Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.11 n.53, p. 162-169. May/June., 2025.
DOI: https://doi.org/10.5935/jetia. v11i53.1652

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

APPLICATION OF MACHINE LEARNING FOR REAL-TIME PHISHING ATTACK DETECTION

Andrina Dsouza[1], Jimit Mehta[2], Prajwal Naik[3], Akshay Agrawal[4], Sanketi Raut [5]

[1,2,3] Department of Information Technology, Universal College of Engineering, University of Mumbai, Vasai, India.
[4,5] University of Mumbai, Mumbai. India.

[1]https://orcid.org/0009-0006-2743-840X, [2]https://orcid.org/0009-0009-1689-2158, [3]https://orcid.org/0009-0008-7885-3210, [4]https://orcid.org/0000-0002-8722-7181, [5]https://orcid.org/0009-0007-1511-8019

Email: andrinarosanne05@gmail.com, jimitmehta553@gmail.com, naikprajwal20@gmail.com, akshay1661@gmail.com, sanketiraut28@gmail.com

ARTICLE INFO

Article History
Received: February 20, 2025
Revised: May 20, 2025
Accepted: June 15, 2025
Published: June 30, 2025

Keywords:
Phishing,
Machine Learning,
Python,
Random Forest,
Real-time detection.

ABSTRACT

Over the years, the Internet has been exploited to carry out a range of cyber attacks, with phishing being the most prominent one. Increasingly sophisticated techniques of phishing have threatened the security of many Internet-based systems. To be able to detect suspicious websites is a potential first step in reducing the amount of phishing attacks occurring daily. This paper outlines the development and implementation of a platform to detect phishing websites. It highlights the pressing need for early detection of possible phishing attacks to prevent data theft, frauds, etc. The system uses machine learning algorithms to distinguish legitimate websites from phishing websites and generate a prediction to be used for the platform. A user interface is implemented to have two parts. The first part includes a text field for entering a URL, which the ML model processes to give a prediction that gets displayed to the user. Another module gathers URLs as they arrive from an API and scans them for potentially suspicious websites. The final ML model, a Random Forest classifier with 27 estimators, had an accuracy of 96.12% and F1 score of 95.94%. Future enhancements and research directions are also discussed for further development of the system.

I. INTRODUCTION

A phishing attack is a form of fraud in which attackers lure victims into a fake website, manipulating them to enter sensitive details so they can be stolen. It is a prominent cyber attack, responsible for huge financial losses, data theft, users losing trust over Internet services [1], [2] and damaging the reputation of well-known industries [3]. Moreover, such attacks have shown a drastic increase in recent years [4], [5].

In the modern landscape, the Internet is being used for a wide variety of tasks, including conducting transactions between a retailer and a buyer of a product, mobile banking, healthcare, entertainment, social networking and education [6]. All such domains share one aspect in common: they require the user's data in order to operate properly. Usually, web platforms or mobile applications are used to facilitate communication between different parties, and allow sharing and storing important information so they can be accessed anywhere at any time [7]. In a phishing attack, attackers seek to exploit the trust that users have to certain entities, particularly banks and government websites [8], [9].

Although most attackers employ fake emails for phishing, there are many different types of phishing, such as:

1) **Smishing**: Uses SMSs to lure victims to fake websites.
2) **Spear Phishing**: Highly targeted phishing; the target can be an individual or an organization [8].
3) **Vishing**: Steals sensitive data through voice calls.
4) **Whaling**: Targets high-profile individuals, such as celebrities or higher authorities in an organization [8].
5) **Pop-up Phishing**: Uses fake pop-ups with a seemingly urgent tone (eg. a security flaw detected in the victim's computer) to trick victims into downloading malware.
6) **Social Engineering**: Pressures the victim into revealing sensitive information.

Figure 1 depicts a typical phishing attack, which takes place as follows: (1) A fake email or SMS is sent to the victim. The attacker, posing as a genuine entity like a well-known brand or the victim's bank, sends the victim a message that has false offers or an urgent action to be taken, that involves entering sensitive credentials (eg. credit card numbers, account numbers, passwords, etc.). (2) The victim inherently trusts the bank or the brand, making them inadvertently open the email and read it. (3) The victim then opens the attached link.]

They are sent to a fake website that was carefully constructed by attackers to make it indistinguishable from the real website [10]. There, the victim enters their credentials as demanded. (4) The credentials are sent to the attackers and hence, they get stolen. (5) The attacker may use the stolen credentials to impersonate the victim and log in to another website to possibly carry out malicious activities.

