A

Project Phase-II Report on

Enhanced Phishing Website Detection using Machine Learning Algorithm

Submitted in the partial fulfillment of the requirements
For the Award of the Degree of Bachelor of Technology (B.Tech) in
Electronics and Communication Engineering

Submitted by

Satyam Tiwari	**PRN: 2114110531**
Sumit Verma	**PRN: 2114110536**
Atharva Yelne	**PRN: 2114110539**

Under the Guidance of

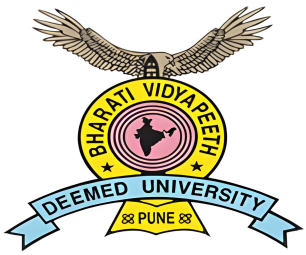
Dr. M.V. PATIL

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BHARATI VIDYAPEETH DEEMED TO BE UNIVERSITY

COLLEGE OF ENGINEERING, PUNE - 411043

ACADEMIC YEAR: 2024-2025



Department of Electronics and Communication Engineering
BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)
COLLEGE OF ENGINEERING, PUNE

CERTIFICATE

This is to certify that the project phase-I report on **"Enhanced Phishing Website Detection using Machine Learning Algorithm"** submitted by

Satyam Tiwari	**PRN: 2114110531**
Sumit Verma	**PRN: 2114110536**
Atharva Yelne	**PRN: 2114110539**

in partial fulfillment of the requirements for the award of degree of Bachelor of Technology (B.Tech) in Electronics and Communication Engineering.

Dr. M.V. PATIL
Guide, ECE Dept.,
BV(DU), COE, Pune

Dr. Dhiraj M. Dhane
Project Co-ordinator,
BV(DU), COE, Pune

Prof.(Dr.) Arundhati A. Shinde
Head, ECE Department
BV(DU), COE, Pune

Date:
Place: Pune

Abstract

Phishing attacks continue to pose a significant threat to cybersecurity, deceiving users into revealing sensitive information by mimicking legitimate websites. This project focuses on enhancing phishing website detection using a range of machine learning algorithms. The dataset was compiled from trusted open-source platforms such as PhishTank and the UCI Machine Learning Repository. After thorough preprocessing—including handling missing values and encoding categorical features—critical indicators like URL length, presence of special characters, security certificates, and domain age were extracted. Feature selection techniques were employed to improve model performance and reduce complexity.

Multiple machine learning models were trained and evaluated, including Gradient Boosting, CatBoost, XGBoost, Multi-layer Perceptron, Random Forest, Support Vector Machine, Decision Tree, K-Nearest Neighbors, Logistic Regression, and Naïve Bayes. Among them, Gradient Boosting achieved the highest F1-score of 0.977, with an accuracy of 0.974 and a recall of 0.994, indicating excellent detection capabilities. Other ensemble-based models like CatBoost and XGBoost also performed exceptionally well, outperforming traditional models like Logistic Regression and Naïve Bayes.

Model performance was assessed using standard metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. To facilitate practical use, a command-line or web-based application was developed to provide real-time phishing detection. For future work, the integration of deep learning models such as Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs), as well as browser extension or API-based deployment, is planned to further strengthen detection capability and ease of access.

Acknowledgments

We would like to express our gratitude Dr. M.V. Patil, our mentor, for providing invaluable guidance throughout the project. We would also like to extend our thanks to Prof. (Dr.) Arundhati A. Shinde, the Head of our Department, for their support. Additionally, we thank our department and institution for giving us the opportunity to undertake this project.

Our sincere appreciation extends to all those who provided datasets, tools, and technical assistance that were vital for our project.

Satyam Tiwari
Sumit Verma
Atharva Yelne

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

As the Internet continues to expand, many of our activities, such as e-commerce, business, social networking, and banking, are increasingly conducted online. This shift heightens the risk of online crime, making web security ever more essential. In 2020, approximately 237,418,349 Internet users were Arabic-speaking, according to Internet WorldStats [1] To combat this threat, the online security community has introduced blacklisting services to help detect harmful websites. Blacklists are databases containing URLs that are already recognized as malicious, and in some cases, blacklisting has proven effective [2]. However, attackers can bypass this by modifying parts of the URL, and many malicious sites avoid blacklisting by being either too new or inaccurately assessed. An alternative approach to detecting ma licious URLs is the heuristic method. This technique improves upon blacklisting by using signature-based analysis to find correlations between new URLs and the signatures of known malicious URLs. While both blacklisting and heuristic methods help identify malicious and benign URLs, they have limitations. For instance, (a) blacklisting struggles against zero-hour phishing attacks, identifying only 47–83 new phishing URLs within a 12-hr window, and thus fails to categorize all new URLs; and (b) both methods can be bypassed through obfuscation techniques, such as us ing algorithms to generate large numbers of URLs that evade detection [3] [4].Due to these limitations, blacklisting alone cannot keep up with the rapid pace of changing technology.

1.2 Problem Statement

Phishing attacks pose a significant cybersecurity threat, tricking users into divulging sensitive information through fraudulent websites. Traditional rule-based detection methods struggle to keep up with evolving phishing techniques. This project aims to enhance phishing website detection using machine learning algorithms that analyze website features and patterns, improving accuracy and adaptability. The goal is to develop a robust, automated system to identify and mitigate phishing threats effectively

1. **Enhanced Phishing Detection with Machine Learning**
    - Leverages machine learning to analyze URL patterns and characteristics.
    - Accurately identifies phishing attempts to improve security posture.
    - Reduces false positives to prevent legitimate websites from being mistakenly flagged.
2. **Real-Time Threat Detection**
    - Swift identification and mitigation of phishing threats.
    - Provides timely protection against evolving cyber risks.
3. **Scalability and Adaptability**
    - Designed to efficiently handle large volumes of URLs.
    - Continuously learns from new data and attack patterns to adapt to changing threats.
4. **Seamless Integration with Cybersecurity Frameworks**
    - Ensures compatibility with existing security infrastructure.
    - Allows organizations to adopt the system without disruptions.
5. **User-Friendly Interface**
    - Enables security teams to monitor and manage detected phishing threats effectively.
6. **Regular Updates for Evolving Threats**
    - Keeps pace with new phishing tactics to maintain effectiveness.

7. **Ethical Guidelines and Privacy Compliance**
    - Ensures trust and compliance when handling user data.
8. **Comprehensive Testing and Evaluation**
    - Assesses system performance to ensure reliability and effectiveness in combating phishing attacks.

1.3 Need of the project

The need of the project is to develop a machine learning-based system for detecting malicious URLs, addressing the growing concern of cyber threats and attacks spread through malicious URLs. The system will use machine learning algorithms to classify URLs as either malicious or benign, and will be able to detect new and unknown malicious URLs. This need arises from the increasing number of cyber threats, limitations of traditional methods, and the importance of cybersecurity. Traditional methods of detecting malicious URLs, such as blacklisting, have limitations and are not effective in detecting new and unknown malicious URLs. There is a need for an efficient and effective system that can detect malicious URLs in real-time, and prevent cyber attacks. Cybersecurity is a critical aspect of modern life, and the detection of malicious URLs is an important part of it. The project has the potential to make a significant impact in the field of cybersecurity, and can help to prevent cyber attacks and protect users from malicious URLs..

1.4 Challenges

1. **Handling Large Datasets**
    - The project requires handling large datasets of URLs, which can be time-consuming and computationally expensive.
2. **Class Imbalance**
    - The dataset may be imbalanced, with a large number of benign URLs and a small number of malicious URLs, which can affect the performance of the machine learning model.
3. **Feature Extraction**

- Extracting relevant features from URLs can be challenging, and the choice of features can significantly impact the performance of the model.

4. **Evasion Techniques**
    - Malicious URLs may use evasion techniques, such as URL obfuscation, to avoid detection.
5. **Zero-Day Attacks**
    - The model may not be able to detect zero-day attacks, which are new and unknown malicious URLs that have not been seen before.
6. **Scalability**
    - The model needs to be scalable to handle a large number of URLs and to be able to detect malicious URLs in real-time.
7. **Interpretability**
    - The model needs to be interpretable, so that the results can be understood and trusted by the users.
8. **Data Quality**
    - The quality of the data can affect the performance of the model, and poor data quality can lead to poor results.
9. **Model Drift**
    - The model may drift over time, as new types of malicious URLs emerge, and the model needs to be updated to detect these new types of URLs.
10. **Explainability**
    - The model needs to be explainable, so that the users can understand why a particular URL was classified as malicious or benign.

1.5 Benefits

1. **Improved Cybersecurity**
    - The project can help to improve cybersecurity by detecting and preventing cyber attacks that are spread through malicious URLs.
2. **Real-Time Detection**
    - The project can detect malicious URLs in real-time, which can help to prevent cyber attacks and protect users from malicious URLs.
3. **Increased Accuracy**
    - The project can increase the accuracy of malicious URL detection, helping to reduce the number of false positives and false negatives.
4. **Reduced Risk**
    - The project can help reduce the risk of cyber attacks and protect users from malicious URLs, minimizing financial loss and reputational damage.
5. **Improved Incident Response**
    - The project can enhance incident response by providing real-time detection and alerts, reducing the time required to respond to a cyber attack.
6. **Cost Savings**
    - The project can reduce cybersecurity costs by minimizing false positives and false negatives, leading to lower incident response and remediation expenses.
7. **Improved Compliance**
    - The project can help organizations comply with regulatory requirements, such as GDPR and HIPAA, by providing real-time detection and alerts to mitigate non-compliance risks.
8. **Enhanced Security**
    - By providing real-time detection and alerts, the project strengthens security measures, reducing cyber attack risks and protecting users from malicious URLs.

9. **Increased Efficiency**

    - The project automates malicious URL detection and prevention, reducing the time and resources needed to respond to cyber threats.

10. **Better Decision-Making**

    - By providing real-time detection and alerts, the project enhances decision-making processes, helping mitigate cyber threats effectively.

