PHISHING DETECTION SYSTEM THROUGH HYBRID MACHINE LEARNING BASED ON URL

Major Project Report Submitted to

SRI PADMAVATI MAHILA VISVAVIDYALAYAM

in Partial fulfilment of the requirement for the

MASTER OF COMPUTER APPLICATIONS

IV SEMESTER

By

ANASURI SRAVANI (2023MCA16003)

Under the guidance of

Prof. T. SUDHA

Accredited by **NAAC with A$^+$** Grade   ISO 9001: 2015 Certified

DEPARTMENT OF COMPUTER SCIENCE

SRI PADMAVATI MAHILA VISVAVIDYALAYAM

(Women's University)

Tirupati-517502(A.P), Andhra Pradesh

JUNE 2025

DEPARTMENT OF COMPUTER SCIENCE

SRI PADMAVATI MAHILA VISVAVIDYALAYAM

(Women's University)

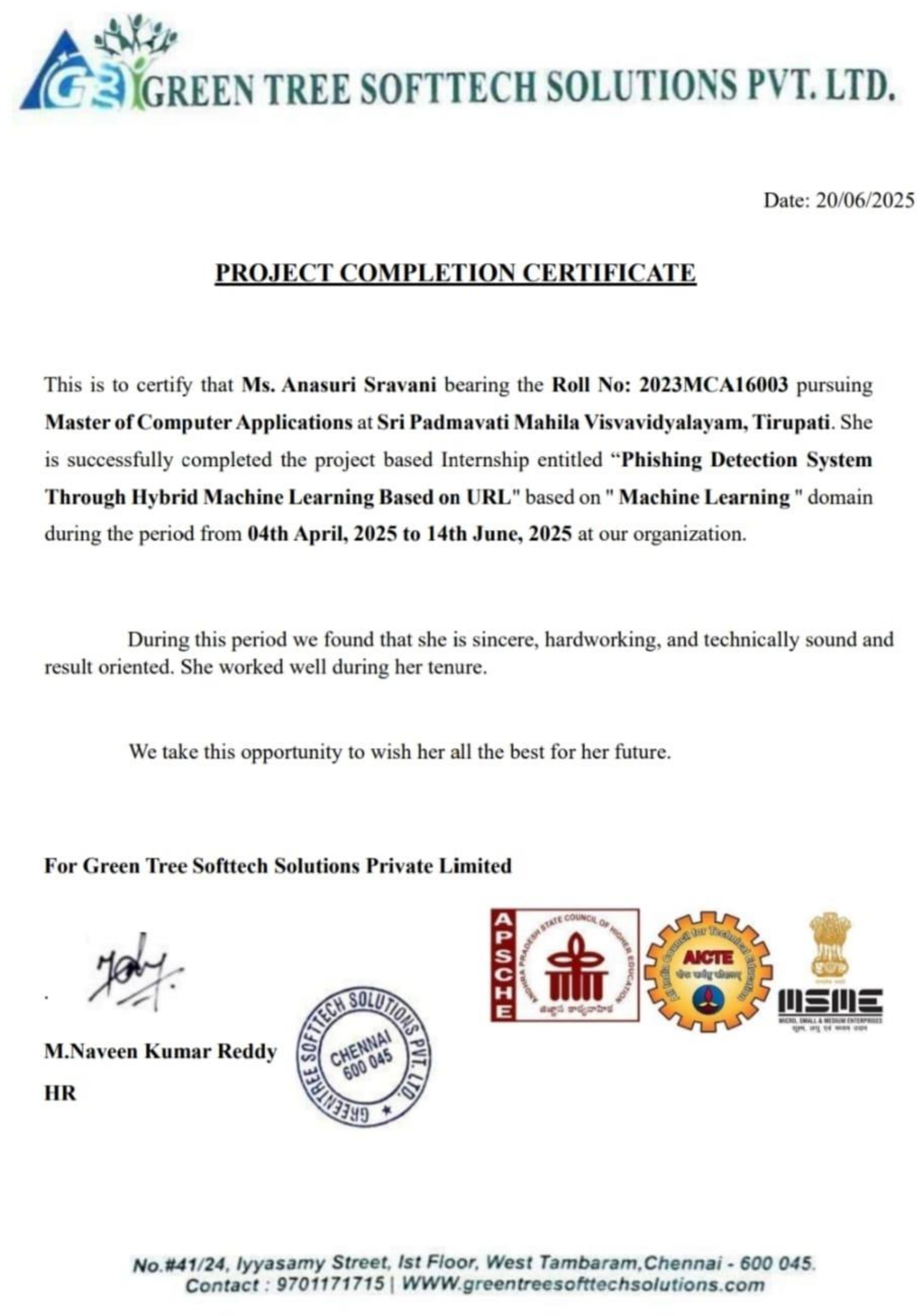
Tirupati - 517502, Andhra Pradesh, India Accredited with

A+ Grade by NAAC

CERTIFICATE

This is to certify that the project work entitled "**PHISHING DETECTION SYSTEM THROUGH HYBRID MACHINE LEARNING BASED ON URL**" is a Bonafide record of work carried out by ANASURI SRAVANI (2023MCA16003) in the **Department of Computer Science**, **Sri Padmavati Mahila Visvavidyalayam**, Tirupati in partial fulfilment of the requirements of I Semester of **MASTER OF COMPUTER APPLICATIONS**. The content of the Project Report has not been submitted to any other University / Institute for the award of any degree.

Guide ***Head of the Department***



GREEN TREE SOFTTECH SOLUTIONS PVT. LTD.

Date: 20/06/2025

PROJECT COMPLETION CERTIFICATE

This is to certify that **Ms. Anasuri Sravani** bearing the **Roll No: 2023MCA16003** pursuing **Master of Computer Applications** at **Sri Padmavati Mahila Visvavidyalayam, Tirupati**. She is successfully completed the project based Internship entitled **"Phishing Detection System Through Hybrid Machine Learning Based on URL**" based on " **Machine Learning** " domain during the period from **04th April, 2025 to 14th June, 2025** at our organization.

During this period we found that she is sincere, hardworking, and technically sound and result oriented. She worked well during her tenure.

We take this opportunity to wish her all the best for her future.

For Green Tree Softtech Solutions Private Limited

[illegible]

GREEN TREE SOFTTECH SOLUTIONS PVT. LTD. CHENNAI 600 045

M.Naveen Kumar Reddy

HR

No.#41/24, Iyyasamy Street, Ist Floor, West Tambaram,Chennai - 600 045.
Contact : 9701171715 | WWW.greentreesofttechsolutions.com

DECLARATION

I hereby declare that MCA IV Semester Major Project entitled **“PHISHING DETECTION SYSTEM THROUGH HYBRID MACHINE LEARNING BASED ON URL”** was done at the **Department of Computer Science,Sri Padmavati Mahila Visvavidyalayam**, Tirupati, in the year 2024-2025 under the guidance of **Prof. T SUDHA** in partial fulfilment of requirements of MCA IV Semester. I also declare that this project is our original contribution of the best of our knowledge and belief. I further declare that this work has not been submitted for the award of any other degree of this or any other university/Institution.

Signature of the Students

ACKNOWLEDGEMENT

I am greatly indebted to our guide **Prof. T. SUDHA** for taking keen interest on our project work and providing valuable suggestions in all the possible areas of improvement.

I express our sincere thanks to the teaching staff of the Department of Computer Science for extending support and encouragement to us in all the stages of the project work.

I gratefully acknowledge and express our gratitude to the non-teaching staff of the Computer Science Department who supported us in preparing the project report.

Signature of the Students

INDEX

ABSTRACT

This study focuses on phishing attacks, a prevalent form of cybercrime organized through the internet. It employs email distortion to engage in deceptive communication, often leading victims to fraudulent websites for data extraction.Various studies have addressed precaution, identification and knowledge of phishing attacks, yet a comprehensive solution remains elusive.

The traditional phishing detection techniques are either list based or relied on single machine learning model for prediction, this study employs several machine learning algorithms and combines them to create a hybrid model, including decision tree (DT), logistic regression (LR), random forest (RF), naive Bayes (NB), K neighbors classifier (KNN), support vector classifier (SVC), and a proposed hybrid LSD model (LR+SVC+DT) and chosen DNR model (DT+NB+RF). These models collectively aim to enhance protection against phishing attacks with high accuracy and efficiency.

Evaluation parameters such as precision, accuracy, recall, F1 score and specificity are used to illustrate the effectiveness and efficiency of the models. The performance of the model is estimated using Python programming language with the Jupyter notebook software and the dataset from Kaggle which comprises the attributes of from over 11,000+ websites in vector form, encompassing both phishing and legitimate URLs.

After evaluating our proposed model, we have crossed the accuracy of the existing model. The accuracy was 95.23 for existing model whereas our model has got an accuracy of 96.96.

1. INTRODUCTION

Phishing Detection System through Hybrid Machine Learning Based on URL

In today's highly interconnected digital environment, cyber threats have evolved rapidly, targeting not just organizations but individuals alike. One of the most prevalent and dangerous forms of cybercrime is **phishing**. Phishing attacks exploit human psychology by tricking users into divulging sensitive information such as passwords, credit card numbers, or personal identity details. These attacks are commonly carried out through deceptive emails, fraudulent messages, and fake websites that impersonate legitimate entities. The intent is to mislead users and collect private data under the guise of a trusted source.

The impact of phishing attacks can be devastating—ranging from **financial loss and identity theft to massive data breaches**. Such incidents not only compromise an individual's security but can also significantly damage the credibility and operational stability of businesses and institutions. As phishing tactics continue to evolve in sophistication and frequency, the need for **automated and intelligent phishing detection systems** has become a pressing priority.

Phishing detection is the process of identifying and preventing phishing attempts before they can cause harm. This process includes the use of various technological solutions and data-driven techniques to distinguish between malicious and legitimate communications. Traditional detection methods rely on manually created blacklists, rule-based filtering, and heuristic analysis. However, these approaches often fall short when it comes to identifying new or cleverly disguised phishing attacks.

This has prompted the integration of **Machine Learning (ML)** in phishing detection systems. ML algorithms can analyze URL structures, patterns, domain data, and other online behavior to learn and identify phishing websites without the need for manually programmed rules. As attackers adapt their methods, machine learning enables detection systems to **continuously learn and improve**, staying one step ahead of cybercriminals.

This project aims to build a **Phishing Detection System based on Hybrid Machine Learning**, which uses URL features as input for classification. The system utilizes multiple ML models and ensemble techniques to enhance detection accuracy and reduce false positives. The goal is to explore and implement different methodologies, evaluate their performance, and propose an effective hybrid solution to identify phishing attempts.

1.1 UNIVERSITY PROFILE

Sri Padmavati Mahila Visvavidyalayam (SPMVV) is a prestigious women’s university located in Tirupati, Andhra Pradesh. It was established in the year of 1983 by Sri N. T. Rama Rao, the Former Chief Minister of Andhra Pradesh. The university has earned recognition for its academic excellence, particularly in women's education, offering undergraduate, postgraduate, and research programs across various fields.

It has also been accredited by the National Assessment and Accreditation Council (NAAC) with an 'A+' grade. The university is recognized for its commitment in providing a supportive and inclusive learning environment, enabling women to excel in their careers and significantly contribute to society.

Objectives:

- **Empowerment of Women**: To provide higher education opportunities that enhance the social, economic, and intellectual empowerment of women.
- **Academic Excellence**: To offer academic programs that promote excellence, innovation, and research in diverse fields.
- **Holistic Development**: To foster the overall development of students, including intellectual, moral, and emotional growth.
- **Community Service**: To contribute to social development through outreach programs and community service initiatives.
- **Global Competence**: To equip women with the skills and knowledge necessary for success in global markets and societal advancement.

2. PROBLEM DEFINITION

2.1 AIM

The aim of this project is to develop a **robust and adaptive phishing detection system** using a **hybrid machine learning approach based on URL analysis**. The system is designed to accurately identify and classify phishing websites by leveraging multiple machine learning algorithms, enhancing detection capabilities against a wide variety of phishing tactics, including newly emerging and sophisticated attacks.

2.2 PROBLEM DEFINITION

Phishing attacks remain a persistent and growing threat to both individuals and organizations, often resulting in **financial losses, identity theft, and data breaches**. Traditional phishing detection techniques—such as blacklists and signature-based approaches—are increasingly ineffective in detecting **new, evolving, or obfuscated phishing attempts**, particularly those involving deceptive URLs.