These may include making a transaction of a very large amount, posting explicit or offensive messages on social media sites or even tampering with stored data in online repositories like Google Drive. The outcome of phishing attacks can range from identity theft to severe organization-level losses. The attacker can use stolen credentials of the victim to impersonate the victim and conduct fraudulent transactions using them [10]. If employee credentials, especially those of higher-level employees are stolen, the attacker can gain access to highly sensitive organizational data and cause data breaches.

The rise of evolving phishing techniques makes it extremely difficult to develop a reliable security system to work against all of them [8], [11]. Detecting suspicious websites before the victim loses their data can ensure safer browsing. Hence, there is a need to develop an automated system that can reliably distinguish between genuine websites and phishing websites [9]. Machine Learning (ML) is one of the techniques by which phishing URLs can be inspected for their authenticity.

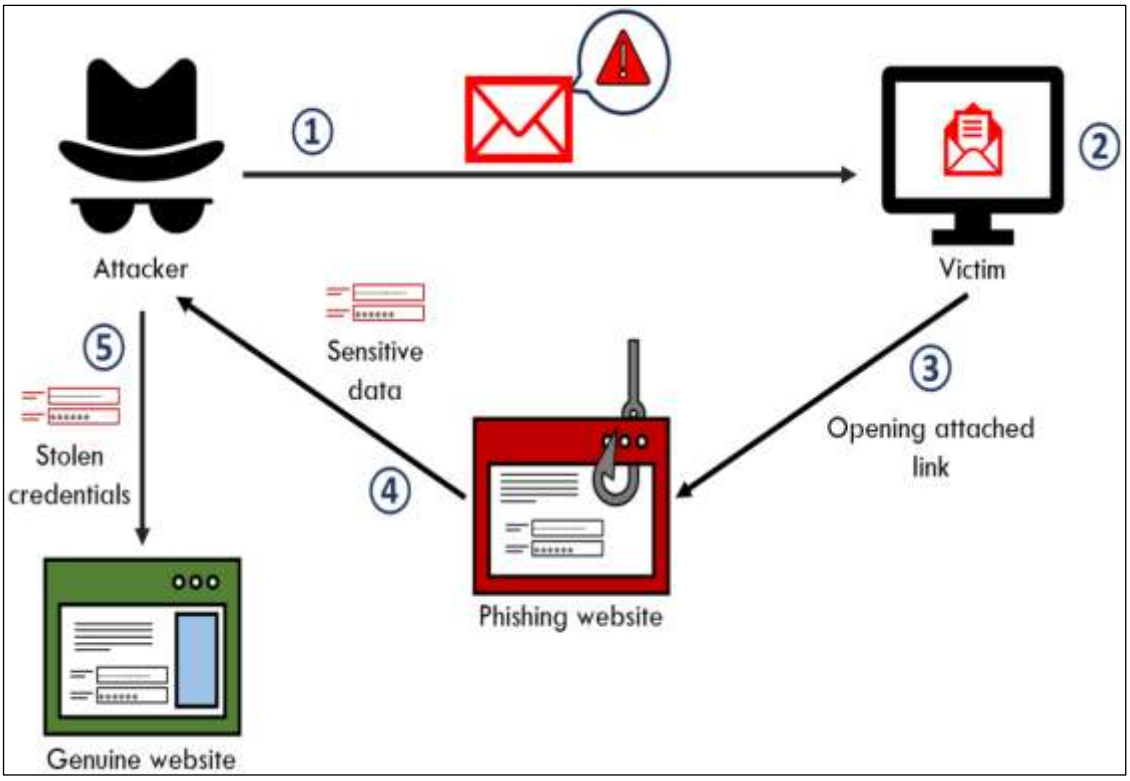

Figure 1: A Typical Phishing Attack.
Source: Authors, (2025).

I.1 OBJECTIVES

The objectives of this paper are as follows:

1. **To utilize website URL data using appropriate datasets**: Finding the best dataset that has all the necessary attributes to describe website URLs, while also allowing for appropriate ML usage.
2. **To implement an ML algorithm to distinguish phishing websites from genuine ones**: Finding the appropriate algorithm that gives optimal performance, particularly accuracy, to allow the system to be as reliable as possible.
3. **To design and develop a User Interface (UI)**: Using Python Flask to develop an application that allows users to enter a potentially suspicious URL to check whether it is a genuine or a phishing URL.
4. **To develop a platform for real-time phishing detection**: To utilize API calls from a source of website URLs to display phishing URLs as they arrive in real time.
5.

I.2 LITERATURE SURVEY

The paper [1] proposes a hybrid model for phishing detection. Its aim is to counter the problems of relying on a single model for classification, since different models have their own benefits and problems. The two best algorithms were found, then combined into a hybrid model. The top-performing models were found to be SVM (Simple Vector Machine), Gradient Boost, Random Forest (RF) and Neural Networks. The approach used is better than traditional approaches for detection; the weaknesses of one algorithm would be balanced by the strengths of another algorithm, hence stacking these together created a better-performing approach.

