * Periodically retrain the model with new phishing and legitimate URL datasets to enhance detection accuracy.

This workflow ensures efficient real-time phishing detection with reliable results.

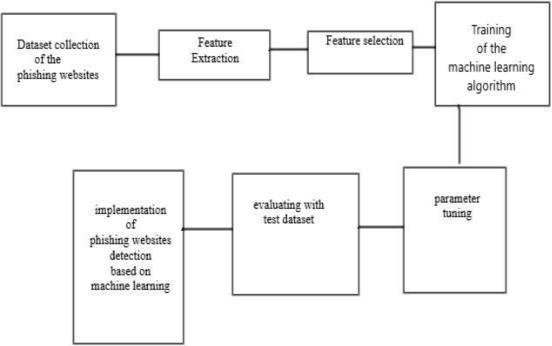


Figure 8.1

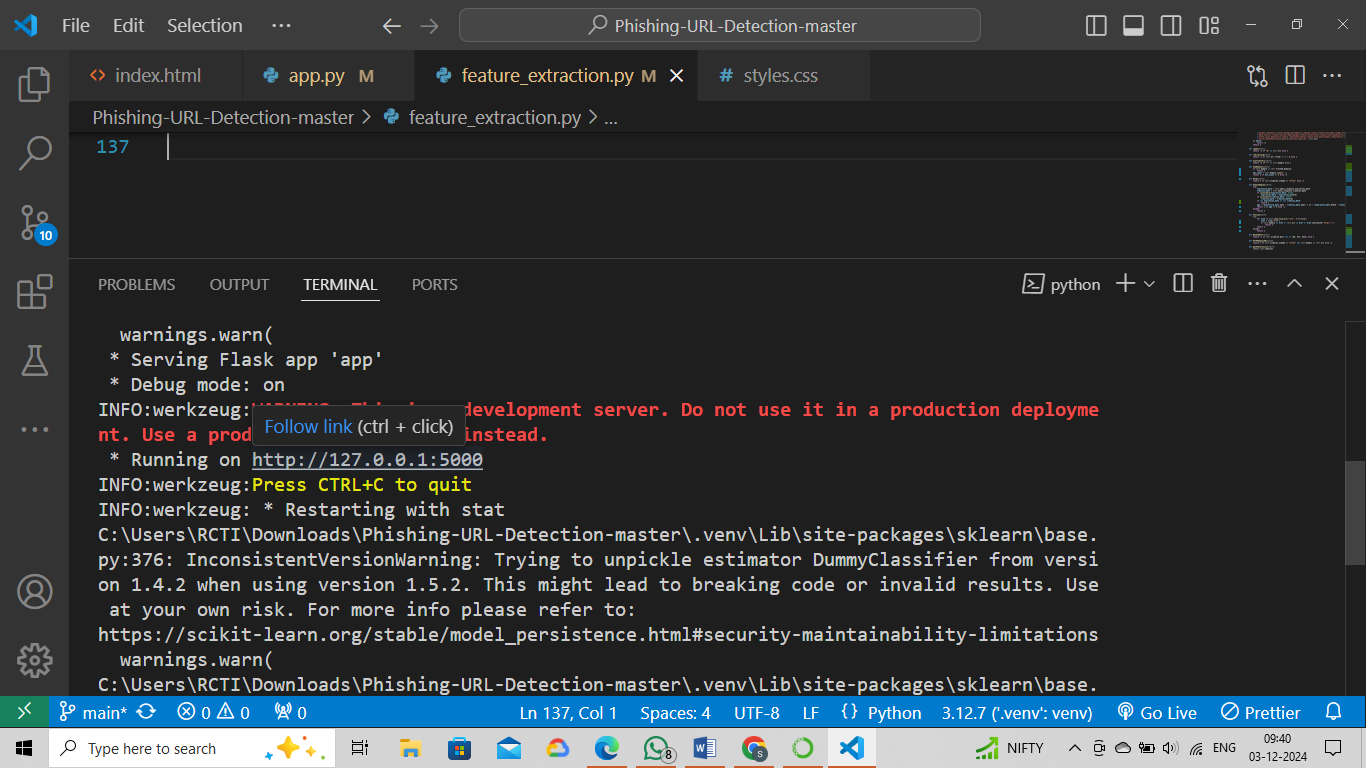


Figure 8.2

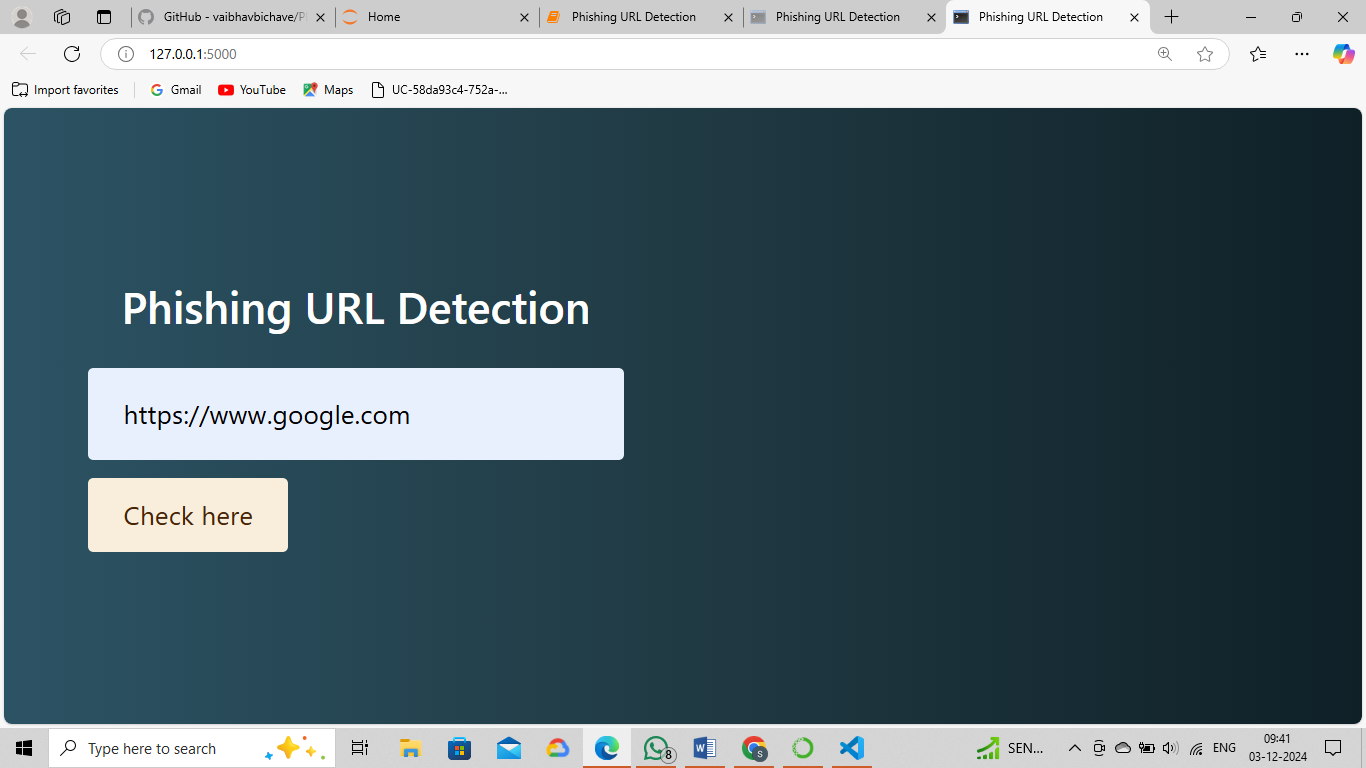


Figure 8.3

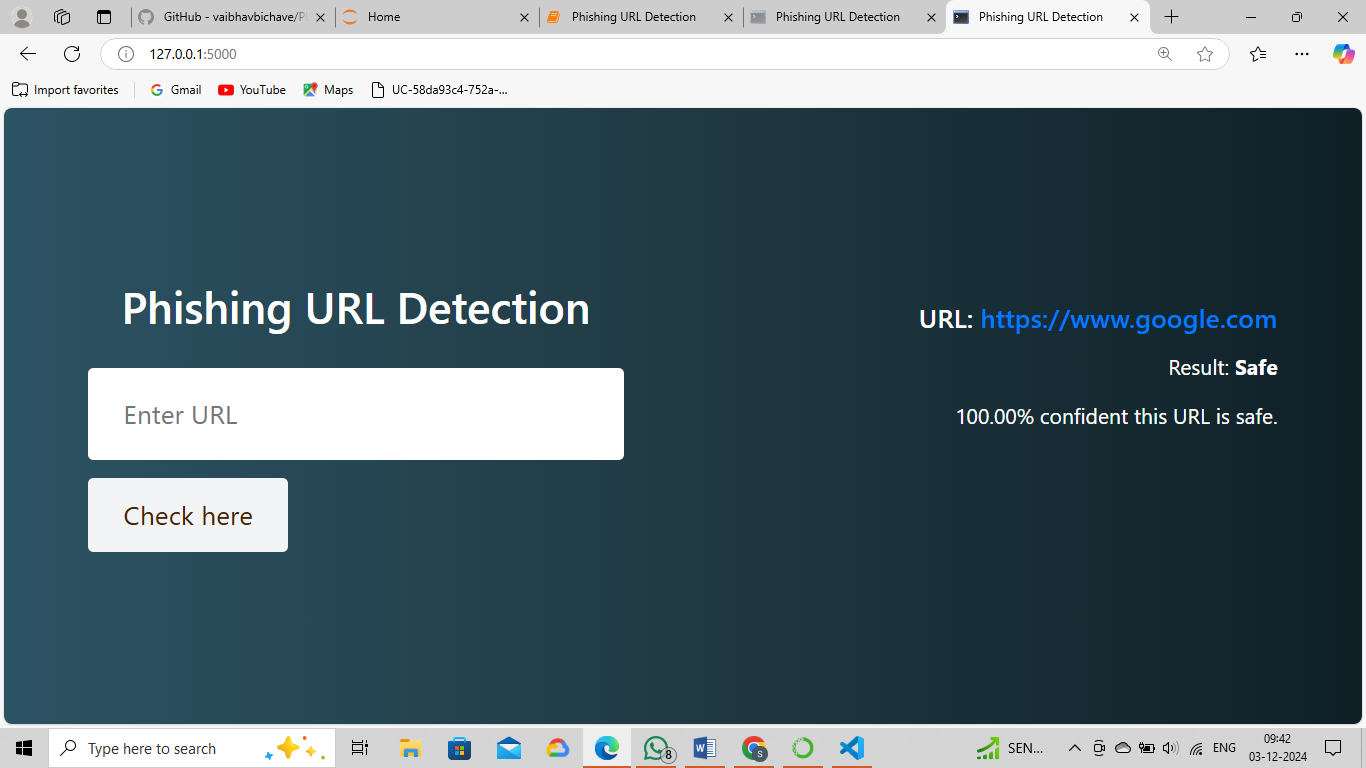


Figure 8.4

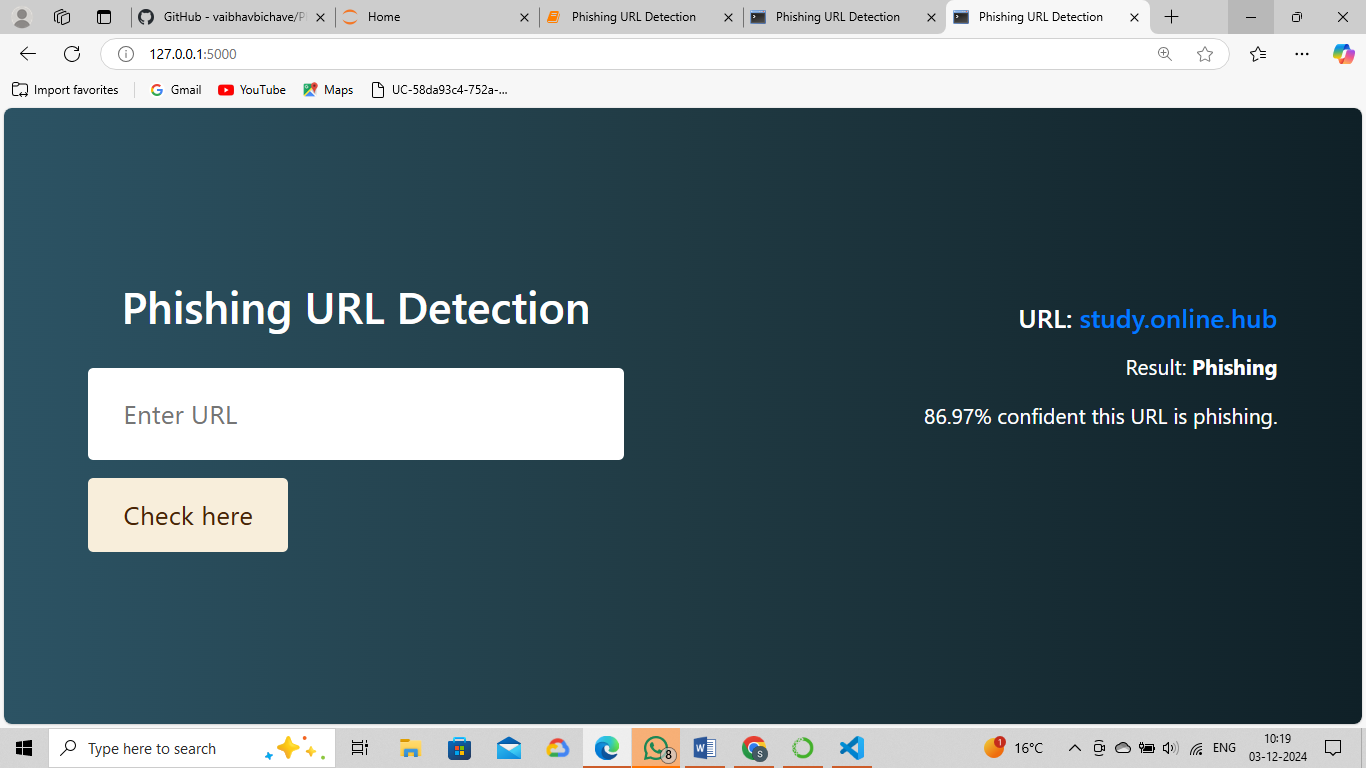


Figure 8.5

**Chapter 9: Conclusion**

The **Phishing URL Detection System** represents a significant advancement in combating phishing threats, offering users a reliable tool to identify malicious URLs in real-time. By leveraging machine learning, feature engineering, and a hybrid approach combining whitelist-based filtering with a robust **Gradient Boosting Classifier**, the system provides accurate classifications with minimal false positives. Its intuitive web interface ensures accessibility for both technical and non-technical users, making it practical for widespread adoption. The system efficiently processes URLs by extracting critical