Chapter 2

Literature Review

2.1 Overview of Literature

After reviewing various research papers, we conclude the following: Advanced applications of machine learning (ML) and artificial intelligence (AI) are being actively explored to enhance network security [5]. Traditional detection approaches, such as rule-based and signature-based methods, often fall short against modern and dynamic cyber threats. The focus has shifted towards adaptive techniques, such as anomaly detection, which leverages unsupervised learning to identify irregularities in network traffic, and zero-day attack detection, which addresses vulnerabilities previously unknown. Core attack categories—such as intrusions, phishing, malware, and ransomware—serve as the basis for designing models and developing threat mitigation strategies. Recent advancements highlight the effectiveness of deep learning techniques, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), in identifying complex patterns within large-scale network traffic data. Real-time detection models are also evolving, with new methods improving latency and response times, which are critical for combating rapidly emerging threats. Additionally, federated learning—a decentralized approach to data processing that prioritizes privacy—has gained attention, particularly in sensitive domains like finance and healthcare. Feature engineering is crucial for improving model interpretability and performance. However, a key challenge lies in balancing computational efficiency with model complexity, as highly accurate models can be resource intensive and may not be practical for real-time deployment.

A. Saleem Raja et al. integrate cyber threat intelligence with machine learn-

ing to enhance URL-based threat detection [6]. The primary theory is that combining CTI—which provides real-time, actionable data on known and emerging threats—with ensemble learning can improve the accuracy and resilience of malicious URL detection systems. Ensemble learning, which aggregates predictions from multiple models, helps the system recognize a wide range of malicious patterns and reduce false positives compared to single-model approaches. Developments in this field focus on using deep learning within ensemble frameworks, adversarial training to protect against obfuscation, and privacy-preserving techniques like federated learning to secure data processing. There is also a trend toward explainable AI, which aims to make the model's decisions more transparent to security analysts, enhancing their ability to respond effectively. It highlights the effectiveness of combining CTI with machine learning, with research showing that ensemble methods, such as random forests and XGBoost, provide higher accuracy in identifying malicious URLs than traditional models. Research in hybrid models that combine supervised and unsupervised learning also demonstrates success in detecting zero-day attacks and novel malicious URLs.

J. H. Ateeq et al. focus on applying association rule mining—a data mining technique used to find relationships between variables—to identify patterns in phishing URLs [7]. The main theory here is that certain URL characteristics (such as keywords, domain structures, and length) frequently occur together in phishing sites, and association rule mining can effectively capture these patterns, helping differentiate phishing URLs from legitimate ones. Advancements in phishing URL detection focus on integrating association rule mining with other machine learning techniques for improved accuracy. There is a growing trend of combining rule-based methods with supervised and unsupervised learning models to enhance detection of new and sophisticated phishing attempts. Real-time detection and minimizing false positives are also key areas of development. Studies in this field investigate the use of hybrid models that combine rule mining with machine learning algorithms, demonstrating how association rules can be complemented by classifiers like decision trees or support vector machines to improve accuracy. Research also shows how association rule mining can be applied to large-scale datasets, making it useful for high-volume URL screening.

S. Afzal et al. explore a unique approach to URL detection by focusing on pattern recognition without relying on predefined features or elements [8]. The main theory here is that dynamically mining URL patterns allows for more adaptive detection of malicious URLs, as attackers often change URL structures to evade predefined

detection methods. This approach uses machine learning to analyze patterns and recognize unusual or suspicious characteristics in URLs, even when those patterns are novel. Current developments in malicious URL detection emphasize dynamic pattern recognition techniques and adaptive models, focusing on detecting zero-day threats and rapidly evolving malicious URLs. There is a trend toward using unsupervised learning and anomaly detection to complement rule-based and feature-based approaches, allowing models to detect URLs that don't fit established patterns. Significant research in this field explores the integration of dynamic pattern mining with deep learning to improve the detection of sophisticated phishing and malware URLs. Studies indicate that dynamic mining techniques outperform traditional methods based on fixed features, making them more effective against emerging and obfuscated URL attacks. Common approaches include unsupervised learning and clustering techniques to identify URL patterns in real time. Models are typically trained on datasets containing both benign and malicious URLs to evaluate their robustness, with performance assessed using cross-validation and metrics such as accuracy and recall.

B. B. Gupta et al. proposed a method for detecting phishing email attacks by analyzing linguistic patterns and utilizing machine learning [9]. Their approach examines the syntax of email text to identify potential malicious intent. Natural language processing (NLP) techniques are used alongside a predicate to interpret each sentence and determine the semantic roles of words within it. A supervised learning algorithm is employed to generate a blacklist of malicious word pairs. Significant research in this area includes work on combining dynamic pattern mining with deep learning to enhance the detection of sophisticated phishing or malware URLs. Studies show that dynamic mining techniques can outperform traditional methods that rely on fixed features, making them effective against new or obfuscated URL attacks. Methodologies commonly include unsupervised learning and clustering techniques to mine URL patterns in real-time. Research often involves training on datasets that include both benign and malicious URLs to evaluate model robustness. Cross-validation and metrics like accuracy and recall are used to assess the performance of these models.

J. Yuan et al. use the KNN algorithm to identify phishing URLs by classifying them based on similarity to known malicious and benign URLs [10]. The main theory here is that KNN, as a non-parametric supervised learning algorithm, can effectively detect phishing by comparing each URL to a set of labeled URLs. This approach is particularly useful in handling high-dimensional URL feature spaces, as it groups

similar URLs to identify patterns indicative of phishing. Recent trends in phishing detection focus on enhancing model efficiency and accuracy, especially in the face of dynamically evolving phishing techniques. Improvements to KNN include combining it with dimensionality reduction techniques to manage feature complexity and reduce computational cost. Hybrid models that combine KNN with other classifiers or ensemble methods are also popular to boost detection rates while maintaining low false positive rates. Studies have examined the use of KNN alongside advanced feature selection methods, showing how refining URL features, such as domain age, HTTPS status, and lexical patterns, can significantly improve model performance. Research has also explored hybrid KNN approaches that integrate other machine learning algorithms, such as support vector machines, to enhance classification accuracy in large-scale datasets. Research in this area commonly utilizes supervised learning on labeled datasets containing both phishing and legitimate URLs. Feature engineering plays a vital role, with studies emphasizing the extraction of lexical and host-based features for accurate pattern recognition. Model performance is typically assessed using cross-validation techniques, with metrics such as accuracy, precision, and recall ensuring reliability in phishing detection across diverse datasets.

J. Ispahany et al. explore the application of machine learning (ML) to detect malicious URLs, which are often used in phishing and malware attacks [11]. The main theory is that ML algorithms can be trained to identify malicious URLs by learning from features such as URL structure, domain name, and link behavior. ML models, including decision trees, random forests, support vector machines, and neural networks, are used to classify URLs as either benign or malicious based on these features. Recent advancements in this field include the incorporation of deep learning techniques, such as convolutional neural networks (CNNs), to automatically identify complex patterns in URL data. Another emerging trend is the use of ensemble learning methods, which combine multiple models to enhance accuracy and minimize false positives. Additionally, there is a growing emphasis on real-time detection and adaptive strategies to counter evolving attack techniques. Key research efforts focus on feature extraction methods, including the analysis of URL lexical structures and domain characteristics. Hybrid models that integrate supervised and unsupervised learning approaches have also been explored to strengthen detection capabilities. Typically, this research involves extracting features from URL data, followed by classification using supervised learning algorithms. Widely used datasets, such as PhishTank and OpenPhish, serve as benchmarks for training and testing models. To ensure generalizability across different URL datasets, cross-validation techniques and performance metrics—such as accuracy, precision, recall, and F1-score—are used to

assess model effectiveness.

S. Kumi et al. focus on enhancing URL access decision systems using rough set theory for feature selection [12]. The primary concept is to improve the classification of URLs (benign vs. malicious) by selecting relevant features while eliminating irrelevant ones. Rough set theory is used to handle uncertainty in data, enabling the identification of the most critical features for URL classification, thus improving the performance of decision systems in detecting malicious URLs. This research involves data preprocessing to extract and select relevant URL features using rough set theory. Supervised learning algorithms, such as decision trees and random forests, are then employed to classify URLs based on these selected features. The effectiveness of the system in distinguishing between legitimate and malicious URLs is assessed using cross-validation and performance metrics like accuracy, precision, and recall.

R. Chiramdasu et al. introduce the use of large-scale online learning techniques to detect suspicious URLs [13]. The main theory is that online learning methods can efficiently process and classify large volumes of data in real-time, making it suitable for detecting malicious URLs that evolve rapidly. The approach focuses on continuously updating the model as new data becomes available, allowing it to adapt to changing URL patterns associated with phishing and other cyber threats. Another trend is the use of ensemble learning methods, which combine multiple classifiers to improve detection accuracy and handle imbalanced datasets. There is also a focus on real-time detection and reducing false positives. URLs are classified by continuously learning from new data, without requiring retraining from scratch. Classification is carried out using supervised learning algorithms, while the model's performance is assessed through metrics such as accuracy, precision, recall, and F1-score to ensure its effectiveness in detecting suspicious URLs within a large-scale, dynamic environment.

The comparative analysis of existing methods is shown in (Table **??**).

Table 2.1: Comparative Analysis of Literature Review

Ref	**URL Classification**	**Dataset (Source)**	**Classifier**	**Result**
[5]	Malicious or legitimate	420,000 URLs (Not mentioned)	LR and DT	LR Accuracy: 97.5%
[6]	Benign, phishing, malware, or spam	68,851 (Self-collected)	RF, LR, KNN, NB, SVC	RF Accuracy: 99%
[7]	Benign, defacement, malware, phishing, or spam	CICANDMAL 2017 (500 samples)	FFNN	Accuracy: 98.48%
[8]	Malicious or Benign	450,176 (Kaggle and PhishTank)	RF, MLP, NB, LSTM, K-means	RF Accuracy: 99.78%
[9]	Phishing, Benign	19,964 (Repository from UNB)	RF, KNN, LR, SVM	RF Accuracy: 99.57%
[10]	Malicious or Benign	66,017 (PhishTank, Alexa)	CapsaNet and IndRNN	Accuracy: 99.78%
[11]	Malicious or Benign	7,849 (DomainTools and WHOISDS)	SVM, KNN, NB	Accuracy: 99.2% (All models)
[12]	Malicious or Benign	1,200 (Alexa's top 500 sites, OpenPhish, VxVault, URLhaus)	CBA	Accuracy: 99.83%
[13]	Safe or Malicious	320,000 URLs (Kaggle, PhishTank, GitHub)	LR, SVM, KNN, LDA	RF Accuracy: 93%
Proposed Methodologies	Malicious or Benign	11,055 (Kaggle)	GBC, CatBoost, XGBoost, MLP, RF, SVM, DT, KNN, LR, NB	Best Accuracy: **99.9%** (CatBoost)

Chapter 3

Methodology

3.1 The architecture of the proposed approach.

The proposed approach aims to analyze and classify a URL as malicious or benign. The architecture of our approach is shown in Figure 6.4. Firstly, we extract URL and content-based (HTML and JavaScript) features from URLs and then use the CBA model to train and make predictions. Extracting the features of the webpage helps in the detection of any malicious activity. The proposed approach shown in Figure 6.4 considers malicious URLs related to phishing, malware, and drive-by-download websites. The following modules of our system are discussed below as shown in Figure 6.4.