The study [2] explores different ML algorithms to detect spam emails. The dataset included nine features to distinguish legitimate emails from spam and phishing emails. Each of these features described emails and their content by presence of specific words (eg. 'bank', 'paypal'), the number of attached links, hidden JavaScript and other relevant features. Of the five algorithms tested, SVM and RF were the best-performing models. The method offers high accuracy, and limits the number of features to the most important ones, giving good performance.

The study [3] proposes a stacked model to detect phishing websites, combining multiple ML algorithms. The most important features were selected through feature reduction. The obtained features were then fed to a multitude of algorithms, including Random Forest, Neural Network, Bagging and SVM. The best-performing models were then stacked. The results were two stacked models: Stacking1, a combination of RF, bagging and Neural Network; and Stacking2, a combination of RF, bagging and k-Nearest Neighbour. Both gave high classification accuracy, but Stacking1 proved to be the best stacked model, giving the highest accuracy of 97.4%.

Paper [4] explores different ML algorithms to develop a phishing detection model and designs a user interface implementing the model. The algorithms tested were RF, ANN and KNN. All tested models showed extremely high accuracy. The best algorithm, the RF model, was then integrated into a Django-based web application. The interface displays a field for the user to enter a URL, which would then be processed by the ML model to return a prediction. A feedback mechanism was also implemented, allowing the user to directly state if the prediction was correct or not, and how can the site be further improved. In this way, the website facilitates real-time phishing detection and allows user feedback to further refine future predictions.

The study [5] aimed to develop an improved dataset intended for URL-based phishing detection. It focused on features within URLs to distinguish malicious websites. The dataset was developed by combining various reliable sources, including OpenPhish. 41 features were generated: 31 from previous studies and 10 novel features. These features describe the characteristics of the URL itself, the domain, and any subdomains. They also include Shannon entropies (degree of randomness of characters) of the URL and domain. They, too, can help identify phishing websites, as they tend to have higher Shannon entropies. The dataset does not include features related to the actual content of the web pages, which makes it faster to process while sacrificing some accuracy. However, it can still be relied on for research.

I.3 EXISTING SYSTEMS

Given the particularly common occurrence of email phishing [3],[4], email clients employ spam filtering to discard any suspicious emails. Email content is analyzed, especially links and attachments, to verify the authenticity of the email being received [6]. Such systems have a major drawback: they may miss targeted emails from social engineering attacks. These attacks seek a specific target rather than mass-sending to a large number of receivers.Hence, the email filter may pass the malicious emails as legitimate, especially ones that appear to be so. Moreover, attackers can adjust the language and content of their phishing email to avoid detection by many email filters.

Identifying phishing websites has multiple approaches. Some systems rely on examining the domain of the website, checking whether the secure HTTPS protocol is used instead of HTTP [6]. While some phishing websites do use the insecure HTTP protocol, attackers are increasingly switching to HTTPS to avoid detection and appear trustworthy.

The most common phishing detection approach is called 'Blacklisting' [1], [6], [8], [10]. Here, a list of suspicious websites is created and maintained. It uses up heavy amounts of computing resources, since each website must be first registered and then verified to be a phishing website. Moreover, with the constant emergence of new phishing websites, a blacklist would continue to grow and become difficult to maintain [8]. A similar approach called 'Whitelisting' lists all authentic sites, but is similarly difficult to maintain.

Anti-phishing sites like PhishTank allow users to identify suspicious websites that then get added into the growing list of phishing websites. However, as of today, new users cannot register into the site; the site was abused in 2020, leading to the removal of this feature. Existing users can continue submitting potential phishing websites.

The popularity of ML algorithms and availability of multiple datasets of phishing websites makes it possible to explore the best-performing algorithms and implement a predictive detection based on the characteristics of the URL or web page content, depending on the requirements [8], [11].

I.4 PROBLEM STATEMENT

Phishing attacks continue to evolve and become increasingly difficult to detect. Many phishing attacks occur as a result of the user having poor knowledge of phishing websites, emails, or SMSs. They do not always check for the authenticity of domains, or whether any organization actually sends them urgent messages. If a victim is unable to recognize a phishing website at the earliest, the loss of valuable data cannot be avoided.

It should be possible to scan URLs in real-time and detect phishing websites without the risk of opening suspicious links. There are numerous features of URLs that would have to be examined in order to distinguish a genuine website from phishing sites [10]. Also, the user should be alerted to any detected phishing websites.

