A PROJECT REPORT ON

LEVERAGING MACHINE LEARNING APPROACHES FOR ENHANCED PHISHING URL DETECTION AND ANALYSIS

PROJECT REPORT SUBMITTED TO ICFAI TECH

AS

A PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BTECH IN DATA SCIENCE AND ARTIFICIAL INTELLIGENCE UNDER THE SUPERVISION OF

DR. PRIYANKA PARIMI

BY

CH. AKSHAY REDDY

22STUCHH010751

IcfaiTech

Faculty of Science & Technology (FST)

Department of Data Science and Artificial Intelligence

Faculty of Science and Technology

Icfai Tech Hyderabad.

CERTIFICATE

This is to certify that the project report entitled Leveraging Machine Learning Approaches For Enhanced Phishing URL Detection and Analysis submitted in fulfillment of the degree of B. Tech in (Data Science & Artificial Intelligence) is a record of original work carried out by me under the supervision of Dr. Priyanka Parimi, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made whenever the findings of others have been cited.

Dr. Priyanka Parimi	Dr. Pavan Kumar
Supervisor	Head Of Department
Dept. of DSAI	Dept. of DSAI

DECLARATION

I hereby declare that the work presented in the Project Report entitled is original and it has been under the guidance of Dr. Priyanka Parimi. The work has not been submitted to any other University for the award of any degree or diploma.

CH. AKSHAY REDDY

Date: 24-04-2025 Signature of the Student

ACKNOWLEDGEMENT

It gives me immense pleasure to acknowledge with gratitude the help and guidance rendered to me by a host of people to whom I owe a substantial completion of the seminar work**.**

I would like to express gratitude to **Dr. Priyanka Parimi**, Department of DS & AI, IcfaiTech for all the timely support and valuable suggestions during the period of my seminar**.** I am extremely thankful to mam for their valuable suggestions and constant support throughout the seminar**.** Finally, thanks to the ones who helped me directly or indirectly, parents and friends for their cooperation in completing the seminar**.**

ABSTRACT

Phishing is one of the fraudulent methods adopted in cyberspace, which leads people to reveal their important information while posing as trusted genuine sites. Cybercriminals create bogus websites and emails that resemble the real ones and prompt people to submit their details, whether personal credentials or bank information, or any other confidential data. Traditional phishing detection systems incorporate mostly machine learning models such as Random Forests, Decision Tree, etc. Unfortunately, the detection job is not successful with these modern techniques; besides, there are limitations like lower accuracy rates, high processing times, and no interface strictly designed for the end user. An even greater disadvantage with these available approaches is that they do not provide a comprehensive comparative study regarding various algorithms to identify that 'cutting edge' approach toward effective detection.

Addressing all these limitations, we propose a new advanced phishing detection system using machine learning in which relevant features of the URL are analyzed for accurate classification of fraudulent websites. This system uses a Gradient gradient-boosting classifier, the proposed system improves detection by extracting and investigating crucial segregations of legitimate and phishing URLs. Experiments show that the system can successfully track real-time websites, real and fake websites, and give better results than conventional systems regarding accuracy. Enhanced Phishing detection is said to provide short latency and better-optimized solutions for tracking cybersecurity threats.

Keywords: Machine Learning, Gradient Boosting Classifier, Cyber Threats, Decision Tree, Random Forest.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

1. INTRODUCTION

Phishing was, is, and will always be among the most common and financially damaging forms of cyberattack; it costs individuals and organizations billions every year. The basis on which these cyber attacks are founded is purely the harnessing of social engineering tactics to trick users into providing confidential information that falls under passwords, bank details, or even personal identifiers through the use of fake websites and emails designed to replicate, closely enough, legitimate platforms. Cybercriminals often duplicate branding elements like logos, layout designs, and domain names to make malicious pages look authentic to the degree that they make it hard for a user to tell from a genuine website whether an address is fraudulent or not [1][2].

With increasingly more complex and diversified phishing attacks being created, the tools and techniques natively implemented to detect them appear to be useless. In the past, phishing detection systems were mostly based on URL pattern analysis, server log analysis, or machine learning classifiers trained on static features [3][9]. Even with these, though, there are limitations. For instance, rule-based systems have the ability to flag legitimate sites as danger, thus generating a very high number of false positives [4][13]. Most of the existing methods are not dynamic enough to be able to collaborate with new and dynamic phishing tactics, especially the ones being used by sophisticated evasion or obfuscation techniques as well [5][8]. Some of the detection systems rely on outside databases that are not always updated in real time so they do not generate threats until too late [6]. Complex algorithms burden processors, thus real-time detection is impractical [10]. Finally, most of the existing tools are not user-friendly, thus further limiting their use for those not very familiar with technical problems [7][12].

In order to solve such problems, this paper introduces a better phishing detection framework in terms of usability, flexibility, and accuracy. The framework is a four-layer framework organized beginning with a simple-to-use interface for suspicious URL input. A data collection module collects URLs from diverse sources, such as user input, automated crawlers, and verified phishing datasets. These URLs are retrieved in a feature extraction layer which analyzes such URLs for lexical features, domain registration details, and content features such as scripts and page structure [2][3]. The features will be tested through an ensemble of machine learning models like Logistic Regression, Decision Trees, Support Vector Machines (SVM), Random Forests, and Deep Learning models [1][4]. Classification accuracy will be enhanced because all models will be combined together. The application will be implemented on a particular database layer where extracted features and classification outputs will be stored for continuous learning and enhancement. Reporting will be exhibited after classification, and a module will show in-depth analysis to users like confidence levels and decision justification. It gives the user the option to mark a wrong result to enhance the model even further in the long term [7]. Blacklist-based traditional techniques are insufficient because phishing URLs are continuously changing. Machine Learning (ML) offers adaptive techniques by learning URL patterns and differentiating

phishing attempts from valid links. Phishing detection through ML utilizes URL features, lexical patterns, and behavior analysis.[14].
This work proposes a machine learning-based system for phishing URL detection. The contributions include:

- ➢ A comparative analysis of different ML algorithms on phishing datasets.
- ➢ Feature extraction from URLs and dataset preparation.
- ➢ Evaluation of models using real-world phishing URL datasets.

This work proposes a machine learning-based phishing URL detection system, with contributions from data preparation to model assessment. Comparative assessment was carried out across a variety of machine learning algorithms, i.e., Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Logistic Regression, and Gradient Boosting. These algorithms were trained on phishing URL classification tasks based on key performance indicators such as accuracy, precision, recall, F1-score, and ROC-AUC to determine the most efficient detection technique. Features were derived from URLs based on lexical and heuristic features such as URL length, number of subdomains, presence of special characters, use of HTTPS, and some specific phishing-related keywords. Training the models included the use of real-world phishing datasets and testing and validating them through cross-validation techniques in order to ensure maximum generalizability. Results were also compared through confusion matrices and pie charts used to enhance the performance of each model. Key features that were significant in phishing detection were identified, and knowledge of how attackers construct deceptive URLs was gained.

This thesis is divided into six major chapters, each involving a major step of research towards phishing URL detection using machine learning techniques. Chapter 1 (Introduction) illustrates the background of phishing attacks, research motivation, objectives, scope, and the necessity of using machine learning for such a cyberattack. Chapter 2 (Literature Review) illustrates current literature in the domain, from traditional rule-based detectors to modern machine learning techniques, and the gap in research and motivation for this proposed method. Chapter 3 (Methodology) illustrates the entire research framework, from data gathering to feature extraction from URLs, data preprocessing, and machine learning techniques employed. Chapter 4 (Experimental Setup and Results) illustrates the implementation setup, tools and libraries employed, and then the performance metrics of model evaluation using accuracy, precision, recall, F1-score, and ROC-AUC. Chapter 5 (Discussion) interprets these outcomes and identifies the advantages & amp; disadvantages of each algorithm and important features responsible for detection. Chapter 6 (Conclusion and Future Work) summarizes the research findings, acknowledges its contribution, and suggests future extensions like using deep learning or live phishing protection frameworks.

2. LITERATURE REVIEW

Phishing attacks are becoming smarter and harder to spot, making them a growing problem in the world of cybersecurity. To keep up, researchers and tech experts have come up with a wide range of ways to detect these threats—starting from basic rule-based checks to more advanced machine learning systems that can learn and adapt over time. In this review, we'll take a look at what's already been done to fight phishing, how the approaches have evolved, and which machine learning models are proving to be the most effective at catching phishing attempts quickly and accurately.

2.1 Phishing

Most of the previously used systems have employed logistic regression Multinomial Naive Bayes and XG Boost. No user-interface & does not provide continuous results. The existing system models can be incorrect when sample size is small. And few of the research studies are based on rule-based and heuristic phishing URL detection. Blacklists and content analysis were one of the earliest solutions, but they are not adaptive.