Given the dynamic and ever-changing nature of phishing strategies, there is a pressing need for an intelligent, **real-time detection system** that can analyze URLs to identify malicious intent with high accuracy. This project addresses this problem by designing a **hybrid machine learning-based solution** capable of detecting phishing websites through URL feature extraction and classification using ensemble learning techniques.

2.1.1 EXISTING SYSTEM

Phishing detection has traditionally relied on a variety of static and rule-based techniques. The **existing system** under consideration in this study is based on the **LSD (Lightweight Statistical Detection)** model, which primarily uses handcrafted URL features and a traditional machine learning classifier to identify phishing attempts. While this system demonstrates reasonable performance in detecting known phishing websites, it faces several limitations in adapting to new and evolving attack strategies.

The LSD model leverages features extracted from the structure of URLs, such as length, number of dots, presence of special characters, and keyword patterns. It then classifies URLs using a standard supervised machine learning algorithm. This approach works well when the data distribution is consistent and when phishing websites follow known patterns. However, the system is less effective in detecting **zero-day attacks**, **obfuscated URLs**, or **highly sophisticated phishing websites** designed to evade simple statistical detection techniques.

Limitations of the Existing System

- **Limited Adaptability:** The system relies on predefined features and rules, making it less capable of adapting to new phishing tactics that differ from the training data.
- **Lower Accuracy Compared to Modern Approaches:** Although it achieves good results, the accuracy and precision levels fall short when compared to hybrid models that use advanced ensemble techniques.
- **Weaker Generalization:** It may not generalize well across different datasets or phishing scenarios, particularly when attackers use techniques such as URL shortening or domain mimicry.
- **Less Robust Against Imbalanced Data:** Traditional models may struggle with class imbalance, especially if phishing samples are underrepresented in the dataset.

2.1.2.Proposed System

The classification of phishing URLs was implemented using machine learning algorithms. Cybercrimes are growing with the growth of Internet architecture worldwide, which needs to provide a security mechanism to prevent an attacker from getting confidential content by breaching the network through fake and malicious URLs. A phishing dataset was used to perform the experiments. The dataset is in the form of data vectors that require null-value removal to remove unnecessary empty values.
Multiple machine learning algorithms, such as decision tree (DT), logistic regression (LR), naive Bayes (NB), support vector classifier (SVC), K-neighbors classifier (KNN), Random Forest (RF) and the proposed hybrid model DNR (DT+NB+RF) model with voting. To improve the prediction results, a cross-fold validation technique with grid search hyper-parameter tuning based on canopy feature selection was designed using the proposed hybrid models. Finally, predictions were made to classify the phishing URLs and evaluate their performance in terms of accuracy, precision, recall, specificity, and F1-score.

2.3.OBJECTIVES

The primary objective of this project is to design and develop a more robust and accurate phishing detection system using a hybrid machine learning approach focused on URL analysis. The system aims to address the limitations of existing methods by enhancing detection capabilities and adaptability to evolving phishing techniques.

Key Objectives:

- **To Improve Detection Accuracy:**
  Develop a phishing detection model that achieves higher accuracy compared to traditional URL-based detection methods, ensuring better classification between legitimate and phishing websites.

- **To Enhance Detection Capabilities:**
  Combine multiple machine learning algorithms to effectively detect a broader range of phishing techniques, including newly emerging and sophisticated attacks that bypass conventional filters.

- **To Explore Hybrid Machine Learning Approaches:**
  Investigate and implement hybrid models (e.g., ensemble learning techniques such as bagging, boosting, and stacking) to assess their effectiveness in phishing URL classification and to build a more adaptable detection system.

- **To Design a Real-Time URL-Based Detection System:**
  Create a lightweight, fast, and scalable phishing detection mechanism that can be deployed in real-time applications such as browsers or email filters.

- **To Evaluate the System Performance:**
  Assess the proposed model using standard evaluation metrics including accuracy, precision, recall, F1-score, and specificity to ensure its reliability and practical effectiveness.

3. SYSTEM ANALYSIS

3.1 Software Requirements Specifications

System Requirement Specification (SRS) defines the functional and non-functional requirements of the system. It serves as the foundation for system design, implementation, and testing. Below are the detailed requirements for the **Phishing Detection System through Hybrid Machine Learning Based on URL**.

Functional Requirements

These requirements define what the system must do:

1. **URL Input Module:**
   The system should allow users to input or upload a URL for phishing detection.
2. **Feature Extraction Engine:**
   Extract features from the given URL such as:
   - URL length
   - Number of dots
   - Presence of "@" symbol
   - Use of HTTPS
   - Subdomain count
   - Domain age (if available)
3. **Phishing Detection Model:**
   Use a hybrid machine learning model combining multiple algorithms (e.g., Random Forest, XGBoost, Decision Tree) to classify the URL as phishing or legitimate.
4. **Prediction Output:**
   Display the classification result (Phishing or Legitimate) along with model confidence score.
5. **Training and Testing Module (Developer-side):**
   Allow developers to train models on labeled datasets, evaluate performance using metrics such as accuracy, precision, recall, F1-score, and specificity.
6. **Visualization (optional):**
   Show evaluation metrics in graphical form (confusion matrix, bar chart comparison, etc.) for analysis and presentation.

Non-Functional Requirements

These define the quality characteristics the system should possess:

1. **Performance:**
   - The system should classify URLs in under 2 seconds.
   - High throughput is desired for batch URL processing (if implemented).
2. **Scalability:**
   - The model should be scalable to large datasets or integrated with web interfaces.
3. **Accuracy:**
   - The hybrid model must outperform the existing system in terms of accuracy and reliability.
4. **Usability:**
   - The user interface (if any) should be simple, clean, and intuitive.
5. **Security:**
   - Input URLs should be sanitized to prevent any form of injection or misuse.
6. **Maintainability:**
   - The system should be modular and easy to maintain or upgrade with new models or features.

3.2 System Requirements

3.2.1 Hardware Requirements:

- CPU : 2 x 64-bit 2.8 GHz CPUs or Higher
- RAM : 4GB 1600 MHz DDR3 RAM or Higher
- Storage : Minimum 300GB

3.2.2 Software Requirements (Tools and Frameworks):

- Operating System : Can be Windows, macOS or Linux
- Programming Language : Python
- Implementation Tool
- Datasets : Jupyter Notebook : The phishing and legitimate website URLs Dataset

3.3 Feasibility Study

3.3.1. Operational Feasibility

The system is user-friendly and can deliver real-time results, improving phishing detection accuracy. It can be easily integrated into applications or browsers.

3.3.2. Economic Feasibiliy

The development cost is minimal since all tools and datasets are freely available. No additional hardware or licensing is required, ensuring economic viability.

3.3.3. Technical Feasibility

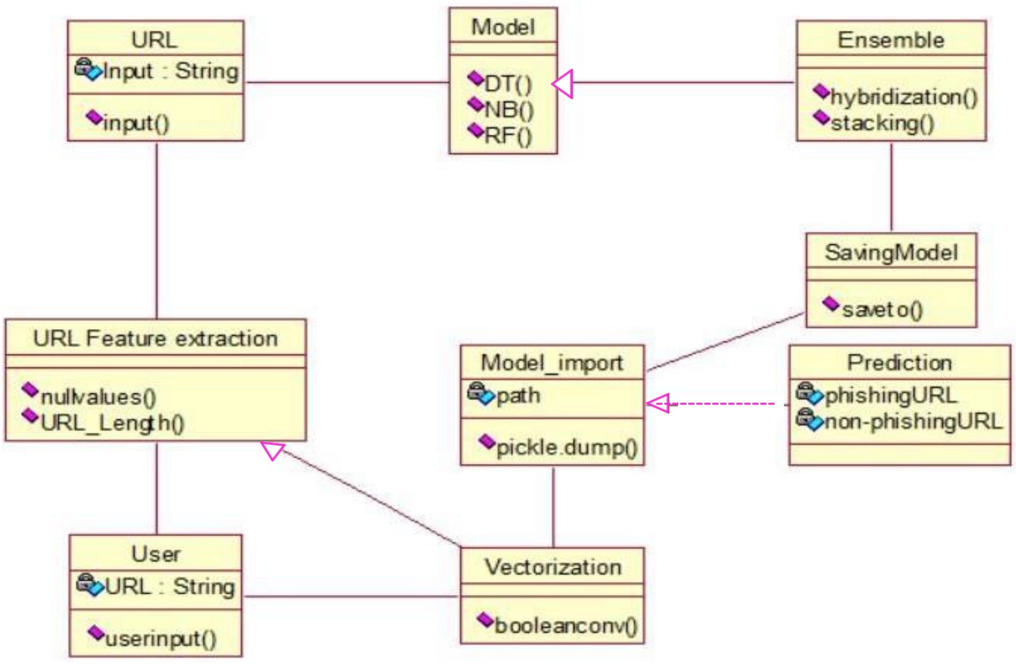
The system uses open-source tools like Python, scikit-learn, and Jupyter Notebook. It runs on standard hardware and supports URL-based feature extraction and ML model training, making it technically feasible.

3.4 Modeling Approaches

3.4.1 UML Diagrams:

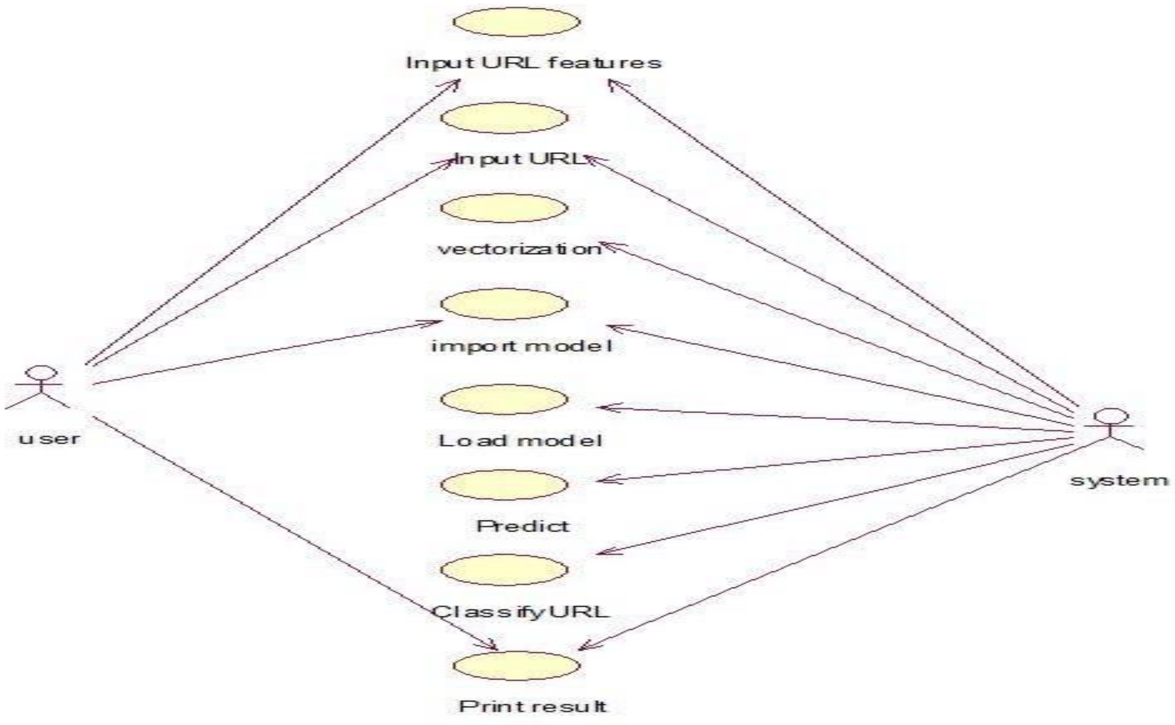
The **Unified Modeling Language (UML)** is a standardized modeling language used to visualize, specify, construct, and document the components of a software system. It helps in creating clear and structured blueprints for system development, ensuring better understanding and communication among project

3.4.1.1.Class diagram: It exhibits a set of classes, interfaces, and collaborations and their relationships. These are widely used in modelling object-oriented systems. Class diagrams which include active classes represent a static design view of the system. Figure 4.4 shows the class diagram for typical system model used in this work for developing the proposed hybrid machine learning model.



Class diagram for DNR model showing class definition and relation

3.4.1.2. Use-case diagram: A use-case diagram is used for representing a set of use-cases and actors (one type of specified class) and their relationships. These diagrams denote the static functional requirements of externally visible behaviour of a system. Figure 4.5 shows the use-case diagram for typical system model used in this work for developing the proposed model for phishing detection system.