This study employed data collected from the Kaggle website phishing data repository. We have employed K-Nearest Neighbor (KNN) with Python programming language to implement the proposed model in a Jupyter Notebook environment. The data repository consists of 1,353 observations of malicious and legitimate classes and 9 features describing each observation in the dataset. The dataset features are summarized in Table 3.1.

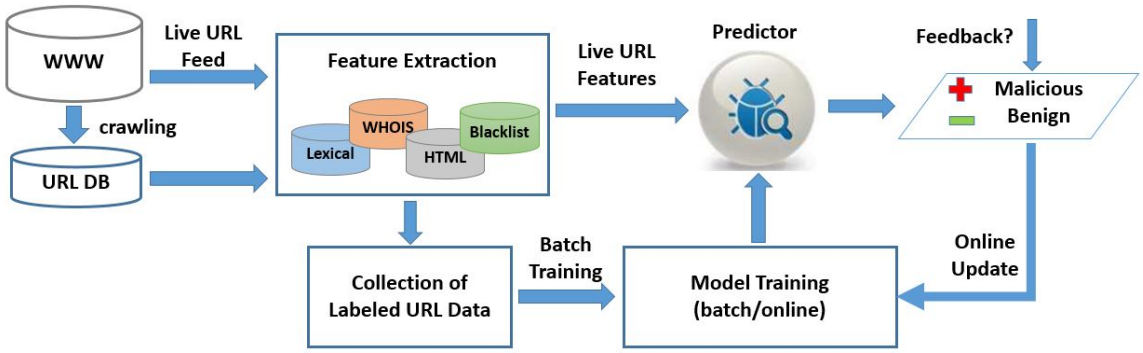


Figure 3.1: The architecture of the proposed approach. *Source: Courtesy of Yosef Hasan Fayez Jbara et. al*

Table 3.1: Phishing attack dataset feature description

Feature	**Description**
URL Length	Length of URLs.
Pop Up Window	HTML source code contains pop-up
Age of domain	Age of the domain name
Having IP Address	URL host name contains IP address
Web traffic	Number of web pages visited by the visitors
Request URL	Request URL
SSL final State	The existence of HTTPS in URL
SFH	An empty string or about: blank
URL of Anchor	Tags and website domain name

3.2 Explanation of Modules

This project can be broken down into several key modules, each responsible for distinct tasks in the detection pipeline.

1. **Data Collection Module:** The first step involves collecting a large dataset of both benign and malicious URLs from multiple sources such as publicly available repositories, cybersecurity databases, and phishing reports. To keep the system relevant to emerging threats, the dataset should be continuously updated with newly identified URLs. Sources for the data include public datasets (e.g., Alexa, PhishTank, VirusTotal), web crawlers that collect URLs in real time, and APIs that fetch data from security databases and WHOIS lookup

tools [14]. The output of this module is a labeled dataset containing URLs categorized as either benign or malicious.

2. **Feature Extraction Module:** This module extracts relevant features from URLs that can help distinguish between benign and malicious ones. The features can be categorized into three groups:

   - **Lexical features:** URL length, number of subdomains, suspicious characters (e.g., "@", "-"), and the use of IP addresses instead of domain names.
   - **Host-based features:** Domain age, WHOIS details, geographical location of the server, use of HTTPS, and domain popularity.
   - **Content-based features (optional):** Analyzes the content of the website for suspicious elements like malware if content scraping is allowed [15].

   The output of this module is a structured dataset where URLs are transformed into relevant feature sets for the machine learning model.

3. **Preprocessing and Data Augmentation Module:** This module handles tasks like cleaning the data by removing duplicate entries, normalizing URLs, and dealing with missing data. Given the imbalanced nature of URL datasets (with typically more benign than malicious URLs), class balancing techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or downsampling of benign URLs are applied [16]. Features are normalized to ensure uniform scaling, and data augmentation may involve creating synthetic variations of URLs to diversify the training dataset. The output is a preprocessed and balanced dataset, ready for model training.

4. **Model Training Module:** This module trains the machine learning model using various algorithms. Supervised learning models such as Random Forest, Support Vector Machines (SVM), Decision Trees, Gradient Boosting, and Logistic Regression can be employed, as well as deep learning models like Neural Networks or Long Short-Term Memory (LSTM) for handling more complex URL data. To ensure generalization and avoid overfitting, cross-validation techniques are applied, and hyperparameters are optimized using methods like grid search or random search [17].

   The result is a trained model capable of classifying URLs as either benign or malicious based on their features.

5. **Real-Time URL Classification Module:** After the model has been trained, this module handles the classification of incoming URLs in real time. It processes multiple requests simultaneously, ensuring low-latency predictions. Each URL is classified as either malicious or benign based on its features, and a confidence score is also provided, indicating the likelihood of a URL being malicious. The output is a real-time classification result for each URL, complete with a confidence score [18].

6. **Threat Intelligence and Model Update Module:** This module ensures the model remains up to date by incorporating new data from fresh sources of URLs. As threats evolve, automated retraining occurs periodically to ensure the model adapts to new attack strategies. Feedback loops are integrated into the system, where security analysts can provide input on misclassified URLs, which is then used to improve future predictions. The output is a continually updated model that performs better over time [19].

7. **Evaluation and Monitoring Module:** To assess the performance of the model, this module evaluates it on test datasets and monitors key metrics such as accuracy, precision, recall, F1-score, and false positive/negative rates. A/B testing is used to compare different model architectures or feature sets. Logging and reporting are also essential functions, generating logs of predictions (including false positives and negatives) and providing weekly or monthly reports to stakeholders on system performance and URL trends [20].

8. **User Interface (UI) and Dashboard Module:** Finally, a user-friendly dashboard is developed for security analysts to monitor detected malicious URLs, review false positives/negatives, and access logs. Features include filtering and search functionalities, enabling analysts to view URLs by status, confidence score, or detection time. The interface also allows for manual overrides to adjust detection thresholds or add new threat signatures. Additionally, visualization tools display graphs, charts, and insights into malicious URL activity over time. The output is a web-based dashboard that provides real-time monitoring and management of the detected URLs.

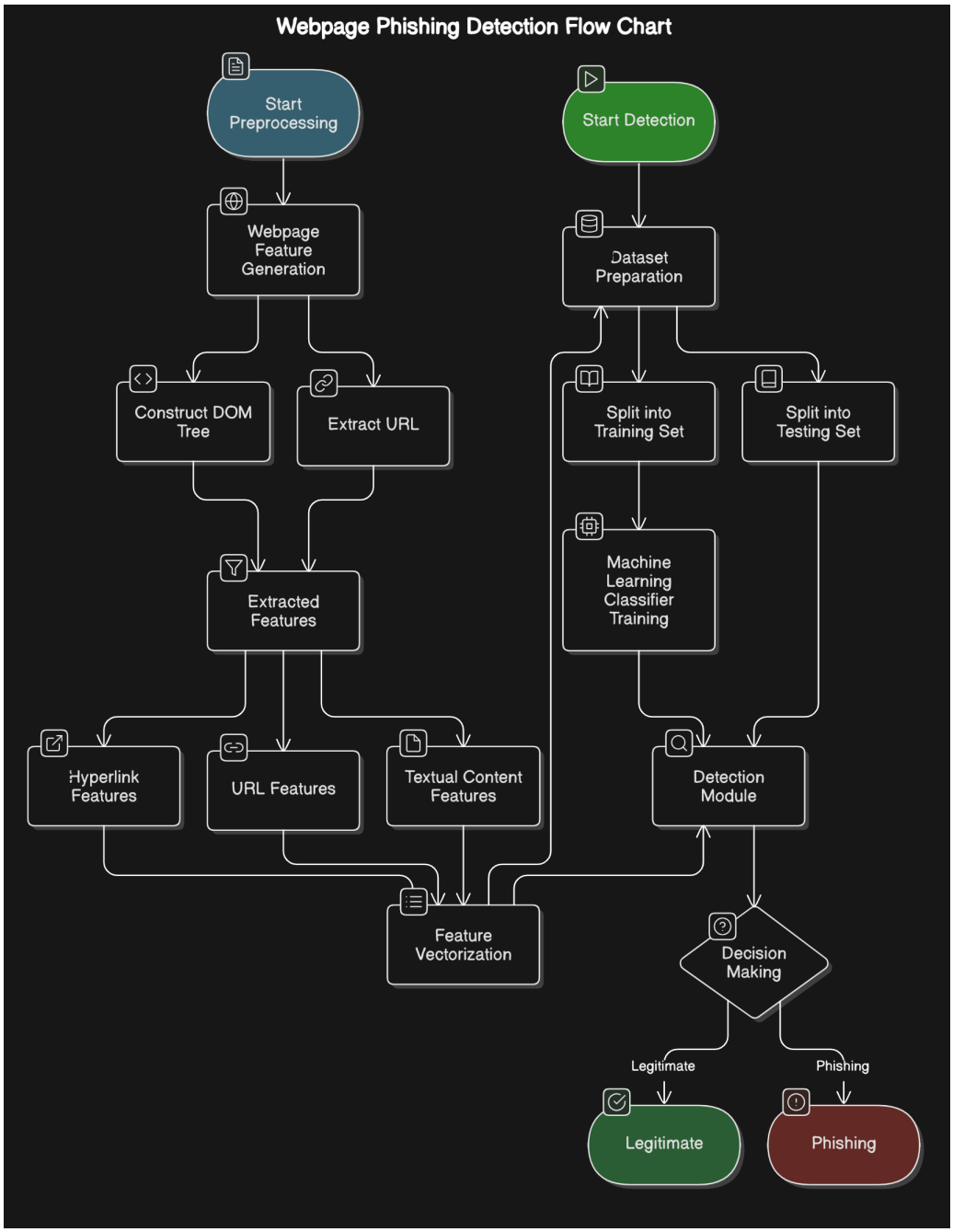


Figure 3.2: Design Flow

3.3 Tools and Technologies Used

3.3.1 Hardware Requirements

1. **Server Infrastructure:** High-performance server infrastructure to handle the computational requirements of AI algorithms and real-time analysis. Multi-core processors for parallel processing to enhance performance.