Phishing Detection Survey, the current research examines different techniques employed to identify and counter phishing attacks. It categorizes them into four general categories: detection, offensive countermeasures, corrective measures, and preventative measures. It is not feasible that a single method can eradicate phishing threats, as the authors believe. They advocate a general, multi-level strategy to cybersecurity.[1] Nudges for Privacy and Security, the current paper examines the way small hints, or nudges, can assist users in making smarter decisions regarding online security and privacy. The paper also considers the ethical and design problems of employing such methods. The results indicate that nudges can enhance user behavior without restricting their autonomy. The study favors the application of persuasive design to facilitate safer online behavior.[2]}

Priming and Warnings in Social Engineering Attacks, in this paper, the impact of psychological cues—such as priming and warnings—on social engineering user resistance is studied. Empirical results show that such cues have a minimal impact on prevention of information disclosure. It raises questions regarding the efficacy of such methods in real-world contexts. The authors propose studying more effective and more resilient prevention strategies against manipulation.[3] Phishing Detection with PNN and K-Medoids, this paper proposes a phishing detection system based on Probabilistic Neural Networks (PNN) and K-Medoids clustering. The combined model not only provides outstanding detection rates—of over 97%—but also reduces

processing complexity by more than 40%. By combining the strengths of the two methods, the system is optimized for application in real time and shows robust general performance.[4] Malicious Domain Name Detection, this paper answers the challenge of detection of domains created by botnets, e.g., by Conficker. It uses a range of similarity and distance measures to differentiate valid and malicious domain names. The method is extremely accurate and has an extremely low rate of false positives, which gives it a significant amount of value in exposing suspicious domain activity characteristic of botnet attacks.[5]

Phishing Website Detection using Machine Learning, the paper explores some of the machine learning algorithms for phishing website detection and finds ensemble-based methods consistently perform better than single-model methods. The findings highlight the potential for the utilization of numerous models as a technique for making predictions more robust and augmenting cybersecurity defense.[6] Lizhen Tang and Qusay H. Mahmoud provide an overview of various machine learning techniques employed for the detection of phishing websites. It brings forth existing trends within the field and highlights directions for future research. The work serves as an informative starting point for researchers wishing to become familiar with the landscape of ML for phishing detection.[7] A. Karim et al. in their paper "Phishing Detection System Through Hybrid Machine Learning on URL" propose a hybrid machine learning model specifically for phishing detection by the analysis of the URL. The hybrid model enhances accuracy and reduces false positives effectively, serving as a strong model to implement practically within phishing prevention systems.[8] Faisal S. Alsubaei et al. utilizes deep learning within a hybrid setup for improving phishing detection performance. The model synergizes the strengths of the advantages of deep learning methods to achieve enhanced detection rates and facilitates cybercrime forensics.
[9] S. Remya et al., in their paper An Effective Detection Approach for Phishing URL Using ResMLP, employ a ResMLP (Residual Multi-Layer Perceptron) architecture for phishing URL classification. The model is unique with its high accuracy and strength, particularly against methods designed to evade detection systems.[10] Ahmad Sahban Rafsanjani et al. propose a new framework for malicious URL detection. It employs a priority coefficient technique combined with advanced feature evaluation to improve the accuracy of detection, thus making the system more efficient in detecting threats.[11] Valentine Adeyemi Onih presents a machine learning model for phishing detection. The study is centered on combining various ML approaches to enhance the overall capacity to detect phishing and fight online threats more effectively.[12] JianTing Yuan et al. propose a new method of malicious URL detection using a joint machine learning model. The method enhances the performance and accuracy of current detection systems by integrating the strengths of various models into a single monolithic system.[13]]

Haya T. Alhuraib et al. conduct a comprehensive review of machine learning-based solutions for malicious URL detection. The article presents current methods, their merits and demerits, and suggests areas of future research improvement in threat detection using ML.[14] Malak Aljabri et al. focus on phishing website detection using machine learning techniques. It discusses various ML-based techniques that assist in the detection of phishing attacks with emphasis on detection accuracy improvement and response to threats in real-time.[15] Ru Zhang et al. introduce an approach for APT attack traffic classification, especially in cases where data is scarce. With the development of two statistical anomaly features, their approach enhances APT attack detection accuracy in small-sample sets of traffic.[16]

S.no	Authors	Focus Area	Key Findings
1	Andrew Jones, Mahmoud Khonji, Youssef Iraqi [1]	Phishing Mitigation	Surveys phishing mitigation techniques, categorizing them into detection, offensive defense, correction, and prevention. No single solution effectively mitigates all phishing vulnerabilities.
2	Alessandro Acquisti, Idris Adjerid, Rebecca Balebako, Laura Brandimarte, Lorrie Faith Cranor, Saranga Komanduri ,Pedro Giovanni Leon, Norman Sadeh, Florian Schaub [2]	Privacy and Security Decision-Making	Analyzes how soft paternalistic interventions (nudges) assist users in making better privacy and security choices, while discussing ethical and design challenges.
3	F.J. Overink, M. Junger, L. Montoya [3]	Social Engineering	Investigates the effectiveness of priming and warnings in preventing social engineering attacks. Found that neither intervention significantly influenced disclosure behavior, with possible adverse effects.

4	M. El-Alfy, El-Sayed M [4]	Phishing Detection	Proposes a phishing detection model using Probabilistic Neural Networks (PNN) and K-Medoids clustering. Achieves >97% accuracy with >40% complexity reduction.
5	Shuang Hao, Luca Invernizzi, Yong Fang, Christopher Kruegel, Giovanni Vigna [5]	Botnet Detection	Develops a methodology to detect algorithmically generated domain names used by botnets (e.g., Conficker). Uses multiple distance metrics and achieves minimal false positives.
6	Ammara Zamir et al. [6]	Phishing Detection	Evaluates multiple machine learning models for phishing detection, comparing their accuracy and performance. Highlights the effectiveness of ensemble techniques.
7	Lizhen Tang & Qusay H. Mahmoud [7]	Phishing Detection	Provides an overview of machine learning techniques used for phishing website detection. Discusses challenges and future research directions.
8	A. Karim et al. [8]	Hybrid ML for Phishing Detection	Proposes a hybrid ML-based phishing detection model that improves accuracy while reducing false positives.
9	Faisal S. Alsubaei et al. [9]	Deep Learning for Phishing Detection	Utilizes deep learning for phishing detection with a hybrid model, improving detection rates.
10	S. Remya et al. [10]	ML-Based URL Classification	Uses ResMLP for phishing URL classification, achieving high accuracy and robustness against evasion techniques.
11	Ahmad Sahban Rafsanjani et al. [11]	Malicious URL Detection	Proposes a priority coefficient-based approach to detect malicious URLs with improved feature evaluation.
12	Valentine Adeyemi Onih [12]	ML for Phishing Detection	Develops a new phishing detection model that integrates various ML techniques for improved threat mitigation.

13	JianTing Yuan et al. [13]	Malicious URL Detection	Introduces a joint ML model that enhances malicious URL detection performance
14	Haya T. Alhuraib et al. [14]	ML-Based Threat Detection	Reviews ML-based techniques for detecting malicious URLs and suggests areas for improvement
15	Malak Aljabri et al. [15]	Phishing Website Detection	Explores ML-based approaches for phishing detection, focusing on accuracy and real-time identification.
16	Ru Zhang et al. [16]	APT Attack Classification	Proposes statistical anomaly features to classify APT attack traffic in small datasets, improving detection accuracy.

2.2 Machine Learning Models

In computer science, Machine Learning is an application of artificial intelligence, which makes computers learn using data and hence, improve on their own without programming. Its functionality seeks to articulate an integration of data with prediction and insight techniques, which are statistical for use with data.

Learning Phase: The system analyzes data to discover patterns and build a model.

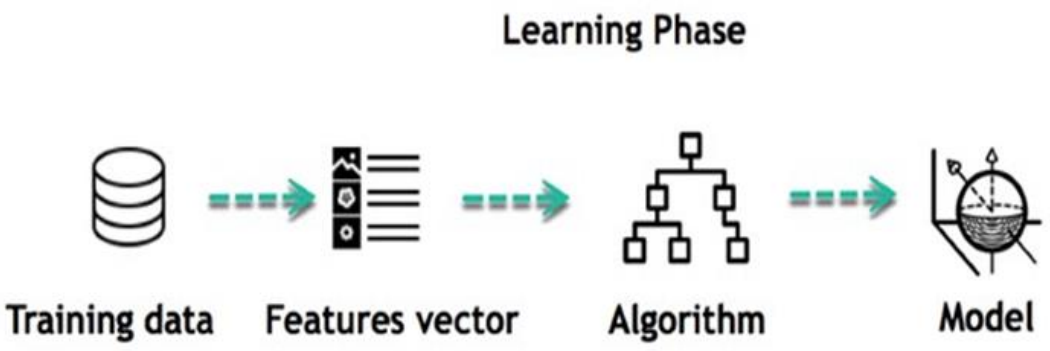


Fig :1 Learning Phase of ML

Inference Phase: The trained model makes predictions on new data.

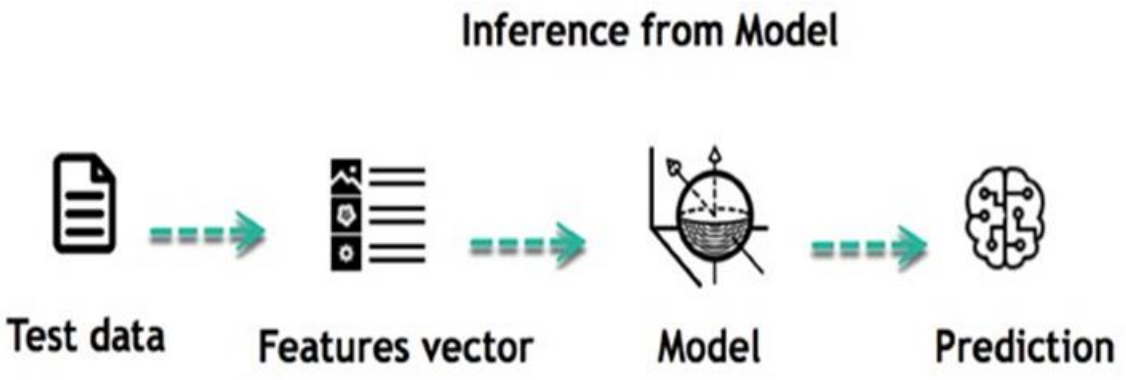


Fig: 2 Inference Phase of ML

Gradient Boosting Classifier (model pkl):

In machine learning, a Gradient Boosting Classifier is a powerful ensemble technique used for both classification and regression problems. When saved as a **.pkl file (Pickle format)**, it represents a pre-trained model that can be loaded and used directly for making predictions without the need to retrain it.
The Gradient Boosting Classifier builds an ensemble of decision trees in a sequential manner. Each new tree is trained to correct the errors made by the previous trees. Over time, this "boosting" process improves the model's accuracy significantly.