3.4.1.3.Sequence diagram: A sequence diagram is basically an interaction diagram in which importance is given to time-oriented process. It is useful for understanding the flow of work in the system. Sequence diagrams and collaboration diagrams are isomorphic, one can take a diagram and transform it into another.

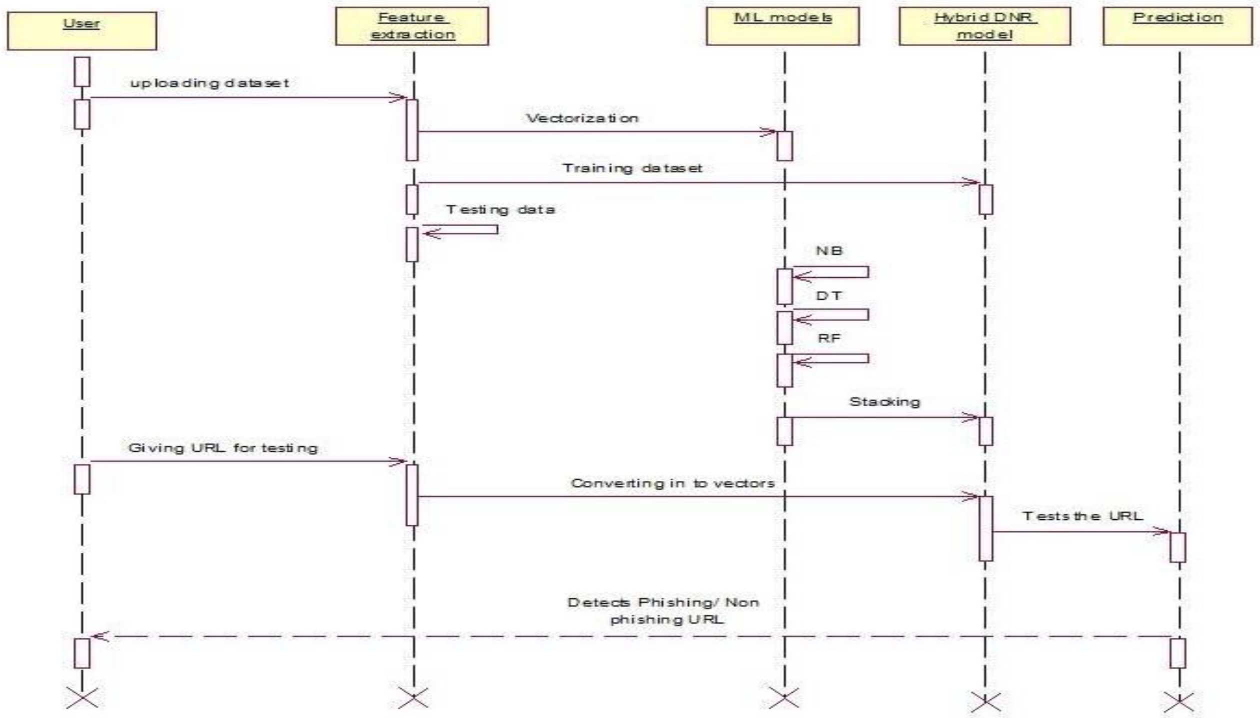
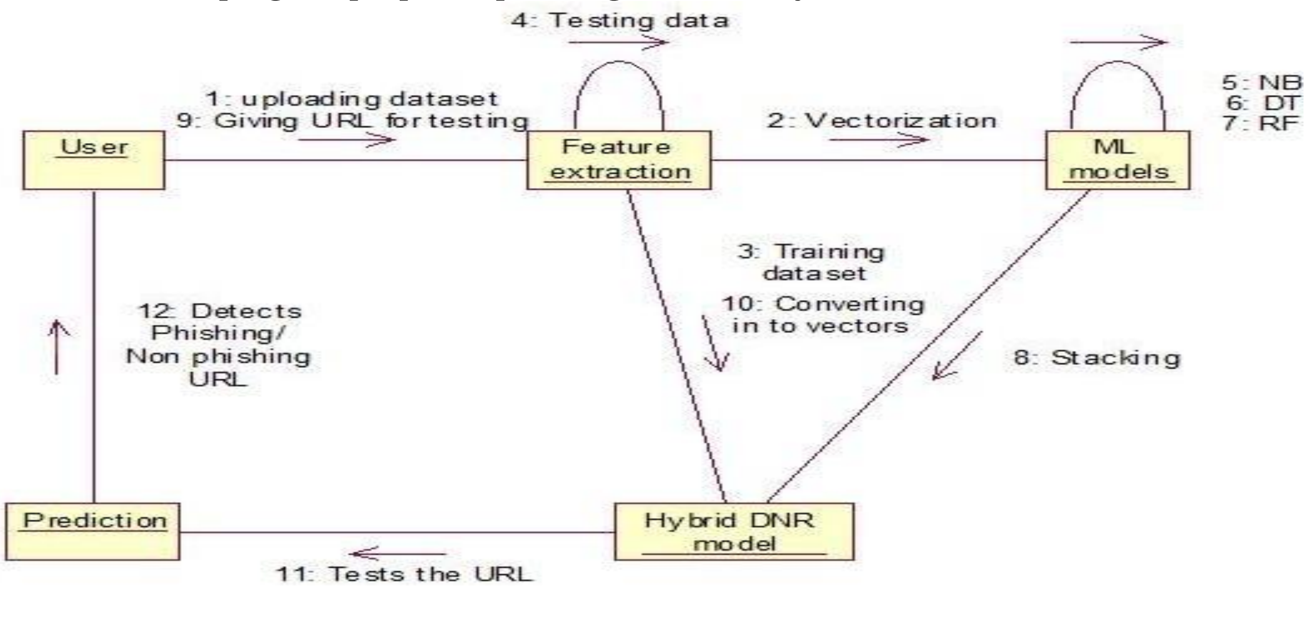


Fig 4.6: Sequence diagram showing the interaction of model

3.4.1.4.Collaboration diagram: A collaboration diagram is an interaction diagram where significance is given to the structural organization of objects which send and receive messages. Figure 4.7 shows the collaboration diagram for typical system model used in this work for developing the proposed phishing detection system.



Collaboration diagram for phishing detection system

4. SYSTEM DESIGN

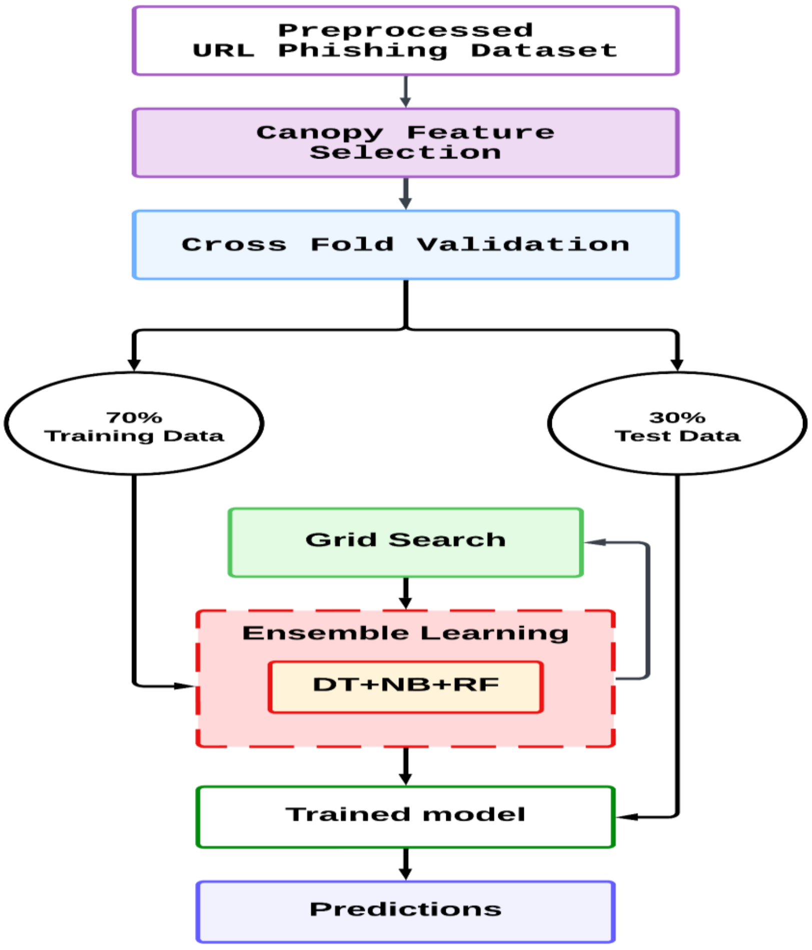
4.1 Design of Proposed System

The classification of phishing URLs was implemented using machine learning algorithms. Cybercrimes are growing with the growth of Internet architecture worldwide, which needs to provide a security mechanism to prevent an attacker from getting confidential content by breaching the network through fake and malicious URLs. A phishing dataset was used to perform the experiments. The dataset is in the form of data vectors that require null-value removal to remove unnecessary empty values.

Multiple machine learning algorithms, such as decision tree (DT), logistic regression (LR), naive Bayes (NB), support vector classifier (SVC), K-neighbors classifier (KNN), Random Forest (RF) and the proposed hybrid model DNR (DT+NB+RF) model with voting. To improve the prediction results, a cross-fold validation technique with grid search hyper-parameter tuning based on canopy feature selection was designed using the proposed hybrid models. Finally, predictions were made to classify the phishing URLs and evaluate their performance in terms of accuracy, precision, recall, specificity, and F1-score.

Architecture of the Proposed System

The proposed system Phishing Detection System Through Hybrid Machine Learning Based on URL combines different machine learning techniques to detect phishing URLs.



These principles help ensure that the Online Fee Payment System for Students is secure, scalable, and user-friendly, improving both the administrative process and student experience when managing fee payments.

Dataset and Preprocessing

The system utilizes a pre-processed dataset consisting of labelled URLs, including both phishing and legitimate examples. Preprocessing steps might involve cleaning the data, normalizing URL features, and addressing any missing or inconsistent information to ensure high quality input for the machine learning models.

Feature Selection

The system would employ canopy feature selection techniques to identify the most relevant features that contribute to phishing detection. This step helps in reducing the dimensionality of the dataset and improves model performance by focusing on the most significant attributes.

Hybrid Machine Learning Models

The system would employ a hybrid approach, combining multiple machine learning algorithms like Decision Tree, Naive Bayes and Random Forest to improve detection accuracy.

➢ Decision Tree

Decision Tree is a popular supervised machine learning algorithm used for both classification and regression tasks. It works by recursively partitioning the feature space into smaller regions, with each partition being associated with a specific class or predicting a continuous value. The decision tree builds a tree-like structure where each internal node represents a decision based on a feature attribute, and each leaf node represents the output (class label or regression value).

➢ Naive Bayes

Naive Bayes is a popular and efficient supervised learning algorithm used for classification tasks. It is based on Bayes' theorem with the "naive" assumption of feature independence, meaning that features are assumed to be conditionally independent given the class label. Despite this simplifying assumption, Naive Bayes often performs surprisingly well in practice, especially for text classification tasks.

➢ Random Forest

Random Forest is a widely-used and powerful supervised learning algorithm for classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The algorithm introduces two key concepts: bagging (Bootstrap Aggregating) and random feature selection, which enhance the robustness and accuracy of the model.

Supervised Learning

In supervised learning, the system would use labelled data to train classifiers to distinguish between phishing and legitimate URLs. Algorithms such as decision trees, naive bayes and random forest will be used in this phase

The combination of these algorithms in the supervised learning phase aims to create a comprehensive and robust phishing detection system. Decision Trees offer interpretability, Naive Bayes provides computational efficiency, and Random Forests deliver high accuracy and robustness.

Model Optimization

Model optimization is a critical step in enhancing the performance and reliability of machine learning models. In this phase, techniques such as cross-fold validation and grid search are employed to fine-tune the hyperparameters of the classifiers, ensuring that they achieve optimal results.

➢ **Cross-Fold Validation**

This technique involves dividing the dataset into several subsets, or "folds." The model is trained on some folds while tested on the remaining one, rotating through all folds. For instance, in a 10-fold cross-validation, the dataset is divided into ten parts, and the model is trained and validated ten times, each time with a different fold as the validation set.

Cross-fold validation helps in mitigating overfitting and provides a robust estimate of the model's generalizability to unseen data.

➢ **Grid Search**

This method involves an exhaustive search over a specified parameter grid to find the best hyperparameters for the model. Hyperparameters are external configurations set before training, such as the depth of a decision tree, the number of trees in a random forest, or the smoothing parameter in a Naive Bayes classifier. Grid search systematically tests different combinations of these hyperparameters to identify the most effective configuration.