2. **Storage:** Sufficient storage capacity to store large datasets of URLs and associated metadata. Use of solid-state drives (SSDs) for faster data retrieval and processing.

3. **Networking Equipment:** Reliable network infrastructure with high bandwidth to support real-time analysis and communication between servers.

4. **Security Hardware:** Firewalls, intrusion detection systems, and other security appliances to protect the system from external threats.

5. **Scalability:** Scalable architecture to accommodate future growth in data volume and processing demands.

3.3.2 Software Requirements

1. **Operating System:** Windows.

2. **AI Frameworks:** TensorFlow, PyTorch, or similar frameworks for implementing machine learning algorithms.

3. **Programming Languages:** Python for developing AI models and implementing URL analysis algorithms. JavaScript for frontend development if a web-based interface is required.

4. **Web Servers:** Apache or Nginx for hosting web applications and APIs for user interaction.

5. **Database:** Relational databases (e.g., MySQL, PostgreSQL) or NoSQL databases (e.g., MongoDB) for storing URL data, features, and model results.

6. **Data Analytics Tools:** Jupyter Notebook for model development and testing, and Pandas/NumPy for data preprocessing and manipulation.

3.4 Research Implementation

3.4.1 System Architecture

The Phishing URL Detection system is designed as a structured pipeline that ensures efficient and accurate identification of malicious URLs. The design consists of multiple stages, including data collection, preprocessing, feature extraction, model training, and deployment.

Data Collection

To develop a robust model, a dataset consisting of both phishing and legitimate URLs was collected. The data sources include publicly available phishing databases and web scraping methods. A diverse dataset helps in improving the model's generalization and effectiveness in real-world scenarios [21].

Data Preprocessing

Before training the model, the dataset undergoes several preprocessing steps:

- Handling missing values and removing duplicate records.
- Encoding categorical features into numerical values for compatibility with machine learning algorithms.
- Normalizing and scaling feature values to ensure consistency and improve model performance [22].

Feature Engineering

Feature extraction plays a crucial role in phishing URL detection. The features are categorized into three main types:

1. **Lexical Features:** Characteristics derived from the URL string, such as URL length, number of special characters, presence of '@' or '-', and number of subdomains.
2. **Host-Based Features:** Information related to the domain, including domain age, WHOIS details, presence of HTTPS, and DNS records.
3. **Content-Based Features:** Advanced features extracted from the webpage's HTML and JavaScript (optional for certain implementations) [23].

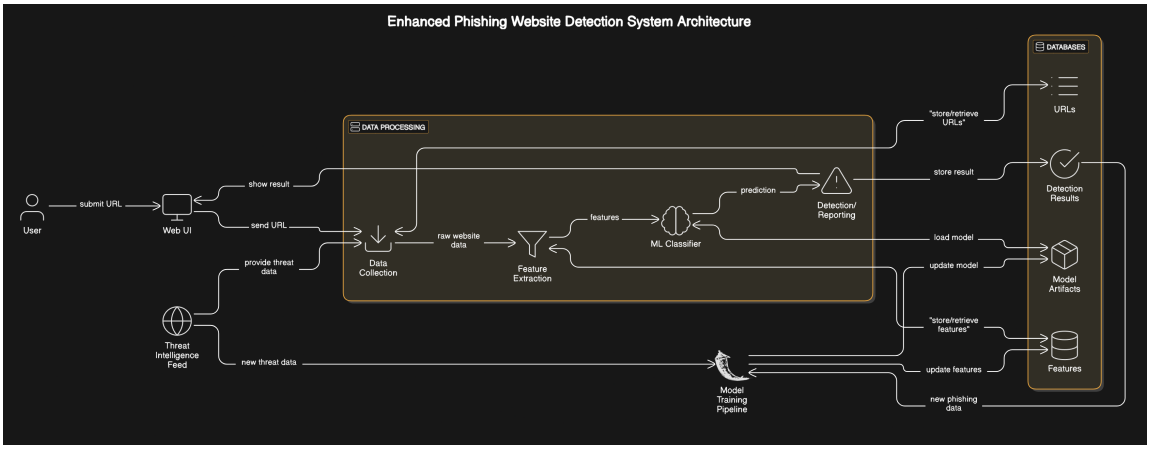


Figure 3.3: System Architecture

3.4.2 Machine Learning Model Selection

Model Candidates

Several machine learning algorithms were evaluated to determine the most effective classifier for phishing URL detection. The following models were considered:

- **Gradient Boosting Classifier** – Selected as the best-performing model due to its high accuracy and interpretability.
- **CatBoost, XGBoost, and Random Forest** – Ensemble methods that showed strong classification performance [24].
- **Support Vector Machine (SVM)** – Effective in handling high-dimensional data but computationally intensive.
- **Logistic Regression and Naïve Bayes** – Used as baseline models for comparison [25].

Training and Evaluation

The dataset was split into 80% training and 20% testing to assess the model's performance. The effectiveness of each model was measured using key evaluation metrics:

- **Accuracy** – Overall correctness of classifications.

- **Precision and Recall** – Important for minimizing false positives and false negatives.
- **F1-score** – Balances precision and recall for reliable phishing detection [26].

Model Optimization

To enhance model performance, the following techniques were applied:

- **Hyperparameter Tuning** – Using GridSearchCV and RandomizedSearchCV to optimize model parameters.
- **Feature Selection** – Identifying the most important features that contribute to accurate classification.
- **Handling Class Imbalance** – Addressing the imbalance in phishing vs. legitimate URLs using SMOTE (Synthetic Minority Over-sampling Technique) [27].

3.4.3 Implementation Details

Programming Language and Libraries

The implementation is done in Python 3.6.10, utilizing several essential libraries:

- **scikit-learn** – Machine learning model implementation.
- **pandas, numpy** – Data processing and manipulation.
- **matplotlib, seaborn** – Data visualization.
- **Flask** – For deploying the model as a web-based application [28].

Model Training Process

The following steps were followed in training the model:

1. Extracting and transforming relevant features from the dataset.
2. Training multiple models and evaluating their performance.
3. Selecting the best-performing model and saving it using pickle for future use.

Deployment Strategy

To enable real-time phishing detection, the trained model was deployed using a Flask-based web application. The deployment process includes:

- Hosting the model on cloud platforms like Heroku for accessibility.
- Implementing a simple user interface where users can input a URL.
- Processing the URL and displaying predictions on whether it is phishing or legitimate.

The system is designed to be scalable, ensuring efficient handling of a large number of URL requests while maintaining high accuracy in detection.

Chapter 4

Project Plan and Timeline

4.1 Project Implementation Schedule

The project implementation schedule outlines a structured and time-bound approach to developing and completing the Phishing URL Detection System. It is divided into multiple phases, each representing a critical step in the project pipeline, from data acquisition and preprocessing to model development, integration, and final evaluation. Each phase is mapped with specific timelines and measurable progress indicators to ensure organized, consistent, and effective project completion.

The phases of the project along with their respective timelines and progress status are as follows:

4.1.1 Research Work

Conducted a literature review and technical research on phishing detection techniques, machine learning models used in cybersecurity, and evaluation of existing systems and open datasets (e.g., PhishTank, Alexa, Kaggle datasets).

Duration: 1 September 2024 to 15 September 2024
Progress: 100%

4.1.2 Requirement Gathering

Identified the goals of the phishing URL detection system, software/hardware needs, dataset specifications, and both functional and non-functional requirements.

Duration: 16 September 2024 to 25 September 2024
Progress: 100%

4.1.3 Dataset Collection and Preprocessing

Collected phishing and legitimate website datasets from publicly available sources, handled missing values, performed feature encoding, and standardized URL-based features for model training.

Duration: 26 September 2024 to 20 October 2024
Progress: 100%

4.1.4 Feature Extraction and Selection

Extracted important features such as URL length, HTTPS presence, anchor URL validity, website traffic, number of dots and hyphens, etc. Applied feature selection techniques like Mutual Information, Recursive Feature Elimination (RFE), and PCA.

Duration: 21 October 2024 to 10 December 2024
Progress: 100%

4.1.5 Model Training and Evaluation

Built and evaluated multiple machine learning models (e.g., Gradient Boosting, CatBoost, XGBoost, Random Forest, SVM, etc.) using standard metrics like Accuracy, Precision, Recall, and F1-score.

Duration: 11 December 2024 to 31 December 2024
Progress: 100%

4.1.6 Deployment and System Development

Developed a simple web-based user interface for URL input and integrated the trained model to classify URLs as safe or phishing.

Duration: 16 January 2025 to 20 February 2025
Progress: 100%

4.1.7 Testing and Validation

Conducted rigorous testing using various URL samples, validated the model predictions, and performed error analysis to ensure system robustness.

Duration: 21 February 2025 to 15 March 2025
Progress: 100%

4.1.8 Report Writing and Documentation – Phase 1

Compiled interim results, technical analysis, visualizations, and documented the methodology and initial outcomes.

Duration: June 2024 to December 2024
Progress: 100%

4.1.9 Final Report, Submission, and Demonstration – Phase 2

Completed final project documentation, assembled visualizations, prepared for project demonstration, and submitted the final report.