There are several high-performance implementations of Gradient Boosting:

- XGBoost (Extreme Gradient Boosting): Known for its speed, regularization, and scalability. It is widely used in Kaggle competitions and industry applications.
- LightGBM (Light Gradient Boosting Machine): Developed by Microsoft, it uses histogram-based algorithms and is optimized for speed and efficiency on large datasets.
- CatBoost: Developed by Yandex, CatBoost is particularly efficient in handling categorical features and often requires less preprocessing.
- Scikit-learn's GradientBoostingClassifier: A straightforward and user-friendly implementation in Python’s sk learn library, suitable for small to medium-sized datasets.

The .pkl Format: Saving and Loading Models

In Python, a trained model can be serialized and saved to a .pkl (Pickle) file using the pickle or joblib library. This allows developers to save time and computational resources, as the model doesn't need to be retrained every time it is used.

Gradient Boosting is Effective for This Task:

- High Accuracy: It can capture complex relationships in feature space, leading to better classification performance.
- Robustness to Imbalanced Data: Phishing datasets are usually skewed; Gradient Boosting handles this well by focusing on hard-to-classify samples.
- Feature Sensitivity: GBCs naturally rank features by importance, helping developers understand what signals are most predictive of phishing behavior.
- Real-Time Deployment: When saved as a .pkl file, the model can be embedded in browser extensions, email filters, or network intrusion detection systems for fast threat detection.

3. Methodology

3.1 Problem Statement

Phishing is the most serious of cybersecurity attacks that fool users into divulging sensitive information by impersonating authentic websites using malicious URLs. Conventional detection techniques—such as blacklists and rule-based systems—are incapable of handling the high evolution rate and obfuscation techniques used by attackers. These are reactive systems that are incapable of detecting newly hatched (zero-day) phishing URLs and lack flexibility to deal with the dynamic aspect of phishing. Phishing, in the age of digitalization, has become one of the most pervasive and menacing cyber attacks that are inflicting people, organizations, and institutions around the world. With the help of cleverly designed deceptive URLs resembling authentic sites, attackers can mislead individuals into divulging confidential information such as login credentials, bank details, and personal data. These types of phishing attacks are increasingly becoming sophisticated and generally manage to bypass conventional defenses by repeatedly altering their structure and function.

Since phishing detection has traditionally been limited, obviously there needs to be more capable, flexible systems that can pick up on phishing URLs before they can cause damage. Machine learning offers a solution that is at least worth a try—it can process huge amounts of data, detect subtle trends, and improve over time as new patterns of threats emerge. But it's not a case of simply plugging in a model. A big part of the issue is determining which algorithms to use, how to pull something useful out of a URL, and how to generate predictions that it's safe to take action on without generating too many false positives.

This project meets that challenge by exploring several machine learning methods to improve phishing URL detection. The aim is to create a smart system that can learn automatically from the structure of web links, derive useful features, and accurately differentiate between secure and suspicious URLs. Through this, the aim is to create a better and more proactive defense against phishing attacks—making web browsing, as well as email use, safer for everyone.
Machine learning offers an exciting solution through the ability to automatically extract features, identify patterns, and update models continually. The fundamental question this research will address is: How do machine learning systems most effectively optimize phishing URL detection system accuracy, efficiency, and responsiveness compared to traditional methods?

3.2 System Architecture

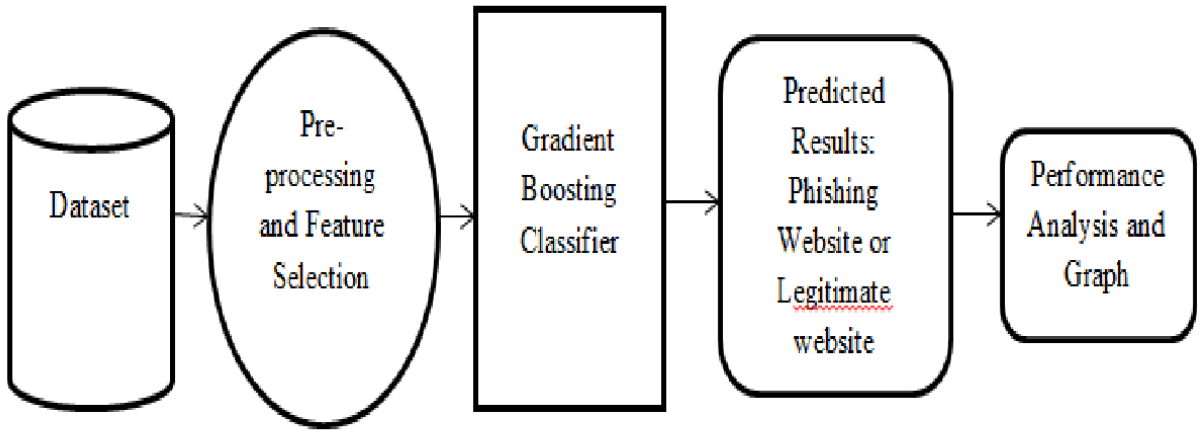


Fig : 3 System Architecture

We have developed our project using a website as a platform for all the users. This is an interactive and responsive website that will be used to detect whether a website is legitimate or phishing.

This website is made using different web designing languages which include HTML, CSS, Javascript and Flask framework in Python. The basic structure of the website is made with the help of HTML. CSS is used to add effects to the website and make it more attractive and user-friendly. It must be noted that the website is created for all users, hence it must be easy to operate with and no user should face any difficulty while making its use.

After the system is trained with the dataset, the classifier identifies the given URL dependent on the preparation information, that is if the site is phishing it prompts the user that the website is phished and if genuine, it prompts the user that the website is legitimate. We have detected phishing websites using Gradient Boosting Classifiers with an accuracy of 97%.

Process:

This flowchart illustrates the steps in a machine learning process that starts with the user input and ends with producing a prediction or output. The steps are as follows in detail:
Accepts zip from user (URL): The system starts by accepting a ZIP from the user, typically a URL. The ZIP likely stores data or files for further processing.Calls Features Extraction library:

The system then employs a feature extraction library to find useful information (features) in the input files. The features are important to prediction.Converts all the features of the URL: The step processes the features derived from the URL content only, converting them into a usable format. Converts into a numpy array: The processed and extracted features are in the shape of a NumPy array. NumPy arrays are Python's default numerical data structure, especially in machine learning. Run the gradient boosting using model.pkl: The NumPy array is passed into a pre-trained gradient boosting model in model.pkl. Gradient boosting is a high-performance machine learning algorithm for regression or classification. Get the result: Finally, the model processes the input and returns a result, for example, a prediction or classification.

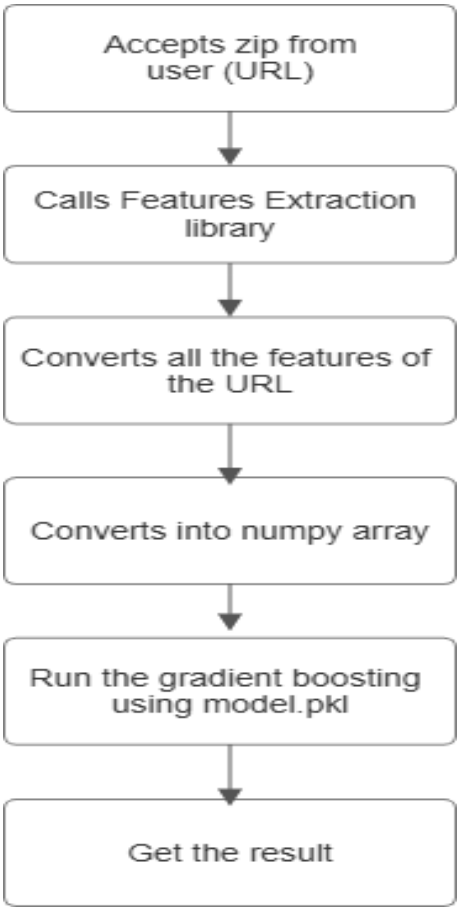


Fig : 4 Process diagram

3.3 Model Selection

Models such as Random Forest, Gradient Boosting, SVM, and Logistic Regression were tested. Gradient Boosting was selected for its robustness and high precision in detection.

Boosting is an ensemble technique that assembles weak learners, usually decision trees, into a strong ensemble predictive model. First, a basic model is fitted to the dataset. The data records, which were misclassified from the previous model, would receive increased weights in order that the subsequent model focuses on these. This process is performed repeatedly, wherein for every new model, the intention is to rectify the mistake of past models. Finally, all the predictions would be combined-given weighted votes (for classification)-to form the final decision. Thus, while increasing the overall model accuracy, it also helps in reducing biases, as well as variance [2][3][16].

Gradient Boosting Algorithm

In a nutshell, Gradient Boosting techniques create their models in a sequential manner, in which each model is learned or trained to minimize the loss function of the previous model using gradient descent. For a classification task such as phishing detection, Gradient Boosting Classifier will be employed, where many loss functions are generally log-loss. For regression, the function will be like in the case of MSE (Mean Squared Error) [2][5][15].

The model is very effective for imbalanced datasets with nonlinear relationships and complex feature interactions, which make this model applicable for phishing URL detection scenarios [1][4][7].

Gradient Boosting Classifier:

Gradient Boosting is an excellent machine learning algorithm that constructs a powerful prediction model from a huge set of tiny, simple models—usually decision trees. Instead of trying to learn everything at a time, it learns in incremental steps. It trains the subsequent new tree so that it fixes the mistakes made by previous ones. It accomplishes this via a technique called gradient descent that helps the model decide how best to minimize its mistakes. For something such as predicting phishing URLs, Gradient Boosting is excellent. It's tested on how

well it can predict using something referred to as log-loss, especially when it's trying to predict probabilities. Through learning and improving through numerous iterations, it becomes extremely proficient at detecting sneaky patterns in data—things other models may not catch.