Ensemble Methods

Ensemble methods are a powerful strategy in machine learning, aimed at enhancing the performance and robustness of predictive models. By combining the predictions of multiple individual models, ensemble methods can produce a more accurate and reliable final prediction than any single model alone. This approach leverages the strengths and mitigates the weaknesses of various models, leading to improved overall performance.

Types of Ensemble Methods

1. Bagging:

Bagging involves training multiple instances of a single model type on different subsets of the training data, generated by sampling with replacement. The most common example is the Random Forest algorithm, which creates an ensemble of decision trees

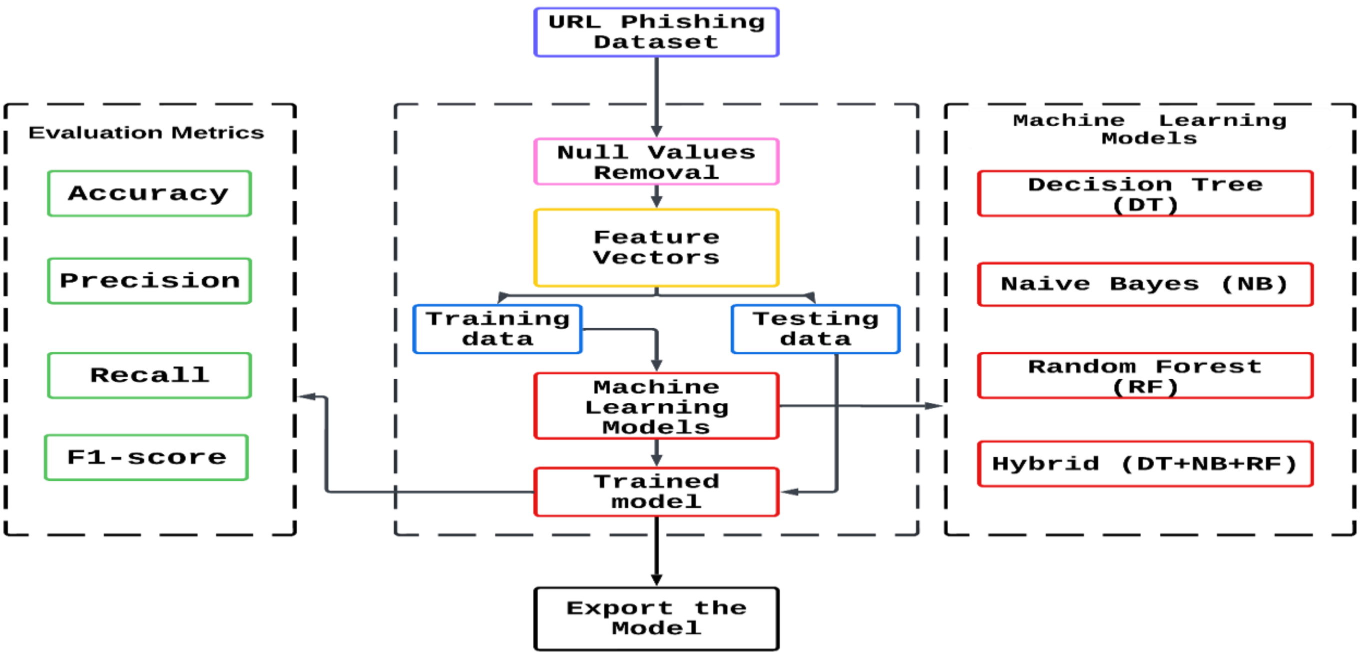
2. Boosting:

Boosting builds an ensemble by sequentially training models, where each new model focuses on correcting the errors made by the previous onesIn boosting, models are added iteratively, with each one being trained to emphasize the instances that were misclassified by the previous models. This approach reduces bias and improves the accuracy of the model.

3. Stacking:

Stacking involves training multiple base models and then using a meta-model to combine their predictions. The base models (level-0 models) are trained on the original training data, and their predictions are then used as inputs to the meta-model (level-1 model), which makes the final prediction

Design of the Proposed DNR Model



Designed Methodological Structure

Overview of the Proposed DNR Model

In our proposed hybrid model for phishing detection, we integrate Decision Tree (DT), Naive Bayes (NB), and Random Forest (RF) algorithms. This approach leverages the unique strengths of each algorithm to enhance the overall detection accuracy and robustness of the system. By combining these models, we aim to create a comprehensive system that can effectively identify phishing attempts while minimizing false positives and false negatives.

Components of the Hybrid Model

1. **Decision Tree (DT):**
   - **Function:** Decision Trees classify instances by learning simple decision rules inferred from the data features. They are particularly useful for their interpretability and ability to handle both numerical and categorical data.
   - **Strengths**: Provides clear decision paths, easy to visualize and interpret, and can capture non-linear relationships.
2. **Naive Bayes (NB):**
   - **Function:** Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between features. It calculates the probability of each class

and assigns the instance to the class with the highest probability.

- **Strengths:** Efficient and fast, particularly effective for large datasets and high-dimensional data. It performs well with text data, making it suitable for email content analysis.

3. Random Forest (RF):

- **Function:** Random Forest is an ensemble method that constructs multiple decision trees during training and outputs the class that is the mode of the classes (classification) of the individual trees. It enhances the performance of single decision trees by averaging their results.
- **Strengths:** Reduces overfitting, handles large datasets with higher dimensionality, and improves predictive accuracy through aggregation of multiple trees.

Stacking Ensemble Architecture

In a stacking ensemble architecture, the process would be as follows:

Level 0 (Base Models):

- Train a decision tree (DT) on the training data.
- Train a logistic regression (LR) model on the same data.
- Train a naive Bayes (NB) model.
- Train a support vector classifier (SVC).
- Train a K-neighbors classifier (KNN).
- Train a random forest (RF).

Level 1 (Meta-Model):

- Collect the predictions from each base model.

Use these predictions as input features to train a meta-model, such as another logistic regression or a neural network.

Final Output:

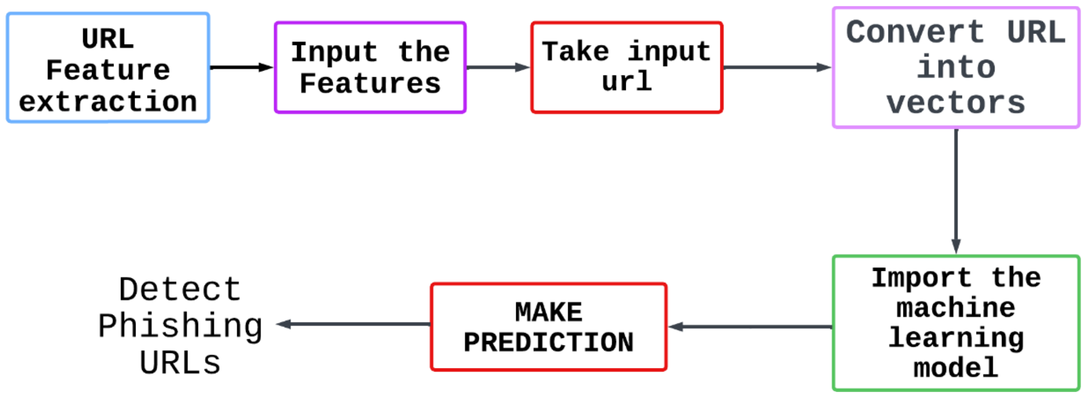
The meta-model combines the predictions from the base models to produce the final classification result. This stacked generalization method leverages the strengths of each base model while mitigating their individual weaknesses, resulting in a more robust and accurate phishing detection system.

Exporting the Hybrid Model

For the prediction of the URLs, we export the model which combines Decision Tree (DT), Naive Bayes (NB), and Random Forest (RF) algorithms using ensemble learning.

4.2.Detection of URLs using Proposed Approach

Once the trained hybrid model is saved, the next step is to use it to make predictions. This involves extracting features from the URL, transforming these features into a format compatible with the model, and using the model to predict whether the URL is phishing or legitimate.



Detection of Phishing URLs

4.2.1URL Feature Extraction:

- **Feature Extraction:**

Extract relevant features from the URL. This can include features such as the length of the URL, presence of special characters, number of subdomains, etc.

- **Vectorization:**

Convert the extracted features into a vector format that the model can use for predictions.

4.2.2.Making the Prediction

Input the URL and Use the extracted and vectorized features to make a prediction with the loaded model. The model will output a label indicating whether the URL is phishing or legitimate.

5. SYSTEM TESTING

System testing ensures that the developed phishing detection system functions correctly as a whole, including feature extraction, model integration, and result prediction. The system was tested using various real and dummy URLs to evaluate its performance, accuracy, and response to edge cases.

5.1 Testing Objectives

- To verify correct extraction of URL-based features.
- To ensure the model loads and responds without errors.
- To validate accurate classification of phishing vs. legitimate websites.
- To test the system's behavior with malformed or suspicious inputs.

Type	Description
Functional Testing	Tested whether the system performs the required tasks: feature extraction, prediction, and output display.
Manual Testing	Sample URLs were input manually to validate predictions and catch any errors.
Boundary Testing	Very long or short URLs, or URLs with excessive subdomains, were tested.
Exception Handling	Checked how the system responds to invalid or empty URLs.

Observations

- The system successfully predicted phishing and legitimate URLs based on extracted features.
- Invalid inputs were gracefully handled using exception handling logic.
- The model responded quickly (real-time prediction).
- No runtime errors were encountered when tested with multiple types of URLs.

6. IMPLEMENTATION

The implementation of the phishing detection system involves several key steps, ensuring a robust and efficient process for identifying phishing URLs. Initially, the system undergoes a thorough data preparation phase, where URLs are cleaned, and relevant features are extracted to represent their characteristics accurately. These features include URL length, the number of dots, and special characters presence, among others.

Subsequently, individual machine learning models—Decision Tree, Naive Bayes, and Random Forest are trained on this pre- processed dataset. These models are then integrated using an ensemble learning technique, such as stacking or majority voting, to form a hybrid model that combines their strengths. The hybrid model is serialized using joblib, enabling it to be easily saved and later loaded for use in various deployment environments.

Once deployed, the system extracts features from new URLs, vectorizes them as needed, and uses the loaded hybrid model to make real-time predictions, determining whether a URL is phishing or legitimate. This modular and scalable approach ensures the system's efficiency and effectiveness in providing accurate phishing detection, thereby enhancing cybersecurity measures for users and organizations.

Appendices

6.1 Source Code for Model Building

```
# importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

#importing dataset and dataset analysis
df=pd.read_csv(r"C:\Users\geeth\OneDrive\Desktop\finalproject\dataset\phishing_detection.csv")

df.head()
# result is the target attribute
df.shape
df.columns
df.info()
df.isnull().sum()
plt.figure(figsize=(7,6))
# count numbers of class records for 'Result' target attribute
sns.countplot(x='Result', data = df)
```

```
df['Result'].value_counts()

col_corr = set() # Set of all the names of deleted columns
def correlation(dataset, threshold):
  corr_matrix = dataset.corr()
  for i in range(len(corr_matrix.columns)):
      for j in range(i):
          if (abs(corr_matrix.iloc[i, j]) >= threshold) and (corr_matrix.columns[j] not in col_corr):
              colname = corr_matrix.columns[i] # getting the name of column
              col_corr.add(colname)

# remove multicollinear column with collinearity greater than 0.85
correlation(df, 0.85)
col_corr

# identifying weakly correlated features with target attribute
weak_col_corr = set()
def weakcorrelation(dataset, threshold):
  corr_matrix = dataset.corr()
  idx = 0
  for feature in corr_matrix['Result']:
    if(feature < threshold):
      weak_col_corr.add(corr_matrix.columns[idx])
    idx += 1

 # dropping features with correlation less than 0.01
weakcorrelation(df, 0.01)
print(weak_col_corr)

# gathering all columns that were identified to be deleted
del_col = col_corr.union(weak_col_corr)
del_col

# dropping columns permanantly
df.drop(del_col, axis = 1, inplace = True)

df.isnull().sum()
```

splitting dataset into train and test

```
# Load libraries
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

# input attribute and target attribute
X = df.iloc[: , :-1]
y = df.iloc[:, -1:]

y

# train test split with test size as 0.25
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
```