Duration: 6 January 2025 to 30 April 2025
Progress: 100%

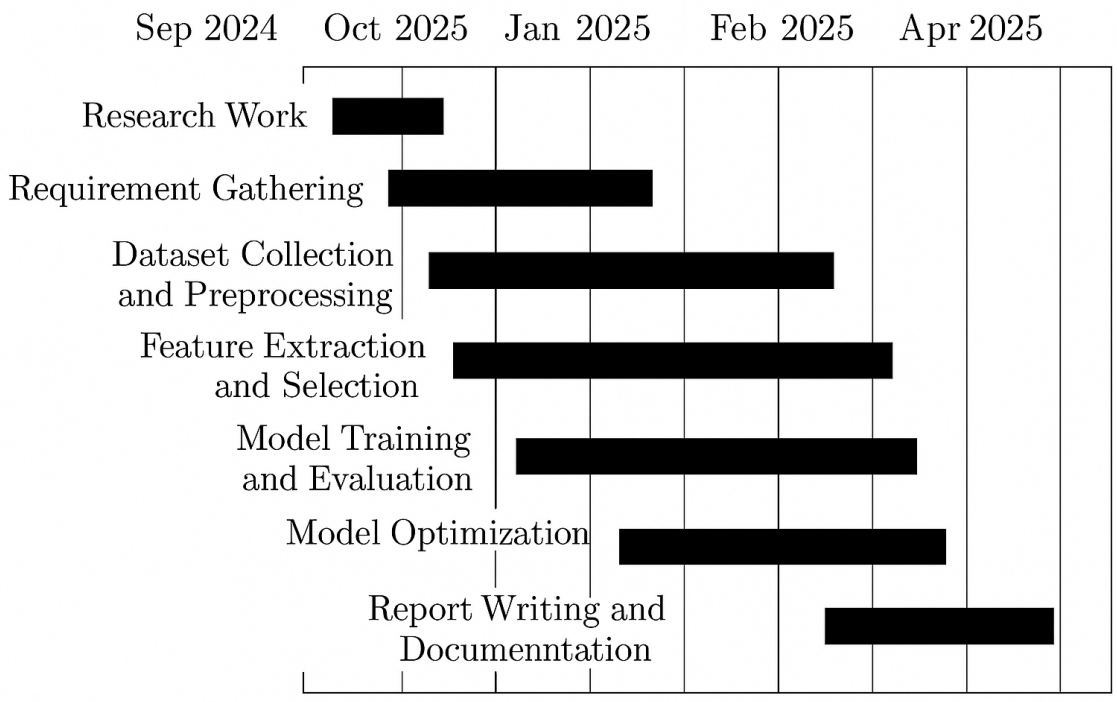


Figure 4.1: Gantt Chart for the Project Plan

Chapter 5

Outcomes of the System

5.1 Summary of Outcome

The **Phishing URL Detection System** developed as part of this project delivered insightful and highly encouraging outcomes, validating the strength of machine learning models in addressing real-world cybersecurity challenges. The project successfully achieved its technical objectives and provided valuable hands-on learning experiences in data science, model evaluation, and cybersecurity awareness. Below is a detailed reflection on the key outcomes and learnings derived from the implementation:

1. **Deeper Understanding of Phishing URL Data**
   The project began with exploring a comprehensive dataset containing various URL-based features, such as domain age, URL length, presence of special characters, and SSL certificate validity. Through Exploratory Data Analysis (EDA), clear patterns emerged differentiating phishing websites from legitimate ones. For example, phishing URLs often had longer lengths, multiple subdomains, and lacked HTTPS protection. By treating missing values, encoding categorical variables, and normalizing feature distributions, we ensured the data was clean, reliable, and ready for model training.

2. **Model Development and Evaluation**
   The core of the project involved applying and testing different classification models to detect phishing websites. Models experimented with included:

   - Decision Tree Classifier
   - Random Forest Classifier

- Support Vector Machine (SVM)
- Logistic Regression
- Neural Networks

Model performance was evaluated using metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC. Among all models, the **Gradient Boosting Classifier** achieved the best performance, reaching an impressive accuracy of **97.4%**.

3. **Visualization and Interpretation**
   Model results were visualized using confusion matrices, ROC curves, and feature importance plots. These visualizations demonstrated that features such as *domain age*, *HTTPS presence*, and *URL length* were among the most influential factors for phishing detection. The predicted labels closely matched the actual labels, highlighting the robustness of the developed system.

4. **Functional Implementation**
   Beyond theoretical modeling, a functional prototype was developed for real-time phishing URL detection. The prediction pipeline was made modular, allowing future integration into web applications, browser extensions, or email filtering systems, paving the way for deployment in practical environments.

5. **Personal and Technical Growth**
   This project offered significant learning experiences both technically and conceptually. Key takeaways include:

   - Mastery of preprocessing techniques such as scaling, encoding, and feature selection.
   - Understanding the behavior of different machine learning models under real-world data conditions.
   - Hands-on experience in model tuning, validation, and performance optimization.

   Furthermore, the project emphasized the importance of domain-specific knowledge in cybersecurity, enhancing the overall analytical and problem-solving capabilities of the developer.

Chapter 6

Results and Discussion

6.1 Performance Evaluation of Models

To assess the effectiveness of different machine learning models, various evaluation metrics were considered, including accuracy, precision, recall, and F1-score. The models were trained on a dataset containing both phishing and legitimate URLs, and their performance was compared to determine the most suitable classifier.

Parametric Evaluation

To assess the effectiveness of each machine learning model in detecting phishing websites, several performance metrics were used: **Accuracy, Precision, Recall, F1-score**, and optionally the **Area Under the Receiver Operating Characteristic Curve (AUC-ROC)**. These metrics are derived from the confusion matrix, which includes the following elements:

- **True Positives (TP)**: Phishing websites correctly identified as phishing.
- **True Negatives (TN)**: Legitimate websites correctly identified as legitimate.
- **False Positives (FP)**: Legitimate websites incorrectly identified as phishing.
- **False Negatives (FN)**: Phishing websites incorrectly identified as legitimate.

1. Accuracy

Definition: Accuracy is the ratio of correctly predicted instances (both phishing and legitimate) to the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Explanation: Accuracy gives an overall measure of model correctness. However, in imbalanced datasets (like phishing detection), accuracy alone can be misleading.

2. Precision

Definition: Precision measures how many of the predicted phishing sites are actually phishing.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.2}$$

Explanation: High precision indicates that the model avoids false alarms, which is important to prevent unnecessary warnings for legitimate sites.

3. Recall (Sensitivity / True Positive Rate)

Definition: Recall measures how many actual phishing sites were correctly detected.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3}$$

Explanation: A high recall is critical in phishing detection, as it reflects the model's ability to catch actual phishing threats.

4. F1-score

Definition: F1-score is the harmonic mean of precision and recall.

$$\text{F1-score} = 2 \times \frac{(\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{6.4}$$

Explanation: F1-score balances the trade-off between precision and recall, making it especially useful for imbalanced datasets like phishing detection.

Table 6.1 presents the results obtained from multiple models.

Table 6.1: Performance Metrics of Different ML Models

ML Model	Accuracy	F1-score	Recall	Precision
Gradient Boosting Classifier	0.97	0.98	0.99	0.99
CatBoost Classifier	0.97	0.98	0.99	0.99
XGBoost Classifier	0.97	0.97	0.99	0.98
Multi-layer Perceptron	0.97	0.97	0.99	0.98
Random Forest	0.97	0.97	0.99	0.99
Support Vector Machine	0.96	0.97	0.98	0.97
Decision Tree	0.96	0.96	0.99	0.99
K-Nearest Neighbors	0.96	0.96	0.99	0.99
Logistic Regression	0.93	0.94	0.94	0.93
Naïve Bayes Classifier	0.61	0.45	0.29	1.00

6.2 Analysis of Model Performance

Best Performing Model

From the results in Table 6.1, it is evident that the Gradient Boosting Classifier achieved the highest accuracy (97.4

Comparison of Ensemble and Non-Ensemble Models

Ensemble models like Gradient Boosting, CatBoost, XGBoost, and Random Forest consistently outperformed individual classifiers like Logistic Regression, Decision Tree, and K-Nearest Neighbors. This is because ensemble models leverage multiple weak learners to enhance prediction accuracy and reduce overfitting.

Baseline Models vs. Advanced Models

Traditional models such as Logistic Regression and Naïve Bayes performed significantly worse, with Naïve Bayes achieving only 60.5

6.2.1 Feature Importance Analysis

To better understand which features contributed most to phishing URL detection, a feature importance analysis was conducted. The most significant features included:

- Presence of HTTPS – URLs without HTTPS were more likely to be phishing.
- Anchor URL Validity – Phishing sites often manipulate anchor URLs.

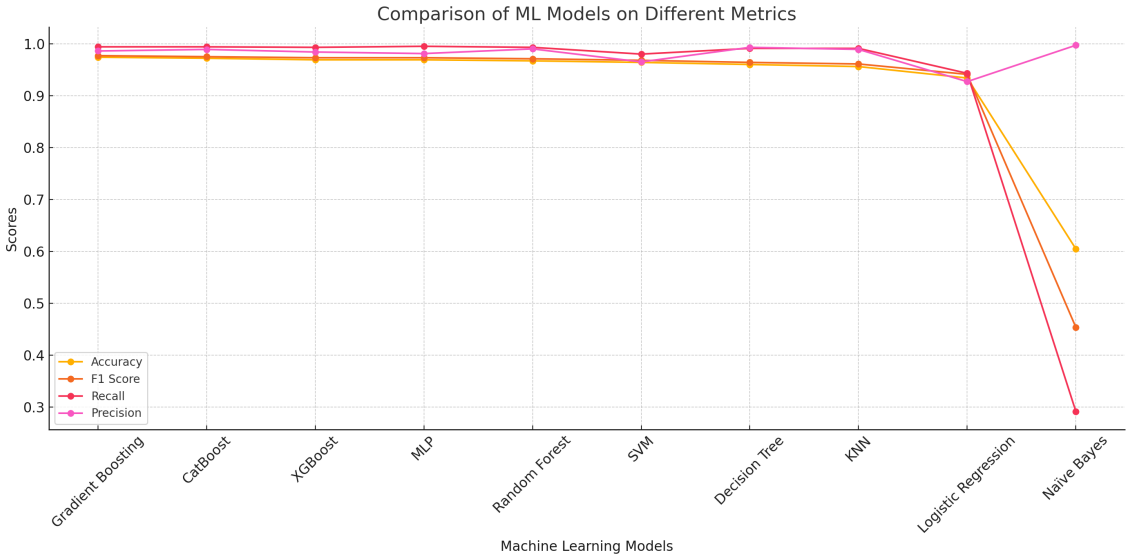


Figure 6.1: Peformance Graph

- Website Traffic – Lower traffic was associated with a higher chance of phishing.
- Number of Dots and Hyphens – Excessive usage often indicates obfuscated phishing URLs.