One of the primary reasons that Gradient Boosting performs so effectively on phishing is that it is effective with imbalanced data. In nearly every real-world application, there would be hundreds and even thousands of times more normal URLs than suspicious URLs. Gradient Boosting is not bothered about that imbalance. It can also learn intricate hidden patterns and relationships in the data without too much hard grueling feature optimization. For example, it can learn to identify that some patterns of symbols occurring in a URL or some patterns of domains would be phishing.[17][24][22]. Even newer Gradient Boosting implementations such as XGBoost, LightGBM, and CatBoost push it further. They are designed to be efficient, accurate, and fast. These models possess real-world characteristics such as integrated overfitting prevention, intelligent categorical data processing, and efficient processing, which qualify them for big-scale or real-time usage.

Actually, Gradient Boosting models have outperformed more traditional methods like Logistic Regression and Naive Bayes consistently in phishing detection. They're widely used in security programs, i.e., browser extensions or email filters, as they excel in classifying malicious URLs—even recently encountered ones.[18][21][20].

3.4 Feature Extraction

```
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, datetime
import time
from dateutil.parser import parse as date_parse
from urllib.parse import urlparse
```

Fig: 5 Importing Libraries

✓ Ipaddress

- functionality: controlling both IPv4 and IPv6 addresses.
- key uses:
  i. validation of ip address.
  ii. check whether the ip address lies in a private or public range.

✓ re (regular expressions)

- functionality: for performing string operations based on a particular pattern.
- key uses: search, extract, or modify parts of a string.

✓ urllib.request

- functionality: allows downloading data from URLs.
- key uses:
  i. to send requests to web servers.
  ii. to download the contents of a web page.

✓ BeautifulSoup (from bs4)

- functionality: extracts data from HTML and XML documents.
- key uses: web scraping and extraction of data from structured documents

✓ Socket

- functionality: for basic networking.
- key uses:
  i. to convert domain name to an ip address.
  ii for TCP/IP communication.

✓ Requests

- functionality: for simplified HTTP requests and responses.
- key uses:

i. to send HTTP requests - get, post;

ii. to receive API responses.

✓ googlesearch (from google search)

- functionality: to search in google.
- key uses: to fetch the first top results according to the designation.

✓ Whois

- functionality: to retrieve ownership and registration records of domains.
- key uses: to get domain expiry and creation details.

✓ Datetime

- functionality: date and time.
- key uses:

i. retrieve the current date and time;

ii. timestamp formatting.

✓ Time

- functionality: for time-related operations
- key uses:

i. performing delays in execution;

ii. fetching timestamps.

✓ dateutil.parser (parse from dateutil.parser)

- functionality: converts various date formats into Python's datetime format.
- key uses: parsing and standardization of various date formats.

✓ urllib.parse (urlparse from urllib.parse)

- functionality: designed to break down the URL and manipulate it.
- key uses: extracting domain, path, and query parameters from a URL

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

Fig: 6 Class Attributes and Initialization

The FeatureExtraction class has a class-level list features, which stores extracted features.

Inside __init__, several instance variables are initialized to store URL details like:

- url → The provided website link.
- domain → Extracted domain name.
- whois_response → WHOIS lookup response.
- urlparse → Parsed URL components.
- response → HTTP response from the server.
- soup → Parsed HTML content.

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

Fig: 7 Fetching web page data

- The program attempts to fetch the webpage content using the requests module.

- If successful, the HTML response is parsed using BeautifulSoup for further analysis.
- If an error occurs (e.g., site is unreachable), the exception is caught, and the process continues.

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

Fig: 8 Parsing URL and Extracting WHOIS Information

- Parsing the provided URL using urlparse() to separate domain names.
- A WHOIS search is used to find domain registration data.
- No exception is thrown if the WHOIS request is unsuccessful (i.e., the domain is not registered).

Each function has a corresponding return value (presumed to be a score or a boolean) which is appended to the features list.

These consist of:

- If the URL is referencing off of an IP address instead of a domain (UsingIp()).
- If its URL is too short or too long (longUrl(), shortUrl()).
- If it contains suspicious symbols (symbol()).
- Whether the URL is redirecting more than once (redirecting()).
- Checking if the domain has a hyphenated prefix or suffix (prefixSuffix()).
- Verification of the number of subdomains (SubDomains()).
- If the protocol is HTTPS (Https()).
- Verifying the registration period of the domain (DomainRegLen()).
- Verifying whether it has a favicon (Favicon()).
- Confirmation of non-standard ports (NonStdPort().
- Whether the domain name itself includes "HTTPS" fraudulently (HTTPSDomainURL()).
- Validating requests to external URLs (RequestURL()).

▪ Phishing URLs check in anchor tags (AnchorURL()).
▪ Looking for malicious links inside JavaScript (LinksInScriptTags()).
▪ Investigating the form handler behavior (ServerFormHandler()).
▪ Obtaining email addresses on the site (InfoEmail()).
▪ Checking whether the URL pattern is abnormal (AbnormalURL()).
▪ Detection of automated site redirection (WebsiteForwarding()).
▪ Verifying whether status bar activity is being addressed (StatusBarCust()).
▪ Malicious pop-up verification (UsingPopupWindow()).
▪ Checking if the page uses hidden iframes (IframeRedirection()).
▪ Get the domain age (AgeofDomain()).
▪ Check DNS record history (DNSRecording()).
▪ Gathering website traffic statistics (WebsiteTraffic()).
▪ PageRank verification for the website (PageRank()).
▪ Verifying if the site is indexed in Google (GoogleIndex()).
▪ Identifying the number of external links referring to the page (LinksPointingToPage()).
▪ Retrieving statistical reports for the website (StatsReport()).

3.5 Flask-Based Web App for Phishing Detection

```
from flask import Flask, request, render_template
import numpy as np
import pandas as pd
from sklearn import metrics
import warnings
import pickle
warnings.filterwarnings('ignore')
from feature import FeatureExtraction
```

Fig: 9 Libraries of App.py

▪ Flask: Used to create the web server and handle HTTP requests.
▪ request: Allows data sent by the user, such as a URL input, to be accessed by the app.
▪ render_template: Used to render HTML files to create the user interface.

▪ numpy (np) & pandas (pd): These libraries aid in numerical computations as well as handling data.
▪ sklearn.metrics: Used to calculate the performance of a machine learning model.
▪ warnings: Prevents unnecessary warning messages.
▪ pickle: Handles the loading of a pre-trained machine learning model for prediction.
▪ from future import FeatureExtraction imports a feature extraction custom class or function that extracts important features from a provided URL.

Initialize Flask Application:

i. app = Flask(__name__) creates an instance of a Flask web application.

Define Routes (URLs for Web Pages):

i. Each @app.route() links a URL with a function.

ii. These functions make use of render_template() to render HTML pages.

Static Pages Included:

i. / or /first → Shows first.html (most probably the home page).

ii. /performance → Shows performance.html.

iii. /chart → Shows chart.html.

iv. /login → Shows login.html.

v. /upload → Upload files from upload.html.

vi. /index → Loads index.html.

```
@app.route("/posts", methods=["GET", "POST"])
def posts():
if request.method - "POST":
url = request.form["url"]
obj = FeatureExtraction (url)
x= np.array(obj.getFeaturesList()).reshape (1,30)
y_pred =gbc.predict(x) [0]
#1 is safe
#-1 is unsafe
print (y_pred)
y_pro_phishing -gbc.predict_proba (x) [0,0]
y_pro_non_phishing gbc.predict_proba (x) [0,1]
print(y_pro_phishing)
print(y_pro_non_phishing)
#if (y_pred ==1):
pred "It is (0:.2f) & safe to go ".format(y_pro_phishing*100)
return render_template('result.html',xx =round (y_pro_non_phishing, 2), url=url)
return render_template("result.html", xx =-1)
```

Fig : 10 Phishing Detection Logic

▪ Receives URL Input:

i.input_url = request.form["url"] gets user-entered URL in an HTML form.

- Extracts Features:
  i.FeatureExtraction(input_url) processes the URL.
  ii.getFeaturesList() gives 30 numeric features that represent security attributes.
  iii.reshape(1, 30) converts the list of features into a form usable by ML algorithms.
- Predicts If URL is Phishing
  i. model.predict(extracted_features)[0]: Identified the URL based on the learned model.
  ii.model.predict_proba(extracted_features): Provides probability scores.
  iii.safe_probability → Probabilities that the URL is secure.
  iv.phishing_probability → Chances that the URL is phishing.
- Shows Results in HTML:
  i.render_template('result.html', xx=round(phishing_probability, 2), url=input_url)
  ii.Pass the result to result.html for display.
- GET Request Behavior:
  i.In case the request is GET, it defaults to showing result.html with xx=-1.

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

Fig:11 Run the Flask App

- Runs the Web App:
  i. app.run(debug=True): Starts the Flask server in debug mode.
- Debug mode:
  i. Automatically restarts the server when changes are made.

3.6 Phishing Detection Webpage

Our project is a web-based Phishing Website Detection System built using HTML, CSS, JavaScript (with some Flask/Python backend implied through url_for() references). It includes Login, Upload, and Result pages, allowing a user to login and upload a dataset.

3.6.1 Login Page (login.html)

Purpose:

The login page is the gateway to access the core functionality of the phishing detection system. It ensures only authorized users (like admin or analysts) can upload datasets and view results.

Key Features:

- **User Interface**: Clean layout with fields for Username and Password.
- **JavaScript Logic**:
    - A function named login() is triggered when the **Login** button is clicked.
    - The script checks if the username and password match predefined values (in this case, "admin").
    - If credentials are valid: The user is redirected to the upload.html page.
    - If credentials are wrong: An alert is shown saying **"Invalid Credentials!"**.

Technologies Used:

- **HTML/CSS**: For layout and design.
- **Bootstrap**: For responsive layout and components.
- **JavaScript**: For validating user input and handling redirection.
- **Jinja Templating** ({{ url_for('...') }}): For Flask route handling.