SVM

```
from sklearn.svm import SVC
# Building a Support Vector Machine on train data
svc_model = SVC(C= .1, gamma= 1, kernel='sigmoid', random_state=42)
```

```
svc_model.fit(X_train, y_train)

prediction = svc_model .predict(X_test)
# check the accuracy on the training set
print('Accuracy of training data: ', svc_model.score(X_train, y_train))
print('Accuracy of validation data: ',svc_model.score(X_test, y_test))

Accuracy of training data:  0.7096852008201664
Accuracy of validation data:  0.701519536903039

from sklearn.metrics import confusion_matrix, classification_report
# generating classification report
print(classification_report(y_test, prediction))

confusion_matrix(y_test, prediction)

sns.heatmap(confusion_matrix(y_test, prediction), annot = True, fmt='0.0f')
```

K-nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

neighbour = []
accuracy = []
for k in range(1, 20):
  k_near = KNeighborsClassifier(n_neighbors=k)
  k_near.fit(X,y)
  Y_pre_test = k_near.predict(X_test)
  Y_pre_train = k_near.predict(X_train)
  test_accurry = accuracy_score(Y_pre_test, y_test)
  neighbour.append(k)
  accuracy.append(test_accurry)

# plotting for n neighbour vs accuracy

plt.plot(neighbour, accuracy)
plt.title('n neighbour vs accuracy')
plt.xlabel('n neighbour')
plt.ylabel('accuracy')


k_near = KNeighborsClassifier(n_neighbors=1)
k_near.fit(X_train,y_train)


Y_pre_test = k_near.predict(X_test)
Y_pre_train = k_near.predict(X_train)

Y_pre_test = k_near.predict(X_test.values)
Y_pre_train = k_near.predict(X_train.values)

from sklearn.metrics import accuracy_score
train_accurry = accuracy_score(Y_pre_train, y_train)
test_accurry = accuracy_score(Y_pre_test, y_test)
print('Accuracy for train dataset for K-neariest : ', train_accurry)
print('Accuracy for test dataset for K-neariest : ', test_accurry)
Accuracy for train dataset for K-neariest :  0.9828729948136533
Accuracy for test dataset for K-neariest :  0.9489869753979739
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test, Y_pre_test ))

confusion_matrix(y_test, Y_pre_test )

sns.heatmap(confusion_matrix(y_test, Y_pre_test), annot = True, fmt='0.0f')
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, CategoricalNB, BernoulliNB, ComplementNB

# Bernoullis Navaive bayes classifier
nvb = BernoulliNB()
nvb.fit(X_train,y_train)

y_pre_test = nvb.predict(X_test)
y_pre_train = nvb.predict(X_train)
from sklearn.metrics import accuracy_score
train_accurry = accuracy_score(y_pre_train, y_train)
test_accurry = accuracy_score(y_pre_test, y_test)
print('Accuracy for train dataset for naive bayes  reg : ', train_accurry)
print('Accuracy for test dataset for naive bayes reg : ', test_accurry)
```

```
Accuracy for train dataset for naive bayes  reg :  0.911590881678929
Accuracy for test dataset for naive bayes reg :  0.9059334298118669
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test, y_pre_test ))

confusion_matrix(y_test, y_pre_test)

sns.heatmap(confusion_matrix(y_test, y_pre_test), annot = True, fmt='0.0f')
```

Hybrid Ensembler

Ensembler method used - Max voting: It is mainly used for classification problems. The method consists of building multiple models independently and getting their individual output called ‘vote’. The class with maximum votes is returned as output.

```
  # importing voting classifier
from sklearn.ensemble import VotingClassifier

  model_1 = SVC()
  model_2 = KNeighborsClassifier(n_neighbors=1)
  model_3 = BernoulliNB()
ensemble = VotingClassifier(estimators=[('SVM', model_1), ('KNN', model_2), ('NaiveBayes', model_3),],
voting='hard')


  ensemble.fit(X_train, y_train)
  y_pred_test = ensemble.predict(X_test)
y_pred_train = ensemble.predict(X_train)


  ensemble.fit(X_train, y_train)
  y_pred_test = ensemble.predict(X_test.values)
y_pred_train = ensemble.predict(X_train.values)
```

```
   from sklearn.metrics import accuracy_score
   train_accurry = accuracy_score(y_pred_train, y_train)
   test_accurry = accuracy_score(y_pred_test, y_test)
   print('Accuracy for train dataset for naive bayes  reg : ', train_accurry)
print('Accuracy for test dataset for naive bayes reg : ', test_accurry)

   Accuracy for train dataset for naive bayes  reg :  0.9583886141599325
   Accuracy for test dataset for naive bayes reg :  0.9439218523878437

   import pickle
   filename = 'finalized_model.sav'
pickle.dump(ensemble, open(filename, 'wb'))

print(X_test.values)

   from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix

   # Confusion matrix
   conf_matrix = confusion_matrix(y_test, y_pred_test)

   # True Positives, False Positives, True Negatives, False Negatives
   TN, FP, FN, TP = conf_matrix.ravel()

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

   # Precision
   precision = precision_score(y_test, y_pred_test)

   # Recall
   recall = recall_score(y_test, y_pred_test)

   # Specificity
   specificity = TN / (TN + FP)

   # F1 Score
   f1 = f1_score(y_test, y_pred_test)

   print("Accuracy:", accuracy)
   print("Precision:", precision)
   print("Recall:", recall)
   print("Specificity:", specificity)
print("F1 Score:", f1)

   Accuracy: 0.9439218523878437
   Precision: 0.9397821909032671
   Recall: 0.9600785340314136
   Specificity: 0.9239482200647249
   F1 Score: 0.9498219488507609
```

Source Code for Prediction of URL

```
   def having_ip_address(url):
     import re
     x = re.search('^(http|https)://\d+\.\d+\.\d+\.\d+\.*', url)
     if x:
       return 1
```

```
    else:
   return -1
```

having_ip_address('https://128.36.54.192/questions/59020008/how-to-import-functions-of-a-jupyter-notebook-into-another-jupyter-notebook-in-g')

URL LENGTH

```
  def URL_Length(url):
    if len(url) < 54:
      return -1
    elif len(url) >= 54 and len(url) <=75:
      return 0
    else:
   return 1
```

URL_Length('https://stackoverflow.com/questions/59020008/how-to-import-functions-of-a-jupyter-notebook-into-another-jupyter-notebook-in-g')

Having @ Symbol

```
  def haveAtSign(url):
    if "@" in url:
      at = 1
    else:
      at = -1
  return at
```

haveAtSign('https://stackoverflow.com/questions/59020008/how-to-import-functions-of-a-jupyter-notebook-into-another-jupyter-notebook-in-g')

prefix_Suffix

```
  def prefixSuffix(url):
      from urllib.parse import urlparse
      if '-' in urlparse(url).netloc:
          return 1
      else:
       return -1
```

prefixSuffix('https://dfhdfhdfgs.tokyo/ja-jp/account/login')

```
  def sub_domain_count(url):
    !pip install tld
    from urllib.parse import urlparse
    from tld import get_tld
    domain = urlparse(url).netloc
    domain = domain.split('.')
    top_domain = get_tld(url)
    top_domain = top_domain.split('.')
    sub_domain = set(domain) - set(top_domain)
    count = len(sub_domain)
    if(count == 1):
      return -1
    if(count == 2):
      return 0
    else:
   return 1
```

```
sub_domain_count('https://stackoverflow.com/questions/59020008/how-to-import-functions-of-a-jupyter-notebook-
into-another-jupyter-notebook-in-g')


  def sslVerify(url):
    from urllib.parse import urlparse,urlencode,urlsplit, urlunsplit
    from urllib.request import Request, urlopen, ssl, socket
    import json
    from datetime import datetime
    split_url = urlsplit(url)
    #some site without http/https in the path
    port = '443'

    hostname = split_url.netloc
    context = ssl.create_default_context()
    try:
      with socket.create_connection((hostname, port)) as sock:
        with context.wrap_socket(sock, server_hostname=hostname) as ssock:
            data = json.dumps(ssock.getpeercert())
            res = json.loads(data)
      notBefore = datetime.strptime(res["notBefore"],'%b  %d %H:%M:%S %Y %Z').date()
      notAfter = datetime.strptime(res["notAfter"],'%b  %d %H:%M:%S %Y %Z').date()
      # print(notBefore, notAfter)
      if(ssl.SSLCertVerificationError(ssock) == True):
        return 1
      elif (notAfter.year-notBefore.year)+(notAfter.month-notBefore.month)*0.1 >= 1:
        return -1
      else:
        return 0
    except:
   return 1

sslVerify("http://postdebanks.com/DIE/POST/diepost/")

Port
  import requests
  import json

  def port(domain):
      try:
          response =
  requests.get("https://api.viewdns.info/portscan/?host="+domain+"&apikey=1bf03196763a201bcc66a59bf88ed8ddf7
  a9432f&output=json")
          response.raise_for_status()  # Raise an exception for any HTTP errors

          # Check if response contains valid JSON
          try:
              myjson = response.json()
          except json.JSONDecodeError as e:
              print(f"Failed to decode JSON response: {e}")
              return -1

          pref_stat = {21: 'closed', 22: 'closed', 23: 'closed', 80: 'open', 443: 'open', 445: 'closed', 1433: 'closed', 1521:
  'closed', 3306: 'closed', 3389: 'closed'}
          flag = -1
          for port_data in myjson.get('response', {}).get('port', []):
              port_number = int(port_data.get('number', 0))
              port_status = port_data.get('status', '')
              if port_number in pref_stat and port_status != pref_stat[port_number]:
                  flag = 1
                  break  # Exit loop as soon as a port status doesn't match
          return flag
      except requests.RequestException as e:
          print(f"Request failed: {e}")
       return -1  # Return -1 in case of any request exception

  # Test the function
print(port('postdebanks.com'))
```

Request_URL

```
  def request_url(url, domain):
    # Your function implementation here

   from urllib.parse import urlparse,urlencode,urlsplit, urlunsplit
   from bs4 import BeautifulSoup
   import requests
   import re
   !pip install tldextract
   import tldextract
   Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever",
              "#content", "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]

   def is_URL_accessible(url):
     page = None
     try:
         page = requests.get(url, timeout=5)
     except:
         parsed = urlparse(url)
         url = parsed.scheme+'://'+parsed.netloc
         if not parsed.netloc.startswith('www'):
             url = parsed.scheme+'://www.'+parsed.netloc
             try:
                 page = requests.get(url, timeout=5)
             except:
                 page = None
                 pass
     if page and page.status_code == 200 and page.content not in ["b''", "b' '"]:
         return True, url, page
     else:
         return False, None, None
   state, iurl, page = is_URL_accessible(url)

   def get_domain(url):
      o = urlsplit(url)
      return o.hostname, tldextract.extract(url).domain, o.path

   if state:
       print('Yes')
       content = page.content
       hostname, domain, path = get_domain(url)
   else:
       print('No state')

   Media = {'internals':[], 'externals':[], 'null':[]}

   def external_media(Media):
     total = len(Media['internals']) + len(Media['externals'])
     externals = len(Media['externals'])
     try:
         percentile = externals / float(total) * 100
     except:
         return 0
     return percentile

   def findMedia(Media,domain, hostname ):
     Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever",
              "#content", "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]
     soup = BeautifulSoup(content, 'html.parser', from_encoding='iso-8859-1')
     for img in soup.find_all('img', src=True):
           dots = [x.start(0) for x in re.finditer('\.', img['src'])]
           if hostname in img['src'] or domain in img['src'] or len(dots) == 1 or not img['src'].startswith('http'):
               if not img['src'].startswith('http'):
                   if not img['src'].startswith('/'):
                       Media['internals'].append(hostname+'/'+img['src'])
                   elif img['src'] in Null_format:
                       Media['null'].append(img['src'])
                   else:
                       Media['internals'].append(hostname+img['src'])
           else:
               Media['externals'].append(img['src'])
     for audio in soup.find_all('audio', src=True):
      dots = [x.start(0) for x in re.finditer('\.', audio['src'])]
      if hostname in audio['src'] or domain in audio['src'] or len(dots) == 1 or not audio['src'].startswith('http'):
           if not audio['src'].startswith('http'):
               if not audio['src'].startswith('/'):
                   Media['internals'].append(hostname+'/'+audio['src'])
               elif audio['src'] in Null_format:
                   Media['null'].append(audio['src'])
               else:
                   Media['internals'].append(hostname+audio['src'])
      else:
           Media['externals'].append(audio['src'])

     for embed in soup.find_all('embed', src=True):
      dots = [x.start(0) for x in re.finditer('\.', embed['src'])]
      if hostname in embed['src'] or domain in embed['src'] or len(dots) == 1 or not embed['src'].startswith('http'):
           if not embed['src'].startswith('http'):
               if not embed['src'].startswith('/'):
                   Media['internals'].append(hostname+'/'+embed['src'])
               elif embed['src'] in Null_format:
                   Media['null'].append(embed['src'])
               else:
                   Media['internals'].append(hostname+embed['src'])
      else:
           Media['externals'].append(embed['src'])

     for i_frame in soup.find_all('iframe', src=True):
      dots = [x.start(0) for x in re.finditer('\.', i_frame['src'])]
      if hostname in i_frame['src'] or domain in i_frame['src'] or len(dots) == 1 or not i_frame['src'].startswith('http'):
           if not i_frame['src'].startswith('http'):
               if not i_frame['src'].startswith('/'):
                   Media['internals'].append(hostname+'/'+i_frame['src'])
               elif i_frame['src'] in Null_format:
                   Media['null'].append(i_frame['src'])
               else:
                Media['internals'].append(hostname+i_frame['src'])
          else:
              Media['externals'].append(i_frame['src'])

      findMedia(Media, domain, hostname)
      if external_media(Media) < 22:
        return -1
      elif external_media(Media) >= 22 and external_media(Media) < 61:
        return 0
      else:
     return 1
#request_url('http://www.budgetbots.com/server.php/Server%20update/index.php?email=USER@DOMAIN.com')

   def url_anchor(url):
       from urllib.parse import urlparse, urlsplit
       from bs4 import BeautifulSoup
       import requests
       import re
       import tldextract

       Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever", "#content",
   "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]

       def is_URL_accessible(url):
           page = None
           try:
               page = requests.get(url, timeout=5)
           except:
               parsed = urlparse(url)
               url = parsed.scheme + '://' + parsed.netloc
               if not parsed.netloc.startswith('www'):
                   url = parsed.scheme + '://www.' + parsed.netloc
                   try:
                       page = requests.get(url, timeout=5)
                   except:
                       page = None
                       pass
           if page and page.status_code == 200 and page.content not in ["b''", "b' '"]:
               return True, url, page
           else:
               return False, None, None

       state, iurl, page = is_URL_accessible(url)

       def get_domain(url):
           o = urlsplit(url)
           return o.hostname, tldextract.extract(url).domain, o.path

       domain = None  # Define a default value for domain
       if state:
           content = page.content
           hostname, domain, path = get_domain(url)

       Anchor = {'safe':[], 'unsafe':[], 'null':[]}

       def anchor(Anchor, content, domain, hostname):
           soup = BeautifulSoup(content, 'html.parser', from_encoding='iso-8859-1')
           for href in soup.find_all('a', href=True):
               dots = [x.start(0) for x in re.finditer('\.', href['href'])]
```

```
            if hostname in href['href'] or domain in href['href'] or len(dots) == 1 or not href['href'].startswith('http'):
                if "#" in href['href'] or "javascript" in href['href'].lower() or "mailto" in href['href'].lower():
                    Anchor['unsafe'].append(href['href'])
            else:
                Anchor['safe'].append(href['href'])

    if state:
        anchor(Anchor, content, domain, url)

    def safe_anchor(Anchor):
        total = len(Anchor['safe']) +  len(Anchor['unsafe'])
        unsafe = len(Anchor['unsafe'])
        try:
            percentile = unsafe / float(total) * 100
        except:
            return 0
        return percentile

    if state:
        if safe_anchor(Anchor) < 31:
            return -1
        elif safe_anchor(Anchor) >= 31 and safe_anchor(Anchor) <= 67:
            return 0
        else:
            return 1
    else:
        return -1  # Return -1 if the URL is not accessible


print(url_anchor('https://www.xiaoji.com/'))
```

Links in Tags

```
  def links_tag(url):
    from urllib.parse import urlparse,urlencode,urlsplit, urlunsplit
    from bs4 import BeautifulSoup
    import requests
    import re
    !pip install tldextract
    import tldextract
    Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever",
               "#content", "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]

    def is_URL_accessible(url):
      page = None
      try:
          page = requests.get(url, timeout=5)
      except:
          parsed = urlparse(url)
          url = parsed.scheme+'://'+parsed.netloc
          if not parsed.netloc.startswith('www'):
              url = parsed.scheme+'://www.'+parsed.netloc
              try:
                  page = requests.get(url, timeout=5)
              except:
                  page = None
                  pass
      if page and page.status_code == 200 and page.content not in ["b''", "b' '"]:
          return True, url, page
      else:
          return False, None, None

    state, iurl, page = is_URL_accessible(url)

    def get_domain(url):
        o = urlsplit(url)
        return o.hostname, tldextract.extract(url).domain, o.path

    if state:
          content = page.content
          hostname, domain, path = get_domain(url)

    Link = {'internals':[], 'externals':[], 'null':[]}

    def find_links(Link, domain, hostname):
      soup = BeautifulSoup(content, 'html.parser', from_encoding='iso-8859-1')
      for link in soup.findAll('link', href=True):
          dots = [x.start(0) for x in re.finditer('\.', link['href'])]
          if hostname in link['href'] or domain in link['href'] or len(dots) == 1 or not link['href'].startswith('http'):
              if not link['href'].startswith('http'):
                  if not link['href'].startswith('/'):
                      Link['internals'].append(hostname+'/'+link['href'])
                  elif link['href'] in Null_format:
                      Link['null'].append(link['href'])
                  else:
                      Link['internals'].append(hostname+link['href'])
          else:
              Link['externals'].append(link['href'])

      for script in soup.find_all('script', src=True):
          dots = [x.start(0) for x in re.finditer('\.', script['src'])]
          if hostname in script['src'] or domain in script['src'] or len(dots) == 1 or not script['src'].startswith('http'):
              if not script['src'].startswith('http'):
                  if not script['src'].startswith('/'):
                      Link['internals'].append(hostname+'/'+script['src'])
                  elif script['src'] in Null_format:
                      Link['null'].append(script['src'])
                  else:
                      Link['internals'].append(hostname+script['src'])
          else:
              Link['externals'].append(link['href'])

    def links_in_tags(Link):
      total = len(Link['internals']) +  len(Link['externals'])
      internals = len(Link['internals'])
      try:
          percentile = internals / float(total) * 100
      except:
          return 0
      return percentile
    #print(links_in_tags(Link))
    if links_in_tags(Link) < 17:
      return -1
    elif links_in_tags(Link) >= 17 and links_in_tags(Link) <= 81:
      return 0
    else:
    return 1
```

```
links_tag('https://www.xiaoji.com/')

SFH

  def sfh(url):
    from urllib.parse import urlparse,urlencode,urlsplit, urlunsplit
    from bs4 import BeautifulSoup
    import requests
    import re
    !pip install tldextract
    import tldextract
    Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever",
               "#content", "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]

    def is_URL_accessible(url):
      page = None
      try:
          page = requests.get(url, timeout=5)
      except:
          parsed = urlparse(url)
          url = parsed.scheme+'://'+parsed.netloc
          if not parsed.netloc.startswith('www'):
              url = parsed.scheme+'://www.'+parsed.netloc
              try:
                  page = requests.get(url, timeout=5)
              except:
                  page = None
                  pass
      if page and page.status_code == 200 and page.content not in ["b''", "b' '"]:
          return True, url, page
      else:
          return False, None, None

    state, iurl, page = is_URL_accessible(url)

    def get_domain(url):
        o = urlsplit(url)
        return o.hostname, tldextract.extract(url).domain, o.path

    if state:
          content = page.content
          hostname, domain, path = get_domain(url)

    Form = {'internals':[], 'externals':[], 'null':[]}

    def findForm(Form, domain, hostname):
      for form in soup.findAll('form', action=True):
        dots = [x.start(0) for x in re.finditer('\.', form['action'])]
        if hostname in form['action'] or domain in form['action'] or len(dots) == 1 or not form['action'].startswith('http'):
            if not form['action'].startswith('http'):
                if not form['action'].startswith('/'):
                    Form['internals'].append(hostname+'/'+form['action'])
                elif form['action'] in Null_format or form['action'] == 'about:blank':
                    Form['null'].append(form['action'])
                else:
                    Form['internals'].append(hostname+form['action'])
        else:
            Form['externals'].append(form['action'])
    def sf(hostname, Form):
      if len(Form['null'])==0:
          return 1
      elif len(Form['null'])>0:
          return 0
      else:
        return -1

  return(sf(hostname, Form))
```

Submitting to email

```
   def sub_email(url):
    from urllib.parse import urlparse,urlencode,urlsplit, urlunsplit
    from bs4 import BeautifulSoup
    import requests
    import re
    !pip install tldextract
    import tldextract
    Null_format = ["", "#", "#nothing", "#doesnotexist", "#null", "#void", "#whatever",
               "#content", "javascript::void(0)", "javascript::void(0);", "javascript::;", "javascript"]

    def is_URL_accessible(url):
      page = None
      try:
          page = requests.get(url, timeout=5)
      except:
          parsed = urlparse(url)
          url = parsed.scheme+'://'+parsed.netloc
          if not parsed.netloc.startswith('www'):
              url = parsed.scheme+'://www.'+parsed.netloc
              try:
                  page = requests.get(url, timeout=5)
              except:
                  page = None
                  pass
      if page and page.status_code == 200 and page.content not in ["b''", "b' '"]:
          return True, url, page
      else:
          return False, None, None

    state, iurl, page = is_URL_accessible(url)

    def get_domain(url):
        o = urlsplit(url)
        return o.hostname, tldextract.extract(url).domain, o.path

    if state:
          content = page.content
          hostname, domain, path = get_domain(url)

    Form = {'internals':[], 'externals':[], 'null':[]}
    def findForm(Form, domain, hostname):
      for form in soup.findAll('form', action=True):
        dots = [x.start(0) for x in re.finditer('\.', form['action'])]
        if hostname in form['action'] or domain in form['action'] or len(dots) == 1 or not form['action'].startswith('http'):
            if not form['action'].startswith('http'):
                if not form['action'].startswith('/'):
                    Form['internals'].append(hostname+'/'+form['action'])
            elif form['action'] in Null_format or form['action'] == 'about:blank':
                Form['null'].append(form['action'])
            else:
                Form['internals'].append(hostname+form['action'])
        else:
            Form['externals'].append(form['action'])

    def submitting_to_email(Form):
      # print(Form)
      for form in (Form['internals'] + Form['externals']):
          #print('inside for')
          if "mailto:" in form or "mail()" in form:
              return 1
          else:
              return -1
      return 1
```

Domain Age

```
  import json
  import requests

  def domain_age(domain):
      url = domain.split("//")[-1].split("/")[0].split('?')[0]
      show = "https://input.payapi.io/v1/api/fraud/domain/age/" + url

      retries = 3  # Number of retries
      timeout = 10  # Timeout in seconds

      for _ in range(retries):
          try:
              r = requests.get(show, timeout=timeout)
              if r.status_code == 200:
                  data = r.text
                  jsonToPython = json.loads(data)
                  result = jsonToPython['result']
                  if result / 30 >= 6:
                      return -1
                  else:
                      return 1
              else:
                  return -1
          except requests.exceptions.Timeout:
              # Retry on timeout
              continue
          except Exception as e:
              print(f"Error occurred: {e}")
              return -1

      # Return -1 if all retries fail
      return -1
print(domain_age('http://walletconnectbits.com/'))
```

Website Traffic

```
  from urllib.parse import quote
  from urllib.request import urlopen
  from bs4 import BeautifulSoup
  from urllib.error import URLError
  def web_traffic(url):
     try:
        # Filling the whitespaces in the URL if any
        url = quote(url)
        rank = BeautifulSoup(urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" + url).read(),
  "xml").find("REACH")['RANK']
        rank = int(rank)
        if rank <= 100000:
           return -1
        elif rank > 100000:
           return 0
     except (TypeError, KeyError, URLError):
        # Handle exceptions gracefully
        return 1

  # Test the function
print(web_traffic('http://postdebanks.com/DIE/POST/diepost/'))
```

page rank

```
  def page_rank(domain):
     import requests
     key = '8o0gg0804g4k0gwkk4oocws04oc0sg88gg844o4k'
     url = 'https://openpagerank.com/api/v1.0/getPageRank?domains%5B0%5D=' + str(domain)
     try:
        else:
           return -1
     except:
      return 1

print(page_rank('http://postdebanks.com/DIE/POST/diepost/'))

  import requests
  from requests.exceptions import RequestException

  def urlsCount(url):
     try:
        response = requests.get(url)
        response.raise_for_status()  # Raise an error for bad responses (4xx or 5xx)
        soup = BeautifulSoup(response.text, "html.parser")
        foundUrls = Counter([link["href"] for link in soup.find_all("a", href=lambda href: href and not
  href.startswith("#"))])
        count = len(foundUrls)
        if count == 0:
           return 1
        elif 0 < count <= 2:
           return 0
        else:
           return -1
     except RequestException as e:
        print(f"Error: {e}")
        return 1  # Return a default value or handle the error as needed