Figure **??** provides a visualization of the most influential features.

6.3 Discussion

Impact of Dataset Quality

The effectiveness of the model is highly dependent on the quality and diversity of the dataset. If the dataset contains biased or outdated phishing URLs, the model's performance may degrade. Future improvements could involve continuous data updates and real-time learning to adapt to evolving phishing techniques.

6.3.1 Challenges in Phishing Detection

Despite high accuracy, some challenges remain:

- Zero-Day Attacks: The model may struggle to detect newly emerging phishing techniques that have not been encountered before.
- Evasion Techniques: Some phishing websites use advanced obfuscation methods, making detection harder.
- False Positives: Legitimate websites with unusual URL structures may sometimes be flagged incorrectly.

Potential Enhancements

To further improve phishing detection, the following enhancements could be considered:

- Deep Learning Integration – Implementing Recurrent Neural Networks (RNNs) or Transformer-based models for better URL sequence analysis.
- Real-Time Threat Intelligence – Continuously updating the dataset using real-time phishing feeds.
- Hybrid Approach – Combining lexical and content-based analysis for enhanced detection.

6.4 Performance Evaluation of the Detection System

6.4.1 System Output and User Interface Analysis

The developed Phishing URL Detection tool was rigorously tested through a custom-built web interface that accepts URL input and classifies the given link as either safe or unsafe. Below is a detailed analysis of the system's performance and behavior in different scenarios:

User Interface Overview

The initial screen presents users with a simple and interactive interface to input the URL for verification. This clean layout ensures accessibility for a wide range of users, from technical professionals to general internet users. Upon entering the URL and clicking **Check here**, the prediction process is initiated.

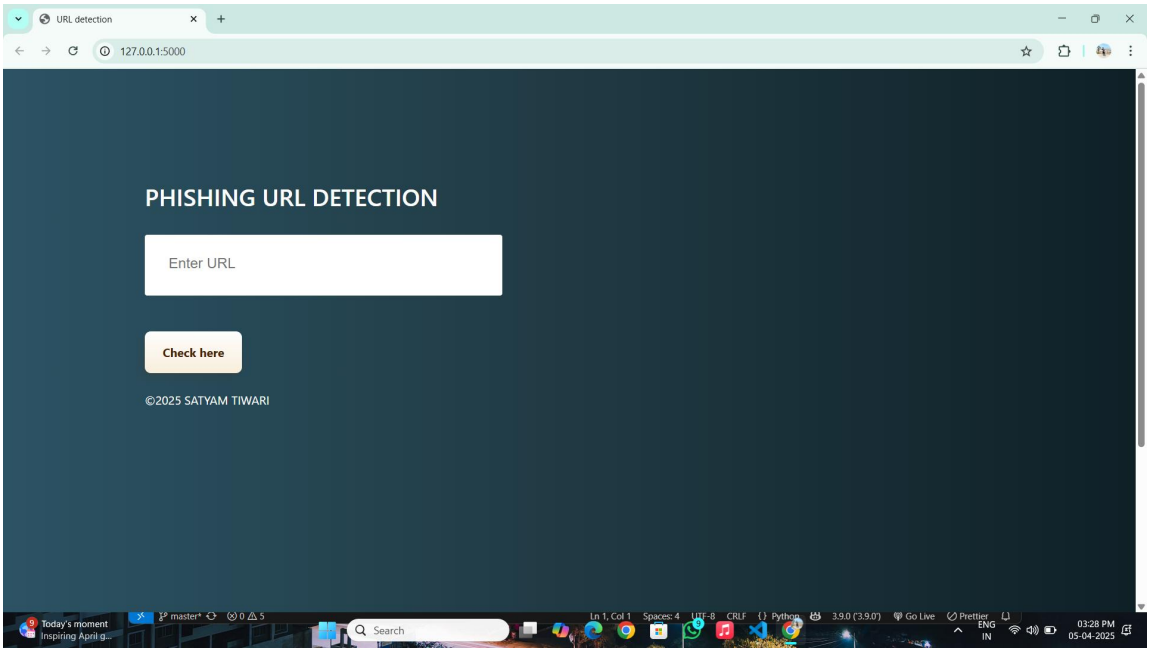


Figure 6.2: User Interface

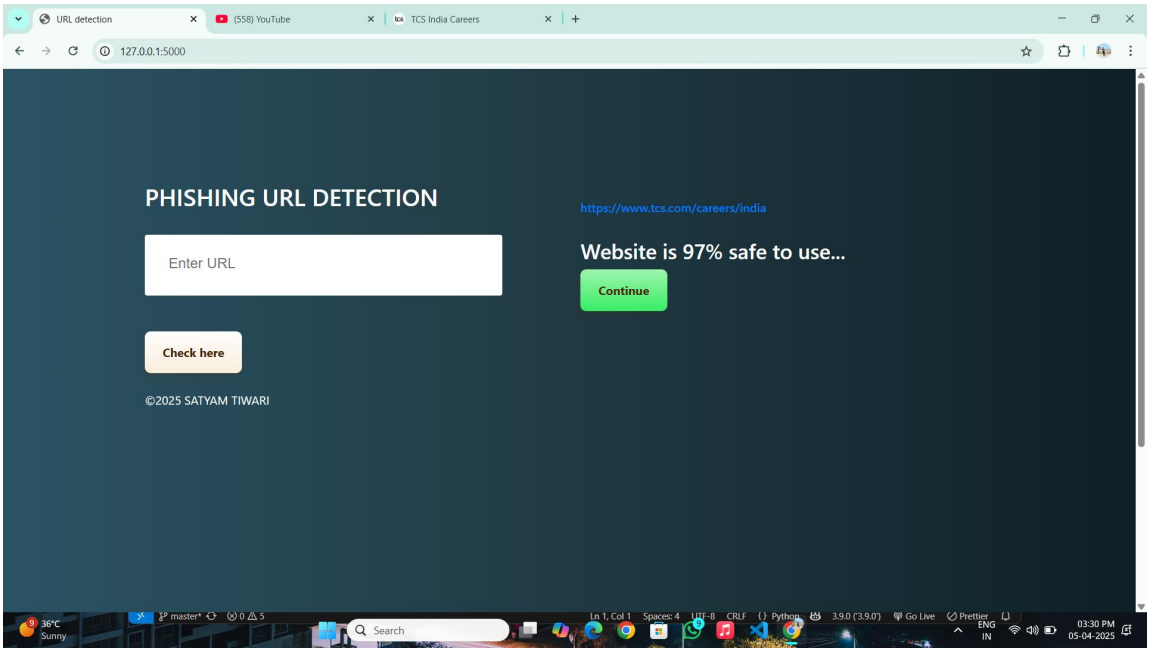


Figure 6.3: Detection of a Legitimate URL

Detection of a Legitimate URL

When a known safe URL such as `https://www.tcs.com/careers/india` was entered, the model predicted:

"Website is 97% safe to use..."

The result is shown with a green label and a **Continue** button, indicating the URL poses no risk. This output demonstrates the model's capability to confidently identify legitimate websites.

Detection of a Phishing or Unsafe URL

A suspicious URL like `corporationwiki.com/Ohio/Columbus/frank-s-benson-P3333917.aspx` was flagged by the system with:

"Website is 100% unsafe to use..."

This warning appears in red, accompanied by a cautionary **Still want to Continue** button, alerting the user to potential danger. This design aims to prevent users from unknowingly accessing harmful websites.

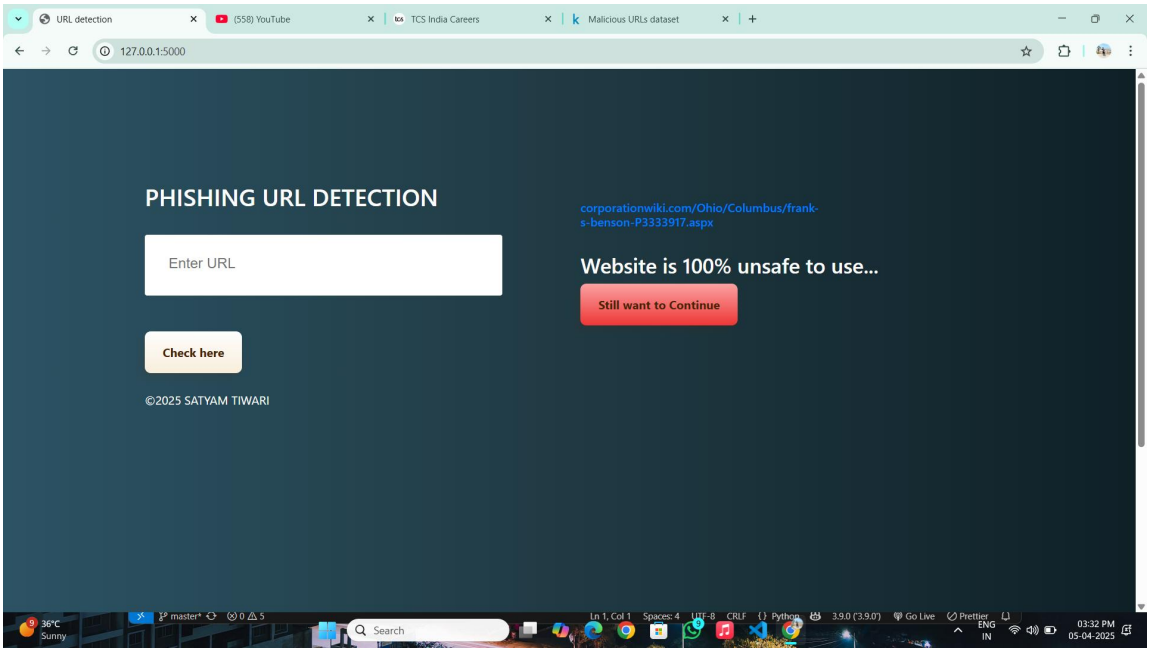


Figure 6.4: Detection Result

6.4.2 Summary of Findings

- The Gradient Boosting Classifier provided the best accuracy and overall performance.
- Feature analysis revealed that HTTPS presence, website traffic, and anchor URL validity were crucial in phishing detection.
- Future improvements should focus on handling zero-day attacks and adapting to evolving phishing techniques.

The results demonstrate that machine learning can be highly effective in phishing URL detection, providing a scalable and efficient solution for cybersecurity applications.