Structure:

- Navbar with **Home** and **Login**
- Background header with animation
- Login form centered with fields for **Username** and **Password**
- On-click JavaScript login() function:
    - Checks if uname === "admin" and pwd === "admin"
    - If correct: redirects using window.location = "{{url_for('upload')}}"

I designed this using HTML templates, added a background image with inline CSS, and used a basic JS function to simulate login. It's all client-side for simplicity.

3.6.2 Upload Page (upload.html)

Purpose:

Allows users to upload a dataset (typically a .csv file containing URLs) that will be processed by the machine learning model to detect phishing URLs.

Key Features:

- **File Input Form**:
    - The form uses enctype="multipart/form-data" to handle file uploads.
    - When submitted, the file is sent via POST to the /preview route on the server (http://localhost:5000/preview).
- **Server-Side Processing** (handled in Flask backend):
    - The server receives the file.
    - Parses and preprocesses the URLs.
    - Feeds them into the trained ML model (e.g., Gradient Boosting Classifier).
    - Predicts whether each URL is **legitimate** or **phishing**.

Technologies Used:

- **HTML/Bootstrap**: For form layout and design.
- **Flask Backend** (implied): Receives file input and runs phishing detection logic.

Structure:

- Navbar + animated header

  Upload form using:

<form action="http://localhost:5000/preview" method="post" enctype="multipart/form-data">

- Upload button
- File input field

Created this page to allow input of the dataset file after login. Using Flask, you'd have a Python function to receive this file, process it, and then redirect to the result page with output.

3.6.3 Result Page (result.html)

Purpose:

Displays the prediction results of the uploaded URLs – whether each URL is **phishing** or **legitimate**.

Key Features:

- While the HTML content resembles the login page (likely reused styling/layout), in a full implementation:
    - This page would list the results in a table or report format.
    - May include columns like:
        - **URL**
        - **Predicted Label** (Phishing/Legitimate)
        - **Confidence Score**
    - Could also display **graphs**, **charts**, or **statistics** (e.g., % of phishing URLs in the file).

Technologies Used:

- **HTML/CSS**: UI presentation.
- **Jinja Templates** (assumed): Used to render dynamic results sent from the Flask backend.

- **JavaScript (optional)**: Could be used to enhance interactivity, like filtering results.

Structure:

- Navbar, header, and login form again (this should probably be replaced by output)
- Currently it doesn't contain result logic — needs a section to display output (like a table or message showing phishing vs. legitimate URLs)

You likely plan to extract results in Python, then pass it to this HTML page via Flask:

Python:

```
return render_template("result.html", result=predicted_value)
```

Then in HTML:

```
<h3>Result: {{ result }}</h3>
```

How Everything Connects

1. **Login → Upload → Result**
    - The user logs in securely.
    - They upload a URL dataset.
    - The system processes the file using a trained machine learning model (likely Gradient Boosting Classifier).
    - Predictions are displayed in the result view.
2. **Back-End Flask Logic (not shown here but assumed)**:
    - Routes: /login, /upload, /preview, /result
    - Handles:
        - Login sessions

- File parsing and cleaning
- Feature extraction
- Feeding into ML model
- Sending results back to the frontend

4. Results and Discussion

Now that our machine learning model for phishing URL detection has been developed and trained, the next step is to see how well it performs in practice. In this section, we'll walk through how the system was set up, what metrics we used to evaluate it, how the model was implemented, and what the results looked like. Our goal is to show how effective the model is and how it stands up against more traditional detection methods. The upcoming parts will break down the technical environment we used, explain the performance indicators, and dive into what the results reveal.

4.1 System Environment

To build the phishing website detection system, we used Python—a powerful and flexible programming language that's widely adopted in data science and machine learning projects. The entire implementation was designed to be efficient, scalable, and easy to test.

System Specifications:

The development was carried out on a standard personal computer with the following setup:

- Processor: Intel Core i5
- Memory (RAM): 8 GB
- Operating System: Windows 10 (or your OS)
- Programming Language: Python 3.x

Even though the hardware was relatively modest, it proved sufficient for processing and training our model on the given dataset.

Libraries and Tools Used:

To make our system smart and efficient, we leveraged a variety of popular Python libraries:

- Scikit-learn: Used for handling tasks such as data preprocessing, splitting datasets, and evaluating models. It also provided some basic classifiers for comparison.
- Pandas: Helped in reading, cleaning, and manipulating the dataset. It's especially useful when working with tabular data like URLs and their extracted features.
- XGBoost: A powerful machine learning algorithm known for its speed and performance. It was our main classifier for detecting phishing websites.

- Gradient Boosting Classifier: Provided by Scikit-learn, this model was used to compare with XGBoost and to test ensemble methods.

Flask Framework:

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

The Http protocol is the foundation of data communication in the world wide web. Different methods of data retrieval from specified URLs are defined in this protocol.

The following table summarizes different http methods −

By default, the Flask route responds to the GET requests. However, this preference can be altered by providing method arguments to route() decorator.

In order to demonstrate the use of POST method in URL routing, first let us create an HTML form and use the POST method to send form data to a URL.

Evaluation Metrics:

To understand how well our model was performing, we didn't just rely on accuracy. We used a combination of industry-standard metrics:

1. Accuracy: Measures overall correctness.

   Accuracy = (TP + TN) / (TP + TN + FP + FN)

2. Precision: Measures how many predicted phishing URLs were actually phishing.

   Precision = TP / (TP + FP)

3. Recall (Sensitivity or True Positive Rate): Measures how many actual phishing URLs were correctly identified.

   Recall = TP / (TP + FN)

4. F1-Score: Harmonic mean of Precision and Recall. Balances both in one metric.

   F1-Score = 2 × (Precision × Recall) / (Precision + Recall)

These metrics helped us get a clearer picture of model performance and ensured we weren't just getting lucky with the predictions.

4.2 Implementation

To actually build the phishing detection system, we followed a clear step-by-step process—starting from gathering the right data all the way to training and testing the machine learning model. Each step plays an important role in making sure the system works well and gives reliable results. In the sections below, we'll walk through each part of the implementation, starting with how we collected and got the data ready for training.

4.2.1 Data Collection:

In the first module we develop the data collection process. This is the first real step towards the real development of a machine learning model and collecting data. This is a critical step that will cascade in how good the model will be, the more and better data that we get, the better our model will perform.

There are several techniques to collect the data, like manual interventions,web scraping. The dataset is referred from the popular dataset repository called kaggle. The following is the Kaggle dataset link for the Detection of Phishing Websites Using Machine Learning.

Kaggle Dataset Link:

https://www.kaggle.com/datasets/jayaprakashpondy/phishing-websites-feature-dataset

4.2.2 Dataset:

The dataset consists of 11054 individual data. There are 32 columns in the dataset, which are described below.

Feature	Type	Possible Values
Index	Index ID	-
UsingIP	Categorical	{-1, 1}
ShortURL	Categorical	{1, -1}
LongURL	Categorical	{1, 0, -1}

Symbol@	Categorical	{1, -1}
Redirecting://	Categorical	{-1, 1}
PrefixSuffix-	Categorical	{-1, 1}
HTTPS	Categorical	{-1, 0, 1}
SubDomains	Categorical	{-1, 0, 1}
Favicon	Categorical	{1, -1}
DomainRegLen	Categorical	{-1, 1}
NonStdPort	Categorical	{1, -1}
RequestURL	Categorical	{1, -1}
HTTPSDomainURL	Categorical	{-1, 1}
AnchorURL	Categorical	{-1, 0, 1}
ServerFormHandler	Categorical	{-1, 0, 1}
LinksInScriptTags	Categorical	{-1, 0, 1}
InfoEmail	Categorical	{-1, 1}
WebsiteForwarding	Categorical	{0, 1}
AbnormalURL	Categorical	{-1, 1}
StatusBarCust	Categorical	{-1, 1}
UsingPopupWindow	Categorical	{-1, 1}
DisableRightClick	Categorical	{-1, 1}
IframeRedirection	Categorical	{-1, 1}
DNSRecording	Categorical	{-1, 1}
AgeofDomain	Categorical	{-1, 1}
WebsiteTraffic	Categorical	{-1, 0, 1}
GoogleIndex	Categorical	{-1, 1}
PageRank	Categorical	{-1, 1}
LinksPointingToPage	Categorical	{-1, 0, 1}
Class	Categorical	{-1, 1}
StatsReport	Categorical	{-1, 1}

Each URL in the dataset was analyzed for features like:

- URL length
- Use of HTTP vs HTTPS
- Presence of suspicious symbols or keywords
- Domain age, subdomain depth, etc.

These features were crucial in helping the model learn to differentiate between phishing and legitimate URLs.

4.2.3 Data Preparation:

Wrangle the data and prepare it for training. Clean that which may require it (remove duplicates, deal with missing values, correct errors, normalization, data type conversions, etc.).

Randomize data, which erases the effects of the particular order

Visualize data to help detect relevant relationships between the variables or class imbalances (bias alert!), or perform other exploratory analysis.

Split it into training and evaluation sets.

4.2.4 Model Selection:

Implemented model GBC would be a beneficial classifier for phishing URL detection compared to its training accuracy of 98.9%.

decision. Thus, while increasing the overall model accuracy, it also helps in reducing biases, as well as variance [2][3][19].

The model is very effective for imbalanced datasets with nonlinear relationships and complex feature interactions, which make this model applicable for phishing URL detection scenarios [1][4][7][23].