  # Test the function
print(urlsCount('https://www.hokabutypolska.pl/'))
```

statistical report

```
  def statistical_report(url, domain):
      import re
      import socket
          if url_match or ip_match:
              return 1
          else:
              return -1
      except:
       return 1

print(statistical_report('http://postdebanks.com/DIE/POST/diepost/', 'postdebanks.com'))
```

input features

```
  def getInput(url):
      print(type(url))
      from urllib.parse import urlparse
      domain = urlparse(url).netloc
      input = []
      input.append(having_ip_address(url))
      input.append(URL_Length(url))
      input.append(haveAtSign(url))
      input.append(prefixSuffix(url))
      input.append(sub_domain_count(url))
      input.append(sslVerify(url))
      input.append(port(domain))
      input.append(domain_age(url))
     #input.append(request_url(url, domain))  # Pass domain as an argument
      input.append(url_anchor(url))
      input.append(links_tag(url))
     #input.append(sfh(url))
      input.append(port(domain))
      input.append(sub_email(url))
      input.append(mouse_over(url))
      input.append(right_click(url))
      input.append(domain_age(url))
      input.append(dns_record(url))
      input.append(web_traffic(url))
      input.append(page_rank(domain))
      input.append(google_index(url))
      input.append(urlsCount(url))
      input.append(statistical_report(url, domain))

   return input
```

# Call the function with the URL

```
input = getInput('https://youtu.be/mSxNEzZLnOw?si=B-j425oSKcYWEnTZ')
[input]
```

#Importing trained Model

```
  import pickle
  filename = r"C:\Users\geeth\OneDrive\Desktop\finalized_model.sav"
loaded_model = pickle.load(open(filename, 'rb'))
```

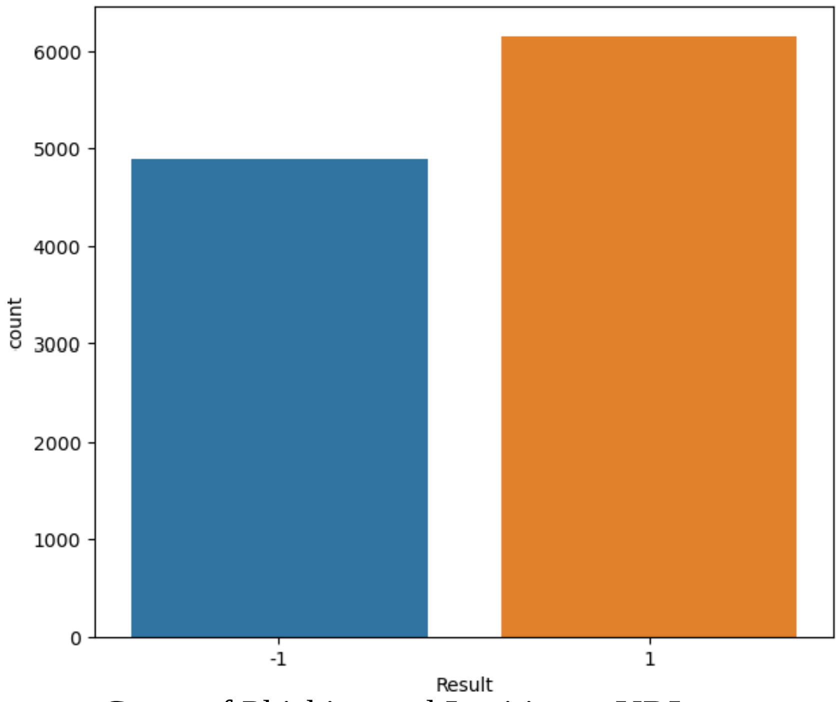
#Prediction

```
  import warnings
warnings.filterwarnings("ignore")

  result = loaded_model.predict([input])
  if result == -1:
    print("A legitimate website")
  else:
print("A Phishing website!!")
```

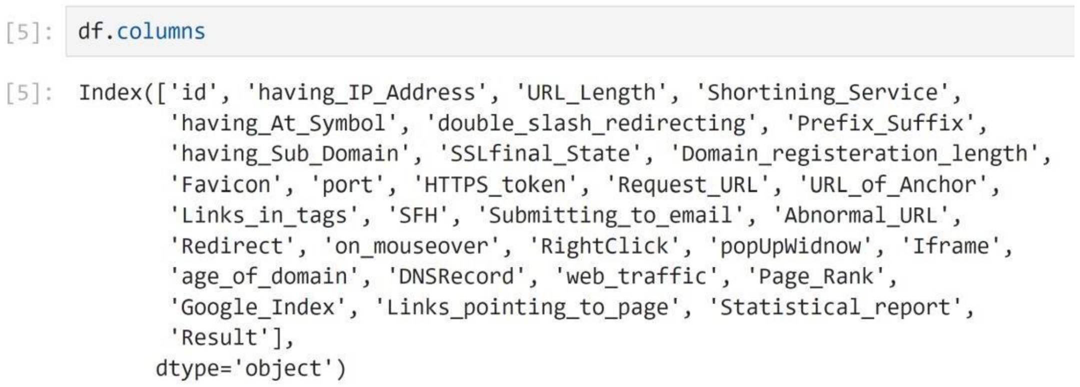
EXPERIMENTAL RESULTS AND ANALYSIS

Data Analysis



Count of Phishing and Legitimate URLs

The dataset consists of 11055 URLs, in the above figure we are checking if the target class is equally balanced or not and we can see that the target class count is almost equally balanced, The dataset has 6157 Legitimate URLs and 4898 Phishing URLs.

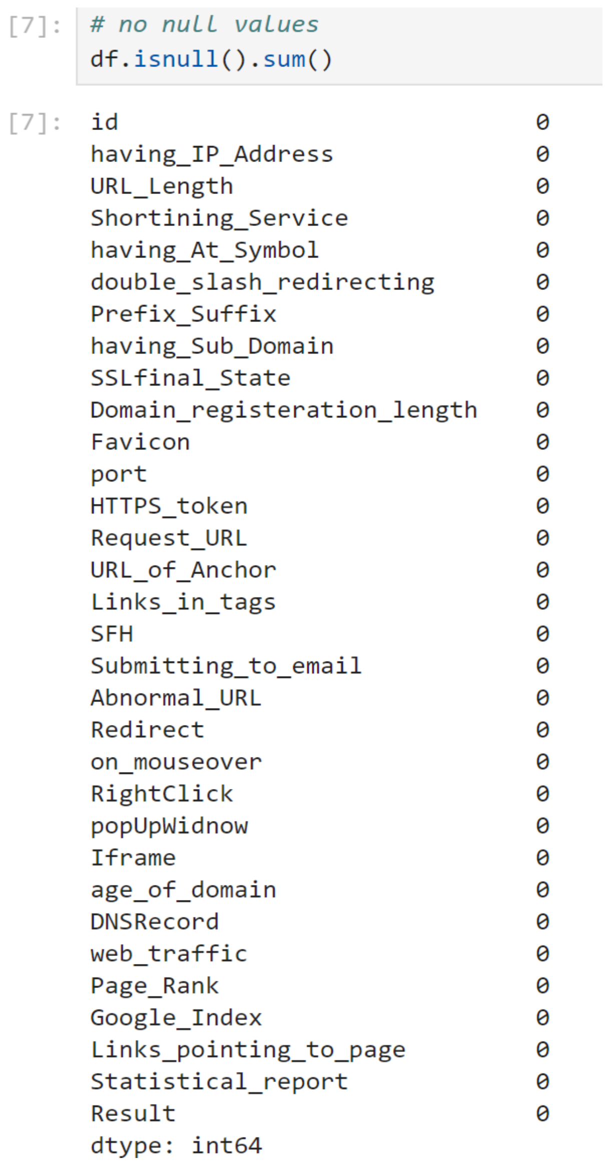


```
[5]:  df.columns

[5]:  Index(['id', 'having_IP_Address', 'URL_Length', 'Shortining_Service',
              'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix',
              'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length',
              'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor',
              'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL',
              'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe',
              'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank',
              'Google_Index', 'Links_pointing_to_page', 'Statistical_report',
              'Result'],
             dtype='object')
```

Fig 7.2: Features of URL dataset

From the above figure 7.2 we can see that the dataset has 32 Features like id, having_ip_length, URL_Length, Page_Rank, etc,. for every URL



```
[7]: # no null values
     df.isnull().sum()