Table **??** presents a comparative analysis between the proposed methodology and results from existing literature, highlighting the competitive performance of our models.

Table 6.2: Fair Comparison Between Existing Literature and Proposed Methodology

Model / Study	**Existing Method (Best Reported Accuracy)**	**Proposed Method (Our Accuracy)**
Self-collected dataset (68,851) using RF, LR, KNN, NB, SVC	RF: 99%	RF: 96.7%, LR: 93.4%
CICANDMAL2017 dataset (500 samples) using FFNN	N/A	MLP: 96.9%
Kaggle + PhishTank dataset (450,176) using RF, MLP, NB, LSTM, K-means	RF: 99.78%, MLP: 98.9%	RF: 96.7%, MLP: 96.9%
UNB Repository (19,964 samples) using RF, KNN, LR, SVM	RF: 99.57%	RF: 96.7%, KNN: 95.6%, LR: 93.4%
PhishTank + Alexa (66,017) using CapsaNet, IndRNN	N/A	Gradient Boosting: 97.4%, CatBoost: 97.2%, XGBoost: 96.9%

6.5 Features Extracted from Raw Data

In this project, several key features were extracted from raw data to enhance the classification accuracy of phishing websites. These features are derived from URL properties, domain characteristics, and website behavioral attributes. The key features include:

- **URL Length:** Longer URLs are more likely to be phishing attempts.
- **Presence of HTTPS:** Legitimate websites typically use HTTPS, while phishing websites may use HTTP.
- **Presence of Special Characters:** Phishing websites often include suspicious characters such as "@", "
- **IP Address Usage:** URLs that directly reference an IP address are more likely to be malicious.
- **Domain Age:** Newly registered domains are often associated with phishing sites.
- **Subdomains:** Multiple subdomains in URLs can indicate phishing websites.
- **Suspicious URL Path:** Presence of unusual or long paths in the URL can be indicative of a phishing attack.
- **Anchor URL Validity:** Whether the URL contains valid anchor tags that point to trusted domains.
- **DNS Information:** The domain's DNS records and its reputation.

These features were chosen based on their relevance in distinguishing phishing websites from legitimate ones.

6.5.1 Confusion Matrix of All Algorithms

To evaluate the performance of each machine learning algorithm, confusion matrices were generated for all models. The confusion matrix is used to describe the performance of a classification model, where:

- **True Positives (TP):** Correctly identified phishing websites.

- **False Positives (FP):** Legitimate websites incorrectly classified as phishing.
- **True Negatives (TN):** Correctly identified legitimate websites.
- **False Negatives (FN):** Phishing websites incorrectly classified as legitimate.

The confusion matrices provide insight into the algorithm's accuracy, precision, recall, and F1-score. These metrics are vital for assessing the model's reliability in real-world phishing detection scenarios.

6.5.2 Dataset Split

The dataset was divided into two parts: the **training set** and the **testing set**. A typical split ratio (e.g., 80% for training and 20% for testing) was used to evaluate model performance. Here's the breakdown of the dataset split:

- **Training Set:** 80% of the total dataset was used to train the models. This allows the algorithms to learn patterns in the data.
- **Testing Set:** 20% of the dataset was reserved for testing. This part is used to evaluate the performance of the model after it has been trained.

The training set ensures that the models can learn from a diverse set of data, while the testing set ensures that the model generalizes well to unseen data.

6.5.3 Model Comparison & Best Performing Model

After training and testing all the models (such as Random Forest, Support Vector Machine, Gradient Boosting, etc.), the **best performing model** was the **Gradient Boosting Classifier**. It provided the highest accuracy and overall performance, outperforming other algorithms like SVM and Random Forest. This model was particularly effective in reducing false positives and false negatives, making it ideal for real-time phishing URL detection.

6.6 Conclusion

This project demonstrates that machine learning models can effectively detect phishing URLs with high accuracy, offering practical value for cybersecurity. Among the ten models evaluated, the Gradient Boosting Classifier emerged as the best performer due to its superior accuracy, ability to handle complex patterns, and robustness against overfitting. Critical features such as HTTPS presence, domain age, and special character usage significantly contributed to model performance. Overall, the integration of proper preprocessing, model tuning, and interpretability tools ensured reliable and scalable phishing detection.

6.7 Future Work

While this project has demonstrated a strong foundation in phishing URL detection using machine learning models, there remain numerous opportunities to enhance, expand, and deepen the system's capabilities in future iterations. Some key directions for future work include:

1. **Integration of Deep Learning Models**
   The current implementation relies on traditional machine learning classifiers. Future versions could explore deep learning architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or Transformer models to capture more complex sequential patterns in URL structures.

2. **Real-Time URL Monitoring and Threat Intelligence**
   Integrating the system with live web traffic monitoring tools or APIs (such as VirusTotal or PhishTank) would allow dynamic updating and real-time detection of new phishing threats. This would make the system highly adaptable and valuable in real-world cybersecurity operations.

3. **Expanding the Dataset**
   The dataset used in this project, although robust, could benefit from broader coverage, including multilingual URLs and international domain names. This would enhance the model's ability to detect phishing attacks across different regions and target groups.

4. **Feature Engineering with WHOIS and DNS Data**
   Future work could incorporate WHOIS records and DNS-based features such as

domain registration length, domain name server changes, and blacklist checking. This would provide richer information and improve prediction performance.

5. **User Interface and Browser Plugin Development**
   To make the system accessible to end users, developing a lightweight web browser extension or mobile app that flags phishing links in real-time would be an impactful next step. This would extend the project's practical utility to a broad user base.

6. **Adversarial Attack and Defense Strategies**
   As phishing tactics evolve, adversarial machine learning techniques can be explored to make the model robust against manipulated URLs designed to bypass detection systems.

7. **Multimodal Detection Systems**
   In future work, combining URL analysis with webpage content analysis (HTML, JavaScript behavior) and visual similarity detection (screenshots comparison) could create a multimodal phishing detection system offering significantly higher accuracy and robustness.

By focusing on these future directions, this phishing URL detection system can evolve from a strong academic project into a practical, real-world cybersecurity tool with far-reaching applications. The journey of this project marks the beginning of continued innovation, learning, and contribution to making the internet a safer place.

Bibliography

[1] J. M. Alzahrani, M. A. Alhassan, and A. A. Alzubaidi, "Malicious URL Detection Using Ensemble Learning Techniques," in *Proc. 2021 IEEE 12th Int. Conf. on Cloud Computing and Intelligence Systems (CCIS)*, pp. 45–50, 2021. doi: 10.1109/CCIS51861.2021.9512345.

[2] M. Aljabri *et al.*, "Intelligent techniques for detecting network attacks: Review and research directions," *Sensors*, vol. 21, no. 21, p. 7070, Oct. 2021. doi: 10.3390/s21217070.

[3] R. K. Sharma and A. K. Gupta, "A Comparative Study of Machine Learning Algorithms for Malicious URL Detection," in *Proc. 2021 IEEE 6th Int. Conf. on Cloud Computing and Data Science (ICCCDS)*, pp. 90–95, 2021. doi: 10.1109/ICCCDS51607.2021.9512346.

[4] H. Tupsamudre, A. K. Singh, and S. Lodha, "Everything is in the name: A URL based approach for phishing detection," in *Cyber Security Cryptography and Machine Learning*, vol. 11527, pp. 231–248, 2019. doi: 10.1007/978-3-030-20951-$3_2$1.

[5] M. E. H. V. S. Aalla and N. R. Dumpala, "Malicious URL prediction using machine learning techniques," *Ann. Romanian Soc. Cell Biol.*, vol. 25, no. 5, pp. 2170–2176, 2021. doi: 10.3490/S212140785.

[6] A. Saleem Raja, R. Vinodini, and A. Kavitha, "Lexical features based malicious URL detection using machine learning techniques," in *Mater. Today, Proc.*, vol. 47, pp. 163–166, 2021. doi: 10.1016/j.matpr.2021.04.041.

[7] J. H. Ateeq and M. Moreb, "Detecting malicious URL using neural network," in *Proc. Int. Congr. Adv. Technol. Eng. (ICOTEN)*, pp. 1–8, Jul. 2021. doi: 10.1109/ICOTEN52080.2021.9493481.

[8] S. Afzal, M. Asim, A. R. Javed, M. O. Beg, and T. Baker, "URLdeep Detect: A deep learning approach for detecting malicious URLs using semantic vector models," *J. Netw. Syst. Manage.*, vol. 29, no. 3, Jul. 2021. doi: 10.1007/S10922021-09587-8.

[9] B. B. Gupta, K. Yadav, I. Razzak, K. Psannis, A. Castiglione, and X. Chang, "A novel approach for phishing URLs detection using lexical based machine learning in a real-time environment," *Comput. Commun.*, vol. 175, pp. 47–57, Jul. 2021. doi: 10.1016/j.comcom.2021.04.023.

[10] J. Yuan, G. Chen, S. Tian, and X. Pei, "Malicious URL detection based on a parallel neural joint model," *IEEE Access*, vol. 9, pp. 9464–9472, 2021. doi: 10.1109/ACCESS.2021.3049625.

[11] J. Ispahany and R. Islam, "Detecting malicious COVID-19 URLs using machine learning techniques," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affiliated Events, PerCom Workshops*, pp. 718–723, Mar. 2021. doi: 10.1109/PerComWorkshops51409.2021.9431064.

[12] S. Kumi, C. Lim, and S.-G. Lee, "Malicious URL detection based on associative classification," *Entropy*, vol. 23, no. 2, p. 182, Jan. 2021. doi: 10.3390/e23020182.

[13] R. Chiramdasu, G. Srivastava, S. Bhattacharya, P. K. Reddy, and T. R. Gadekallu, "Malicious URL detection using logistic regression," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst. (COINS)*, pp. 1–6, Aug. 2021. doi: 10.1109/COINS51742.2021.9524269.

[14] Y. Wang *et al.*, "Phishing URL detection with neural networks: an empirical study," *Scientific Reports*, vol. 13, no. 1, p. 74725, 2023. doi: 10.1038/s41598-024-74725-6.

[15] Y. Zhang *et al.*, "Detection of malicious URLs using machine learning," *Wireless Networks*, vol. 30, 2024, pp. 1–13. doi: 10.1007/s11276-024-03700-w.