Analyze and Prediction:

In the actual dataset, we chose only 30 features:

Feature	Type	Possible Values
UsingIP	Categorical	{-1, 1}
ShortURL	Categorical	{1, -1}
LongURL	Categorical	{1, 0, -1}
Symbol@	Categorical	{1, -1}
SubDomains	Categorical	{-1, 0, 1}
Redirecting://	Categorical	{-1, 1}
PrefixSuffix-	Categorical	{-1, 1}
DomainRegLen	Categorical	{-1, 1}
HTTPS	Categorical	{-1, 0, 1}
Favicon	Categorical	{1, -1}
HTTPSDomainURL	Categorical	{-1, 1}
NonStdPort	Categorical	{1, -1}
RequestURL	Categorical	{1, -1}
LinksInScriptTags	Categorical	{-1, 0, 1}
ServerFormHandler	Categorical	{-1, 0, 1}
AnchorURL	Categorical	{-1, 0, 1}
InfoEmail	Categorical	{-1, 1}
WebsiteForwarding	Categorical	{0, 1}
AbnormalURL	Categorical	{-1, 1}
StatusBarCust	Categorical	{-1, 1}
UsingPopupWindow	Categorical	{-1, 1}
DisableRightClick	Categorical	{-1, 1}
IframeRedirection	Categorical	{-1, 1}
DNSRecording	Categorical	{-1, 1}
AgeofDomain	Categorical	{-1, 1}
WebsiteTraffic	Categorical	{-1, 0, 1}
GoogleIndex	Categorical	{-1, 1}

LinksPointingToPage	Categorical	{-1, 0, 1}
PageRank	Categorical	{-1, 1}
StatsReport	Categorical	{-1, 1}
Class	Categorical	{-1, 1}

4.3 Results & Analysis

This is a login page depicted in the image. It is mainly designed for detecting " Phishing Websites".

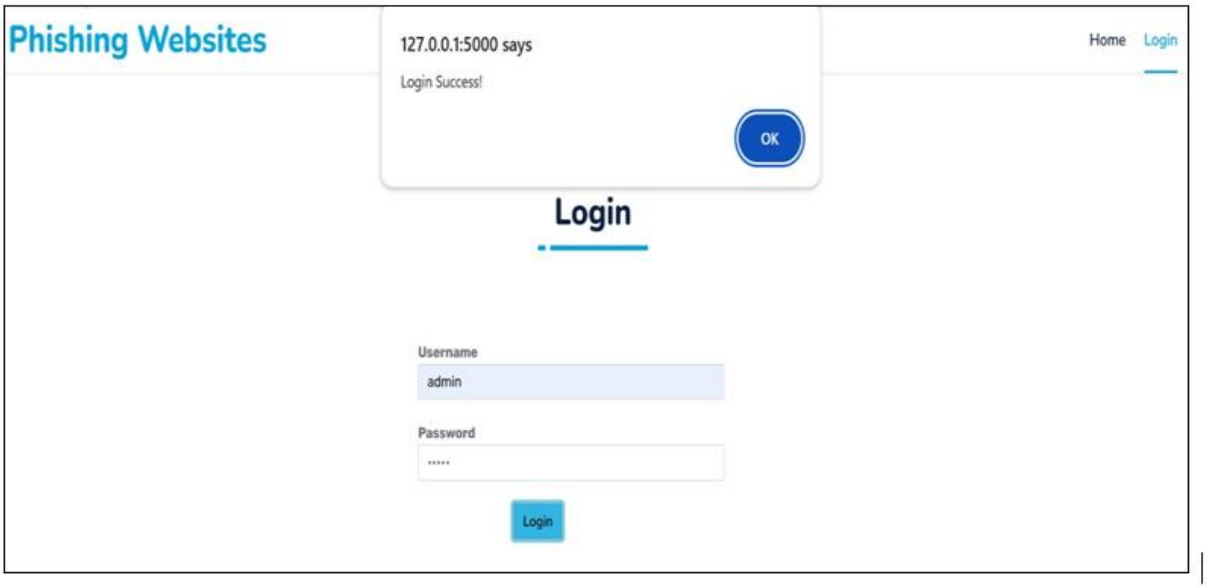


Fig:12 Login Page for the URL Detection Website

The main section of the page is titled "DETECTION OF PHISHING WEBSITES,". The main focus of this is to identify the Phishing websites. The title of the page displays as "Phishing Websites," with different navigation links like "Home" and "Login". The header contains the login features. This page consists of different fields like "Username" and "Password". A "Login" button allows the users to access the system.

This login page is an entry point for administrators and authorized users to interact with the phishing detection tool, it involves URL analysis, pattern recognition, and other methods.Where as the Username ,Password by defaultly set to “admin” only.After entering the credentials of “Login page” click on the “login” button after you will be directed in to the next page shown below.

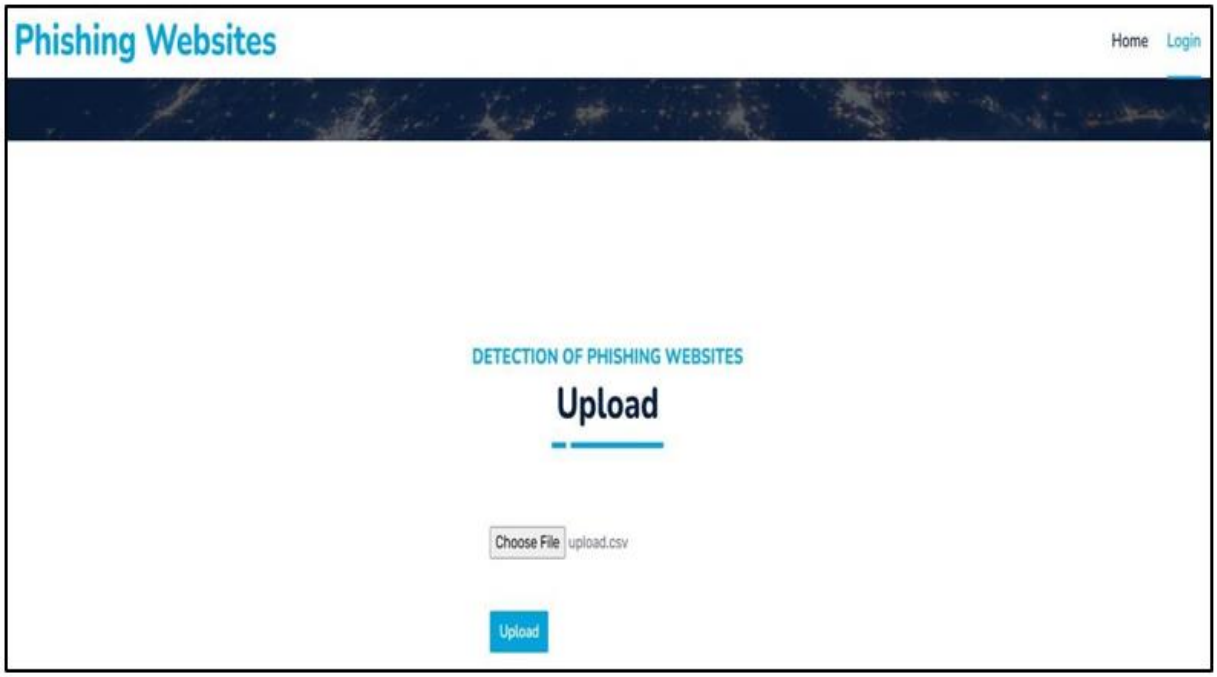
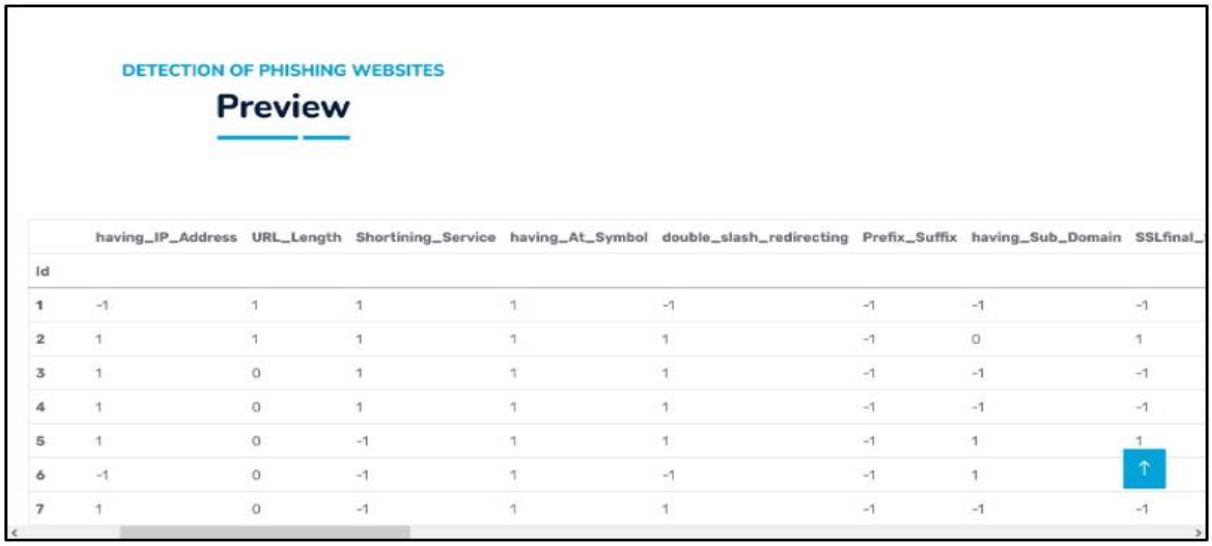


Fig:13 Upload Excel file

The "Phishing Websites" web-page is a phishing threat detection tool. The webpage has a file upload interface to upload files for scanning to detect potential phishing threats.It has a "Browse" button to choose files and an "Upload" button to start the scanning process. The main emphasis is on cybersecurity and online threat protection, enabling users to detect potentially malicious files. This utility is an asset to individuals and organizations alike, fortifying cybersecurity defences and protecting digital assets from phishing attacks.