[7]: id                             0
     having_IP_Address              0
     URL_Length                     0
     Shortining_Service             0
     having_At_Symbol               0
     double_slash_redirecting       0
     Prefix_Suffix                  0
     having_Sub_Domain              0
     SSLfinal_State                 0
     Domain_registeration_length    0
     Favicon                        0
     port                           0
     HTTPS_token                    0
     Request_URL                    0
     URL_of_Anchor                  0
     Links_in_tags                  0
     SFH                            0
     Submitting_to_email            0
     Abnormal_URL                   0
     Redirect                       0
     on_mouseover                   0
     RightClick                     0
     popUpWidnow                    0
     Iframe                         0
     age_of_domain                  0
     DNSRecord                      0
     web_traffic                    0
     Page_Rank                      0
     Google_Index                   0
     Links_pointing_to_page         0
     Statistical_report             0
     Result                         0
     dtype: int64
```

Checking for Null values

Accuracy:
The accuracy of a classification model is calculated as the ratio of the number of correct predictions (true positives and true negatives) to the total number of predictions (all instances in the dataset)

Mathematical formula:

$$\text{Accuracy} = \frac{\text{Number of correct predictions(TP+TN)}}{\text{Total number of predictions (TP+TN+FP+FN)}}$$

Accuracy for DNR Hybrid Model: 96.96

Precision:
Precision measures the accuracy of positive predictions made by the model, specifically focusing on the proportion of correctly predicted positive instances out of all instances predicted as positive
Mathematical Formula:

$$\textbf{Precision} = \frac{True\ positives(TP)}{True\ positives(TP) + False\ Negatives(\text{FN})}$$

➢ **Recall for DNR Hybrid Model: 97.64**

F1 score:

combines the precision and recall of a model into a single value, providing a harmonic mean that balances both measures.

Mathematical Formula:

$$\text{F1 – score} = 2 \text{ x } \frac{Precision \times Recall}{Precision + Recall}$$

- **F1 Score for DNR Hybrid Model: 97.29**

Specificity (True Negative Rate):

Specificity measures the proportion of actual negative instances that are correctly identified as negative by the model. It complements recall and is particularly relevant in scenarios where correctly identifying negative instances is important.

Mathematical Formula:

$$\text{Specificity} = \frac{True\ negatives(TN)}{True\ negatives(TN) + False\ positives(FP)}$$

- **Specificity for DNR Hybrid Model: 96.10**

Comparison of Existing vs Proposed Model

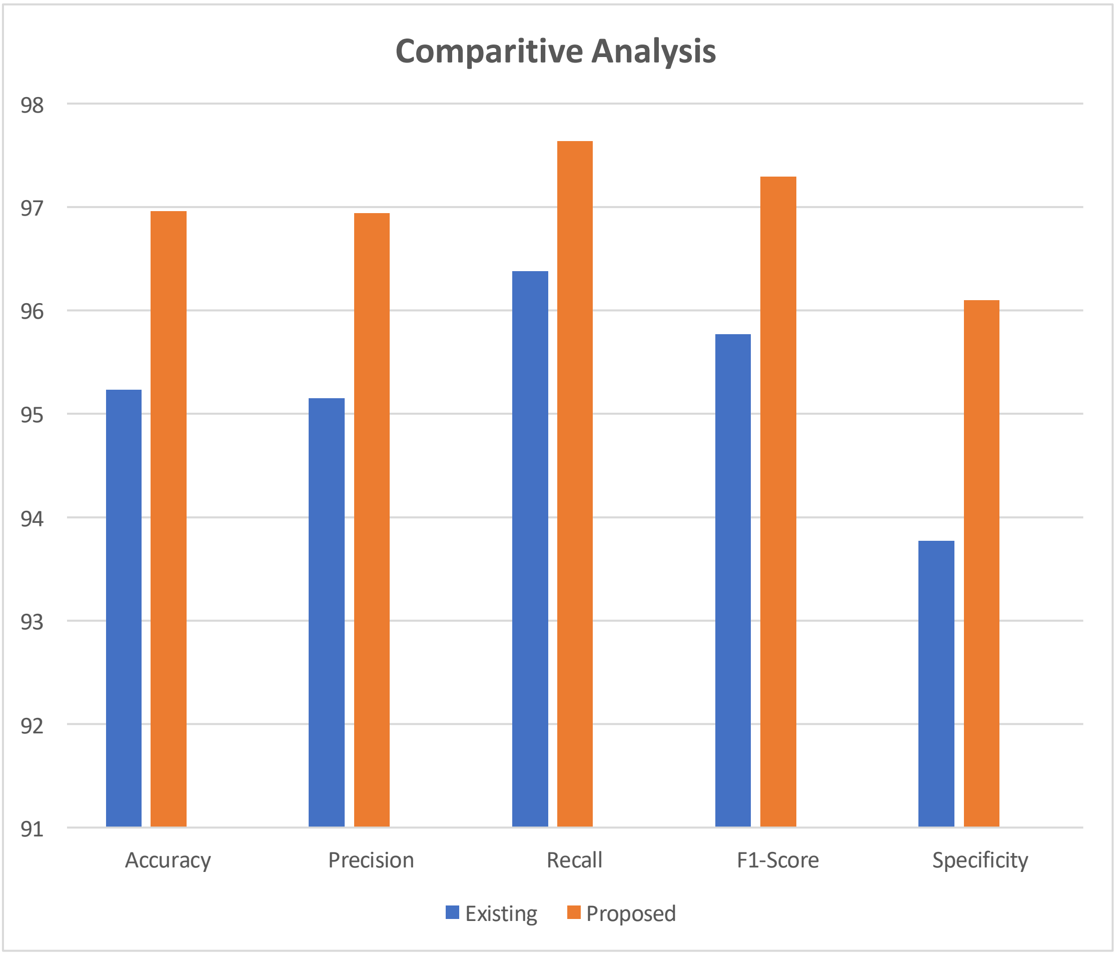
The model is performed on various metrics. Now we visualize the various evaluation parameters between existing to the proposed model

Table 7.2 Comparison between Existing and Proposed Model

	Existing Model (LSD)	Proposed Model(DNR)
Accuracy	95.23	96.96
Precision	95.15	96.94
Recall	96.38	97.64
F1-score	95.77	97.29
Specificity	93.77	96.10

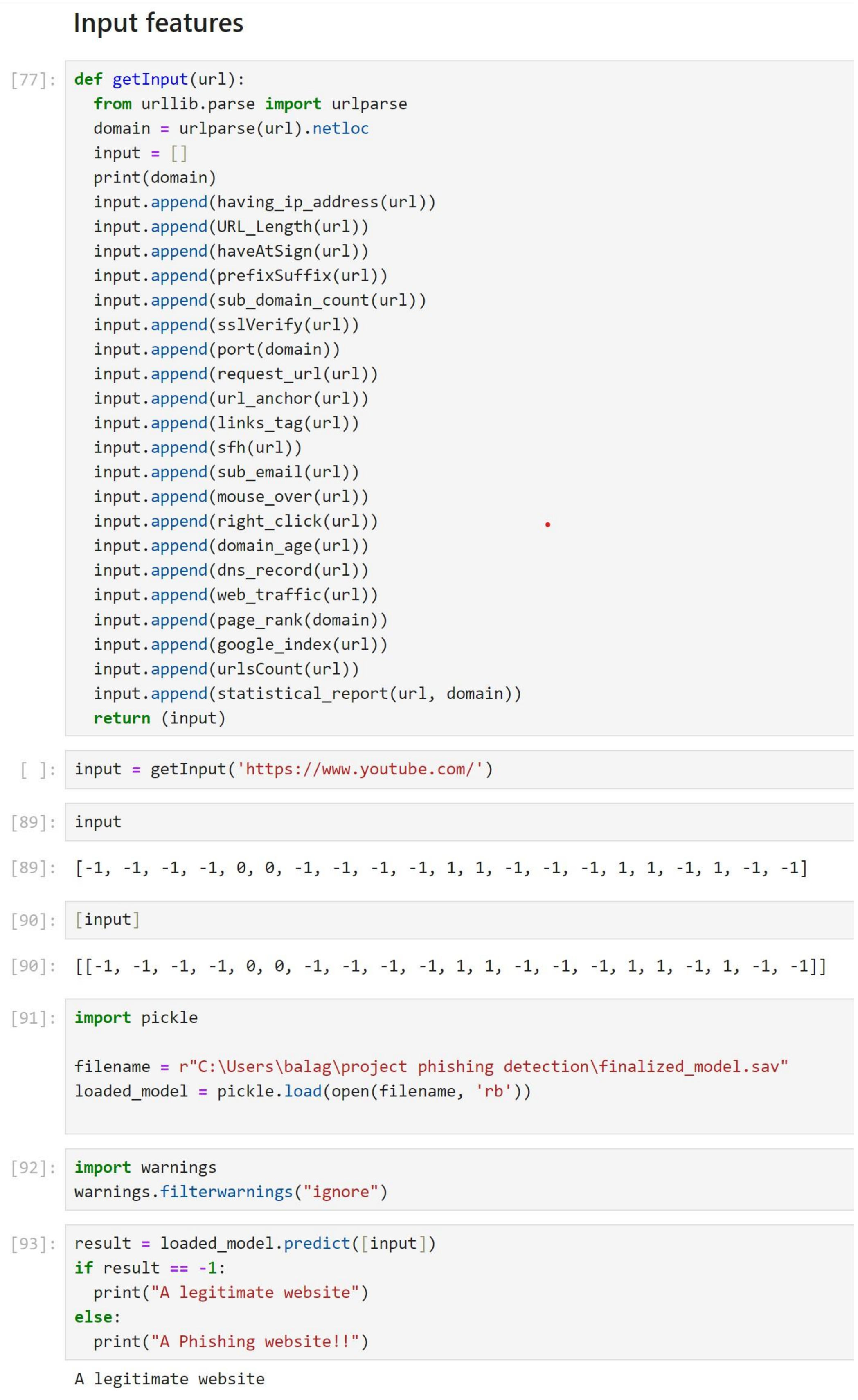
Table 7.2 shows the values of different performance metrics between existing model vs proposed model. The accuracy value of the existing model is 95.23, for proposed model the accuracy is 96.96.

Precision value for the existing model is 95.15 and for the proposed model is 96.94. Recall for the existing model is 96.38 and for the proposed model is 97.64, The F1-score of the existing model is 95.77 and for the proposed model is 97.29, The Specificity of the existing model is 93.77 and for the proposed model is 96.10.



Comparative analysis of the experimental results of the proposed approach with existing system.

URL Classification



Input features

```
[77]: def getInput(url):
        from urllib.parse import urlparse
        domain = urlparse(url).netloc
        input = []
        print(domain)
        input.append(having_ip_address(url))
        input.append(URL_Length(url))
        input.append(haveAtSign(url))
        input.append(prefixSuffix(url))
        input.append(sub_domain_count(url))
        input.append(sslVerify(url))
        input.append(port(domain))
        input.append(request_url(url))
        input.append(url_anchor(url))
        input.append(links_tag(url))
        input.append(sfh(url))
        input.append(sub_email(url))
        input.append(mouse_over(url))
        input.append(right_click(url))
        input.append(domain_age(url))
        input.append(dns_record(url))
        input.append(web_traffic(url))
        input.append(page_rank(domain))
        input.append(google_index(url))
        input.append(urlsCount(url))
        input.append(statistical_report(url, domain))
        return (input)
```

```
[ ]: input = getInput('https://www.youtube.com/')
```

```
[89]: input
```

```
[89]: [-1, -1, -1, -1, 0, 0, -1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, 1, -1, -1]
```

```
[90]: [input]
```

```
[90]: [[-1, -1, -1, -1, 0, 0, -1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, 1, -1, -1]]
```

```
[91]: import pickle

      filename = r"C:\Users\balag\project phishing detection\finalized_model.sav"
      loaded_model = pickle.load(open(filename, 'rb'))
```

```
[92]: import warnings
      warnings.filterwarnings("ignore")
```

```
[93]: result = loaded_model.predict([input])
      if result == -1:
        print("A legitimate website")
      else:
        print("A Phishing website!!")

      A legitimate website
```

7. CONCLUSION

The phishing detection system utilizing a hybrid machine learning approach based on URL analysis represents a significant advancement in cybersecurity measures. By combining multiple machine learning algorithms, specifically Decision Tree, Naive Bayes, and Random Forest, the system leverages the strengths of each to achieve superior detection accuracy and robustness. The integration of these models through ensemble learning techniques mitigates the weaknesses inherent in individual models, providing a more reliable and effective solution for identifying phishing URLs.

This system's comprehensive feature extraction process ensures that a wide range of phishing characteristics are captured, further enhancing detection capabilities. The serialization of the hybrid model allows for easy deployment and scalability, making it practical for real- time applications such as web browsing protection, email filtering, and network security monitoring. Overall, the implementation of this hybrid model significantly bolsters defences against phishing attacks, protecting sensitive information and maintaining user trust.

Future Scope

The future scope of phishing detection through hybrid machine learning models based on URL analysis holds promising potential for further enhancements and applications:

- **Enhanced Feature Extraction:**

Future work could explore more sophisticated feature extraction techniques, including the use of natural language processing (NLP) to analyze the content of URLs and associated webpages. This could improve the system's ability to detect more complex phishing attempts.

- **Incorporation of Additional Algorithms:**

Incorporating more advanced machine learning algorithms, such as deep learning models, could further improve detection accuracy. Models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) can be explored for their ability to capture intricate patterns in URL structures.

- **Integration with Multi-Modal Data:**

Combining URL-based detection with other data modalities, such as email content analysis, IP address reputation, and user behavior patterns, could provide a more comprehensive phishing detection system. This multi-modal approach would enhance the system's ability to identify phishing attempts from various angles.

- **Real-Time Detection and Response:**

Developing real-time detection and automated response mechanisms, such as immediate blocking of detected phishing URLs and alerting users, would enhance the system's practical utility. Integration with browser extensions and network security systems can facilitate real-time protection.

By pursuing these avenues, the phishing detection system can continue to evolve, offering even greater protection against phishing attacks and contributing to a safer digital ecosystem.

8. BIBILOGRAPHY

REFERENCES

[1] *Karim, M. Shahroz, K. Mustofa, S. B. Belhaouari and S. R. K. Joga, "Phishing Detection System Through Hybrid Machine Learning Based on URL," in IEEE Access, vol. 11, pp. 36805- 36822, 2023, doi: 10.1109/ACCESS.2023.3252366.*

[2] *Y. Wei and Y. Sekiya, "Sufficiency of Ensemble Machine Learning Methods for Phishing Websites Detection," in IEEE Access, vol.10, pp.124103-124113, 2022, doi:10.1109/ACCESS.2022.3224781.*

[3] *L. R. Kalabarige, R. S. Rao, A. R. Pais and L. A. Gabralla, "A Boosting-Based Hybrid Feature Selection and Multi-Layer Stacked Ensemble Learning Model to Detect Phishing Websites," in IEEE Access, vol. 11, pp. 71180-71193, 2023, doi: 10.1109/ACCESS.2023.3293649.*

[4] *Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, Banu Diri, Machine learning based phishing detection from URLs, Expert Systems with Applications, Volume 117, 2019, Pages 345-357, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2018.09.029.*

[5] *M. Sánchez-Paniagua, E. F. Fernández, E. Alegre, W. Al-Nabki and V. González-Castro, "Phishing URL Detection: A Real-Case Scenario Through Login URLs," in IEEE Access, vol. 10, pp. 42949-42960, 2022, doi: 10.1109/ACCESS.2022.3168681.*

[6] *Rao, R.S., Pais, A.R. Detection of phishing websites using an efficient feature-based machine learning framework. Neural Comput & Applic 31, 3851–3873 (2019). https://doi.org/10.1007/s00521-017-3305-0*

[7] *Jain, A.K., Gupta, B.B. A machine learning based approach for phishing detection using hyperlinks information. J Ambient Intell Human Comput 10, 2015-2028 (2019). https://doi.org/10.1007/s12652-018-0798-z*

[8] J. Novakovic and S. Markovic, "Detection of URL-based Phishing Attacks Using Neural Networks," 2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE), Ankara, Turkey, 2022, pp. 132-136, doi:10.1109/ICTACSE50438.2022.10009645.

[9] A.Aggarwal, A.Rajadesingan, and P. Kumaraguru, ''PhishAri: Automatic realtime phishing detection on Twitter,'' in Proc. eCrime Res. Summit, Oct. 2012, pp. 1–12.

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

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

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

[13] 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.