features, such as domain name, URL length, HTTPS usage, and WHOIS data, to predict whether a URL is safe or phishing. This real-time detection capability helps prevent sensitive information breaches and financial losses caused by phishing attacks.

While the system achieves high accuracy and scalability, its performance heavily relies on the quality and diversity of the dataset used for training. Addressing limitations like handling highly obfuscated phishing URLs, expanding multi-language support, and incorporating advanced deep learning models can further enhance its effectiveness. Future improvements may include developing a browser extension for real-time detection during web browsing, creating a mobile application for on-the-go URL analysis, and integrating with threat intelligence platforms for continuous updates on phishing tactics. Additionally, detailed reporting and analytics features can improve transparency, helping users understand the classification process and fostering greater trust in the system.

In conclusion, the Phishing URL Detection System offers a robust solution to one of the most pressing cybersecurity challenges. Its ability to adapt to evolving phishing techniques, coupled with its user-friendly design, positions it as a vital tool in enhancing online security. Continuous updates and expansions will ensure its relevance and effectiveness, making it a cornerstone for protecting users and organizations against phishing threats.

**Chapter 10 References**

### ****References for Phishing URL Detection System:****

* W3Schools - [*https://www.w3schools.com*](https://www.w3schools.com/)  
  A popular online resource for web development tutorials, used to understand HTML, CSS, JavaScript, and web-related technologies relevant to building the user interface of the phishing detection system.
* Geeks for Geeks - [*https://www.geeksforgeeks.org*](https://www.geeksforgeeks.org/)  
  Provides tutorials on various programming languages, algorithms, and data structures, useful for learning about machine learning, Python libraries, and web scraping techniques for phishing URL detection.
* freeCodeCamp - [*https://www.freecodecamp.org*](https://www.freecodecamp.org/)  
  Offers extensive tutorials on web development, data science, and machine learning, supporting the development of the backend, feature extraction, and integration of machine learning models.
* **PhishTank: A Repository for Phishing Data**
  + Source: [*https://www.phishtank.com*](https://www.phishtank.com/)  
    A widely-used repository for phishing URLs, providing data for training and testing phishing detection models.
* **Google Scholar: A Platform for Research Papers on Phishing Detection and URL Classification**
  + Source: [*https://scholar.google.com*](https://scholar.google.com/)  
    Provides access to a wide range of scholarly articles and papers on phishing detection, machine learning models, and cybersecurity methods.
* **Phishing URL Detection Using Machine Learning Techniques**
  + Source: [*https://www.sciencedirect.com/science/article/pii/S2352146518302992*](https://www.sciencedirect.com/science/article/pii/S2352146518302992)  
    A paper discussing various machine learning algorithms and features used in phishing URL detection, helping to support the feature-based approach of the system.

.