Id	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_
1	-1	1	1	1	-1	-1	-1	-1
2	1	1	1	1	1	-1	0	1
3	1	0	1	1	1	-1	-1	-1
4	1	0	1	1	1	-1	-1	-1
5	1	0	-1	1	1	-1	1	1
6	-1	0	-1	1	-1	-1	1	
7	1	0	-1	1	1	-1	-1	-1

Fig:14 Preview of the Excel Sheet

After uploading the excel sheet you will get all the eleven thousand data features with an " Preview" page. Where will see all the features . And next will see a button at last " Click to Train Test".By clicking that ,the features get trained. After the training gets finished it shows "Training finished" at top,and the next step is testing.

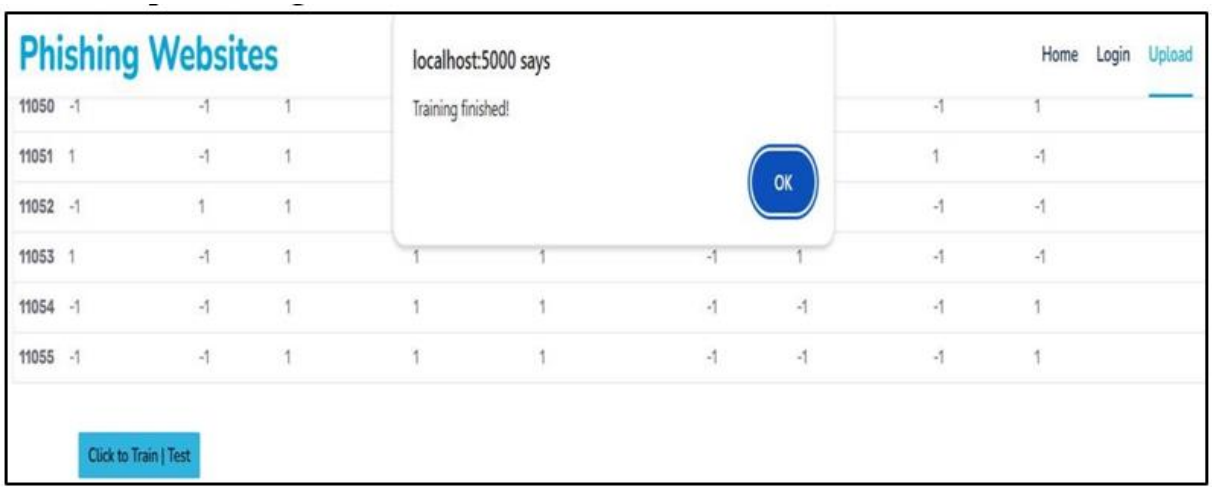


Fig:15 Model Training Completed

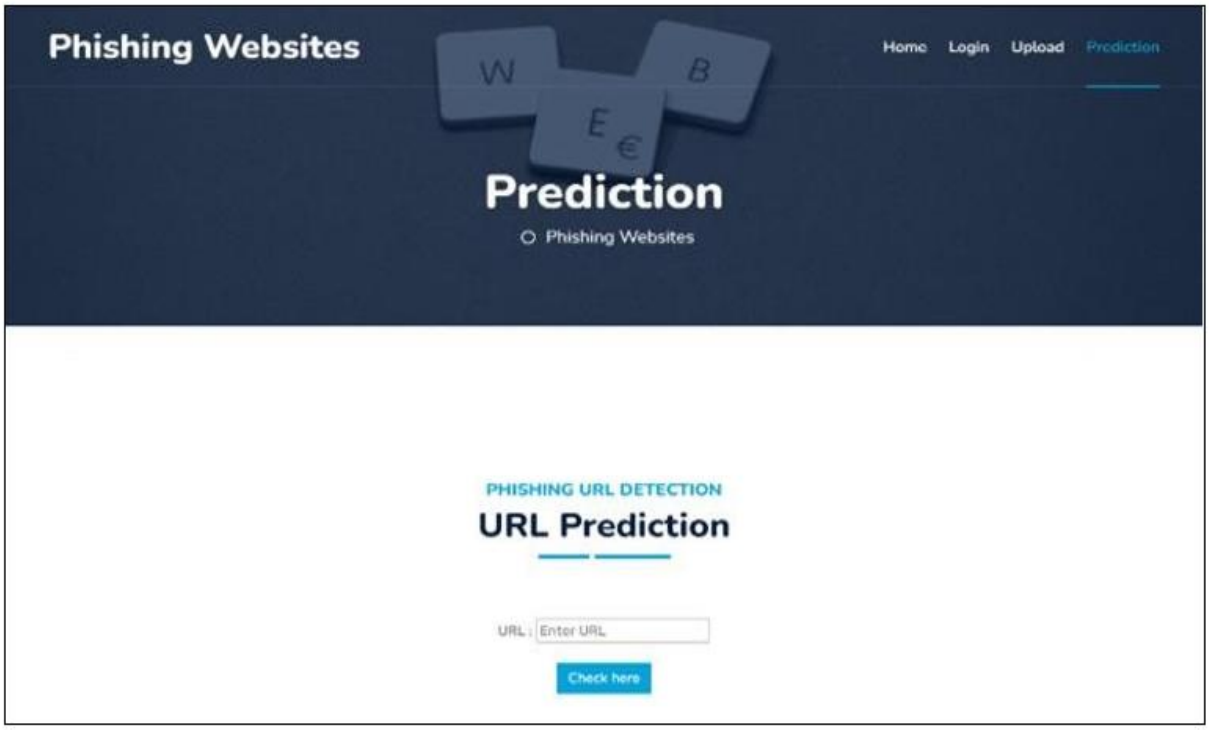


Fig:16 Input page for Detecting URL

The table titled "DETECTION OF PHISHING WEBSITES"presents a dataset. It is used for detecting phishing websites. It contains multiple features. These features are represented by columns such as having_IP_idress,double_slash_redirecting, Prefix_Suffix,having_Sub_Domain,etc.And the excel sheet consists of eleven thousand of data sets.The dataset contains values 1, -1, and 0. 1 means it indicates that it is a legitimate URL, -1 indicates it is a phishing URL, whereas 0 indicates it is neutral.

The page “Phishing Websites”, features a “Prediction” section where users can check URLs for potential phishing threats. Within this section, users are provided with a radio button labelled Phishing Websites and a text box to enter a URL. By clicking the “Check here”button, the system analyzes the URL for phishing activity.

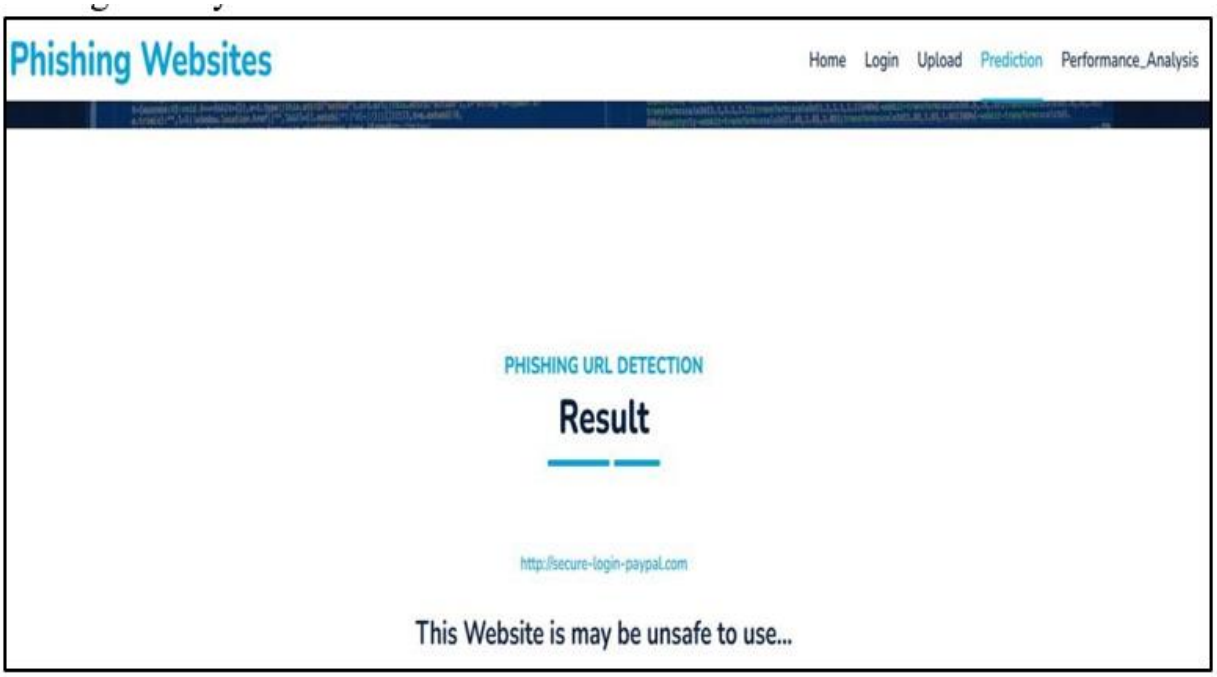


Fig:17 Result page for Phishing URL

Fig:18 Result page for Legitimate URL

Next, the system evaluates the accuracy of its predictions. The more data features we train, the higher the accuracy achieved. This accuracy is analyzed in the performance evaluation, where it is expressed as a percentage. The percentage shown below is 0.97 %.