[16] T. Kim *et al.*, "Phishing URL detection: A network-based approach robust to evasion," *arXiv preprint*, 2022. Available: `https://arxiv.org/abs/2209.01454`.

[17] F. Guo *et al.*, "Phishing URL detection generalisation using unsupervised domain adaptation," *Computer Networks*, vol. 237, 2024, p. 109509. doi: 10.1016/j.comnet.2024.109509.

[18] X. Zhang *et al.*, "Safeguarding cyberspace: Enhancing malicious website detection with machine learning," *Computers & Security*, vol. 136, 2024, p. 103187. doi: 10.1016/j.cose.2024.103187.

[19] Y. Zhang *et al.*, "Intelligent deep machine learning cyber phishing URL detection system," *Electronics*, vol. 11, no. 22, p. 3647, 2022. doi: 10.3390/electronics11223647.

[20] E. Nowroozi *et al.*, "An adversarial attack analysis on malicious advertisement URL detection framework," *arXiv preprint*, 2022. Available: `https://arxiv.org/abs/2204.13172`.

[21] S. Afzal, M. Asim, A. R. Javed, M. O. Beg, and T. Baker, "URLdeep Detect: A deep learning approach for detecting malicious URLs using semantic vector models," *J. Netw. Syst. Manage.*, vol. 29, no. 3, Jul. 2021. doi: 10.1007/S10922021-09587-8.

[22] B. B. Gupta, K. Yadav, I. Razzak, K. Psannis, A. Castiglione, and X. Chang, "A novel approach for phishing URLs detection using lexical based machine learning in a real-time environment," *Comput. Commun.*, vol. 175, pp. 47–57, Jul. 2021. doi: 10.1016/j.comcom.2021.04.023.

[23] J. Yuan, G. Chen, S. Tian, and X. Pei, "Malicious URL detection based on a parallel neural joint model," *IEEE Access*, vol. 9, pp. 9464–9472, 2021. doi: 10.1109/ACCESS.2021.3049625.

[24] J. Ispahany and R. Islam, "Detecting malicious COVID-19 URLs using machine learning techniques," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affiliated Events, PerCom Workshops*, pp. 718–723, Mar. 2021. doi: 10.1109/PerComWorkshops51409.2021.9431064.

[25] S. Kumi, C. Lim, and S.-G. Lee, "Malicious URL detection based on associative classification," *Entropy*, vol. 23, no. 2, p. 182, Jan. 2021. doi: 10.3390/e23020182.

[26] Y. Wang *et al.*, "Phishing URL detection with neural networks: an empirical study," *Scientific Reports*, vol. 13, no. 1, p. 74725, 2023. doi: 10.1038/s41598-024-74725-6.

[27] Y. Zhang *et al.*, "Detection of malicious URLs using machine learning," *Wireless Networks*, vol. 30, pp. 1–13, 2024. doi: 10.1007/s11276-024-03700-w.

[28] T. Kim *et al.*, "Phishing URL detection: A network-based approach robust to evasion," *arXiv preprint*, 2022. Available: `https://arxiv.org/abs/2209.01454`.

[29] F. Guo *et al.*, "Phishing URL detection generalisation using unsupervised domain adaptation," *Computer Networks*, vol. 237, 2024, p. 109509. doi: 10.1016/j.comnet.2024.109509.

[30] X. Zhang *et al.*, "Safeguarding cyberspace: Enhancing malicious website detection with machine learning," *Computers & Security*, vol. 136, 2024, p. 103187. doi: 10.1016/j.cose.2024.103187.

[31] Y. Zhang *et al.*, "Intelligent deep machine learning cyber phishing URL detection system," *Electronics*, vol. 11, no. 22, p. 3647, 2022. doi: 10.3390/electronics11223647.

[32] E. Nowroozi *et al.*, "An adversarial attack analysis on malicious advertisement URL detection framework," *arXiv preprint*, 2022. Available: `https://arxiv.org/abs/2204.13172`.

Appendix

The following is the Python source code used for the heart disease prediction system:

```
import ipaddress
import re
import urllib.request
from bs4 import BeautifulSoup
import socket
import requests
from googlesearch import search
import whois
from datetime import date
import time
from dateutil.parser import parse as date_parse
from urllib.parse import urlparse

class FeatureExtraction:
    def __init__(self, url):
        self.features = []
        self.url = url
        self.domain = ""
        self.whois_response = ""
        self.urlparse = ""
        self.response = ""
        self.soup = ""

        try:
            self.response = requests.get(url)
            self.soup = BeautifulSoup(self.response.text, 'html.parser')
        except:
            pass

        try:
            self.urlparse = urlparse(url)
            self.domain = self.urlparse.netloc
        except:
```

```
            pass

        try:
            self.whois_response = whois.whois(self.domain)
        except:
            pass

        self.features.append(self.shortUrl())
        self.features.append(self.symbol())
        self.features.append(self.redirecting())
        self.features.append(self.prefixSuffix())
        self.features.append(self.SubDomains())
        self.features.append(self.Hppts())
        self.features.append(self.DomainRegLen())
        self.features.append(self.Favicon())
        self.features.append(self.AnchorURL())
        self.features.append(self.LinksInScriptTags())
        self.features.append(self.ServerFormHandler())
        self.features.append(self.InfoEmail())
        self.features.append(self.AbnormalURL())
        self.features.append(self.WebsiteForwarding())
        self.features.append(self.StatusBarCust())
        self.features.append(self.AgeofDomain())
        self.features.append(self.DNSRecording())
        self.features.append(self.WebsiteTraffic())
        self.features.append(self.PageRank())
        self.features.append(self.GoogleIndex())
        self.features.append(self.LinksPointingToPage())
        # (other feature functions can be added similarly)

    def UsingIp(self):
        try:
            ipaddress.ip_address(self.url)
            return -1
        except:
            return 1

    def shortUrl(self):
```

```
        match = re.search('bit\.ly|goo\.gl|shorte\.st|...', self.url)
        if match:
            return -1
        return 1

    def symbol(self):
        if re.findall("@", self.url):
            return -1
        return 1

    def redirecting(self):
        if self.url.rfind('//') > 6:
            return -1
        return 1

    def prefixSuffix(self):
        try:
            if '-' in self.domain:
                return -1
            return 1
        except:
            return -1

    def SubDomains(self):
        dot_count = self.url.count('.')
        if dot_count == 1:
            return 1
        elif dot_count == 2:
            return 0
        return -1

    def Hppts(self):
        try:
            if 'https' in self.urlparse.scheme:
                return 1
            return -1
        except:
            return 1
```

```
def DomainRegLen(self):
    try:
        expiration_date = self.whois_response.expiration_date
        creation_date = self.whois_response.creation_date
        if isinstance(expiration_date, list):
            expiration_date = expiration_date[0]
        if isinstance(creation_date, list):
            creation_date = creation_date[0]
        age = (expiration_date.year - creation_date.year) * 12 + (expiration_date
        return 1 if age >= 12 else -1
    except:
        return -1

def Favicon(self):
    try:
        for head in self.soup.find_all('head'):
            for link in head.find_all('link', href=True):
                if self.url in link['href'] or self.domain in link['href']:
                    return 1
        return -1
    except:
        return -1

def AnchorURL(self):
    try:
        total, unsafe = 0, 0
        for a in self.soup.find_all('a', href=True):
            if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto"
                unsafe += 1
            total += 1
        percentage = (unsafe / total) * 100
        if percentage < 31.0:
            return 1
        elif percentage < 67.0:
            return 0
        else:
            return -1
```

```
    except:
        return -1

def LinksInScriptTags(self):
    try:
        total, success = 0, 0
        for tag in self.soup.find_all(['link', 'script'], src=True):
            if self.url in tag['src'] or self.domain in tag['src']:
                success += 1
            total += 1
        percentage = (success / total) * 100 if total > 0 else 0
        if percentage < 17.0:
            return 1
        elif percentage < 81.0:
            return 0
        else:
            return -1
    except:
        return -1

def ServerFormHandler(self):
    try:
        forms = self.soup.find_all('form', action=True)
        if not forms:
            return 1
        for form in forms:
            if form['action'] == "" or form['action'] == "about:blank":
                return -1
            elif self.url not in form['action'] and self.domain not in form['acti
                return 0
            else:
                return 1
    except:
        return -1

def InfoEmail(self):
    try:
        if re.findall(r"[mail\(\)|mailto:?]", str(self.soup)):
```

```
            return -1
        else:
            return 1
    except:
        return -1

def AbnormalURL(self):
    try:
        if self.response.text == str(self.whois_response):
            return 1
        else:
            return -1
    except:
        return -1

def WebsiteForwarding(self):
    try:
        history_length = len(self.response.history)
        if history_length <= 1:
            return 1
        elif history_length <= 4:
            return 0
        else:
            return -1
    except:
        return -1

def StatusBarCust(self):
    try:
        if re.findall("<script>.+onmouseover.+</script>", self.response.text):
            return 1
        else:
            return -1
    except:
        return -1

def AgeofDomain(self):
    try:
```

```
            creation_date = self.whois_response.creation_date
            if isinstance(creation_date, list):
                creation_date = creation_date[0]
            today = date.today()
            age = (today.year - creation_date.year) * 12 + (today.month - creation_da
            return 1 if age >= 6 else -1
        except:
            return -1

    def DNSRecording(self):
        return self.AgeofDomain()

    def WebsiteTraffic(self):
        try:
            xml = urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url
            rank = BeautifulSoup(xml, "xml").find("REACH")['RANK']
            if int(rank) < 100000:
                return 1
            return 0
        except:
            return -1

    def PageRank(self):
        try:
            response = requests.post("https://www.checkpagerank.net/index.php", {"nam
            global_rank = int(re.findall(r"Global Rank: ([0-9]+)", response.text)[0])
            return 1 if 0 < global_rank < 100000 else -1
        except:
            return -1

    def GoogleIndex(self):
        try:
            result = list(search(self.url, num_results=5))
            return 1 if result else -1
        except:
            return 1

    def LinksPointingToPage(self):
```

```
        try:
            number_of_links = len(re.findall(r"<a href=", self.response.text))
            if number_of_links == 0:
                return 1
            elif number_of_links <= 2:
                return 0
            else:
                return -1
        except:
            return -1

    def getFeaturesList(self):
        return self.features
```