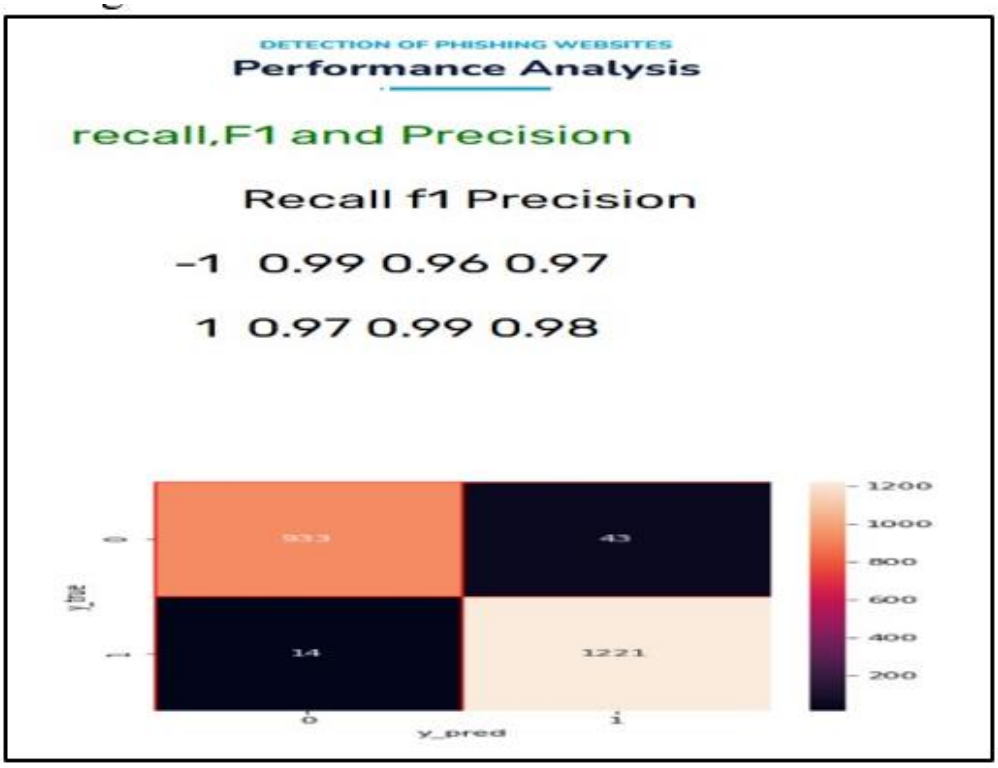


Fig:19 Confusion Matrix

The confusion matrix above illustrates the model's performance, helping us determine the accuracy of the URL classification. Compared to the previous system, the current system delivers better accuracy and is more user-friendly.

	Predicted -1	**Predicted 1**
Actual -1	1246	43
Actual 1	14	1354

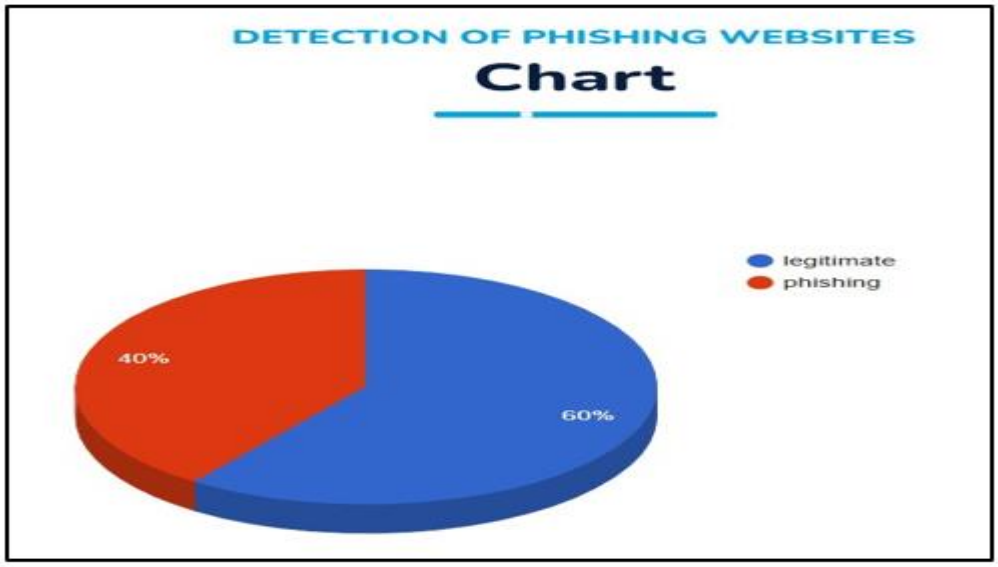


Fig:20 Phishing Detection Analysis

The image features a pie chart "DETECTION OF PHISHING WEBSITES," detected. The chart is divided into two distinct segments:

1. "Legitimate" websites – occupying 60% of the chart and coloured blue.
2. "Phishing" websites – occupy 40% of the chart and are coloured red.

5. CONCLUSION

A good anti-phishing system should be capable enough to anticipate phishing attacks within a reasonable time. Sparing the fact that an availability of a good anti-phishing gadget within a reasonable time is also required for expanding phishing site detection scope. The existing system simply identifies phishing websites through the Gradient Boosting Classifier. We obtained a 97% detection rate with the use of a Gradient Boosting Classifier with the lowest false positive rate. Though the usage of URL lexical features in isolation has been proven to lead to very high accuracy, phishers have adapted how they can render the prediction of the destination of a URL difficult through carefully tampering with the URL to prevent it from being detected. The most effective solution, thus, is to merge such features with other features like host..

Although the use of URL lexical features alone has been shown to result in high accuracy, phishers have learned how to make predicting a URL destination difficult by carefully manipulating the URL to evade detection. Therefore, combining these features with others, such as hosts, is the most effective approach .

For future improvements,we plan to implement the phishing detection system as a flexible web service that will support online usage so that it can learn new patterns of phishing attacks with ease and enhance the precision of our models using an enhanced feature extraction system.

6. REFERENCES

[1] Ammara Zamir et al. - Phishing Website Detection Using Diverse Machine Learning Algorithms (2020).

[2] Lizhen Tang & Qusay H. Mahmoud - A Survey of Machine Learning-Based Solutions for Phishing Website Detection (2021).

[3] A. Karim et al. - Phishing Detection System Through Hybrid Machine Learning Based on URL (IEEE Access, 2023).

[4] Faisal S. Alsubaei et al. -Enhancing Phishing Detection: A Novel Hybrid Deep Learning Framework for Cybercrime Forensics* (IEEE, 2024).

[5] S. Remya et al. - An Effective Detection Approach for Phishing URL Using ResMLP (IEEE, 2024).

[6] Ahmad Sahban Rafsanjani et al. - *Enhancing Malicious URL Detection: A Novel Framework Leveraging Priority Coefficient and Feature Evaluation*

[7] Valentine Adeyemi Onih - Phishing Detection Using Machine Learning: A Model Development and Integration.

[8] JianTing Yuan et al. - A Novel Approach for Malicious URL Detection Based on the Joint Model (2021).

[9] Haya T. Alhuraib et al. - Detecting Malicious URLs Using Machine Learning Techniques: Review and Research (IEEE, 2022).

[10] Malak Aljabri et al. - Detection of Phishing Websites Using Machine Learning (ICCCI, 2022).

[11] Ru Zhang et al. - Construction of Two Statistical Anomaly Features for Small-Sample APT Attack Traffic Classification(Beijing University of Posts & Telecommunications).

[12] N. Z. Harun, N. Jaffar, and P. S. J. Kassim, ‘‘Physical attributes significant in preserving the social sustainability of the traditional malay settlement,’’in *Reframing the Vernacular: Politics, Semiotics, and Representation*. Springer, 2020, pp. 225–238.

[13] D. M. Divakaran and A. Oest, ‘‘Phishing detection leveraging machine learning and deep learning: A review,’’ 2022, *arXiv:2205.07411*.

[14] A. Akanchha, ‘‘Exploring a robust machine learning classifier for detecting phishing domains using SSL certificates,’’ Fac. Compute. Sci., Dalhousie Univ., Halifax, NS, Canada, Tech. Rep. 10222/78875, 2020.

[15] H. Shahriar and S. Nimmagadda, ‘‘Network intrusion detection for TCP/IP packets with machine learning techniques,’’ in *Machine Intelligence and Big Data Analytics for Cybersecurity Applications*. Cham, Switzerland: Springer, 2020, pp. 231–247.

[16] J. Kline, E. Oakes, and P. Barford, ‘‘A URL-based analysis of WWW structure and dynamics,’’ in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2019, p. 800.

[17] A. K. Murthy and Suresha, ‘‘XML URL classification based on their semantic structure orientation for web mining applications,’’ *Proc. Compute. Sci.*, vol. 46, pp. 143–150, Jan. 2015.

[18] A. A. Ubing, S. Kamilia, A. Abdullah, N. Jhanji, and M. Supramaniam, ‘‘Phishing website detection: An improved accuracy through feature selection and ensemble learning,’’ *Int. J. Adv. Compute. Sci. Appl.*, vol. 10, no. 1, pp. 252–257, 2019.

[19] A. Aggarwal, A. Rajadesingan, and P. Kumaraguru, ‘‘PhishAri: Automatic real time phishing detection on Twitter,’’ in *Proc. eCrime Res. Summit*, Oct. 2012, pp. 1–12.

[20] S. N. Foley, D. Gollmann, and E. Snekkenes, *Computer Security—ESORICS 2017*, vol. 10492. Oslo, Norway: Springer, Sep. 2017.

[21] P. George and P. Vinod, ‘‘Composite email features for spam identification,’’ in *Cyber Security*. Singapore: Springer, 2018, pp. 281–289.

[22] H. S. Hota, A. K. Shrivas, and R. Hota, ‘‘An ensemble model for detecting phishing attacks with proposed remove-replace feature selection technique,’’ *Proc. Comput. Sci.*, vol. 132, pp. 900–907, Jan. 2018.

[23] G. Sonowal and K. S. Kuppusamy, ‘‘PhiDMA—A phishing detection model with multi-filter approach,’’ *J. King Saud Univ., Comput. Inf. Sci.*, vol. 32, no. 1, pp. 99–112, Jan. 2020.

[24] M. Zouina and B. Outtaj, ‘‘A novel lightweight URL phishing detection system using SVM and similarity index,’’ *Hum.-Centric Comput. Inf. Sci.*, vol. 7, no. 1, p. 17, Jun. 2017.