This paper aims to utilize Machine Learning to predict whether a URL is genuine or phishing. The choice of dataset, algorithm used and its performance are important factors in accurately predicting the nature of the URL. A probability-based prediction method can help users identify the likelihood that a website is phishing or genuine. The UI consists of two simple modules: one for predicting the nature of a user-entered URL, and another for real-time fetching of URLs from an API to detect phishing websites.

II. PROPOSED SYSTEM

The system contains two web pages. The first page features an input field to accept user-entered URLs. The second page collects website URLs from an API, namely OpenPhish. In both cases, the ML model returns a probability-based prediction, which is displayed as a percentage likelihood that a URL is genuine or phishing. The system is developed using Python, Flask, HTML, CSS and Bootstrap. The main structure of the web page is developed using HTML, CSS and Bootstrap. Python code for the ML model was implemented using Jupyter Notebook. The model and front-end code were integrated using Python Flask to develop a simple web interface.

The choice of ML algorithm is made to be suitable for large data, have good accuracy, and the minimum possible false negative rate. The dataset was obtained from paper [5].

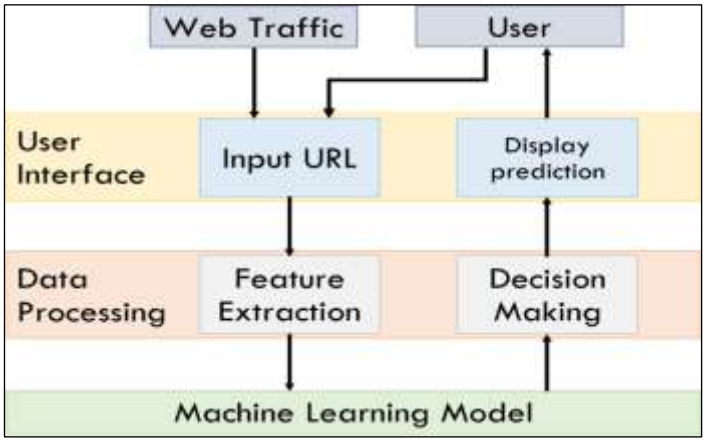

Figure 2: System Architecture.
Source: Authors, (2025).

In Figure 2, the system architecture of the proposed system is shown. The User Interface (UI) features two separate modules to obtain input URLs: in one module, the user manually enters a URL into an input field, and in the second, URLs are fetched from OpenPhish API. The URLs are then passed into the Data Processing layer, where the features of the URL are extracted.

These extracted features are passed to the ML algorithm, which returns a probability-based prediction: the likelihood that a given website link is phishing or genuine. Based on this value, the label for the website URL is decided. The predicted value lies between 0 and 1, so it is then converted into a percentage value which is then displayed to the user along with the label for the URL (either 'Genuine' or 'Phishing').

III. METHODOLOGY

III.1 DATASET CHARACTERISTICS

The dataset, obtained from [5], has 41 features, encompassing the entire URL, its domains, and subdomains. The target feature is 'Type', which labels each tuple as '1' for phishing and '0' for legitimate.

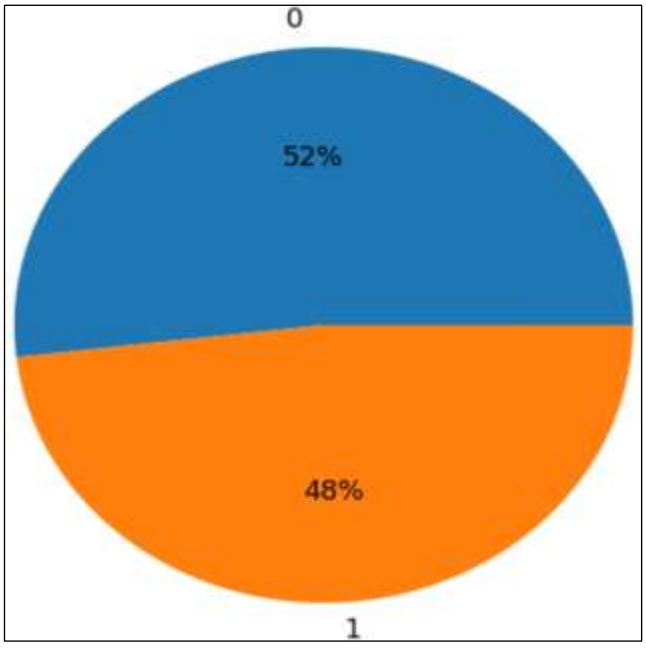

Figure 3: Data Distribution in Obtained Dataset.
Source: Authors, (2025).

The dataset has 247,950 tuples; 128,541 of them are of type 0, while the remaining 119,409 are of type 1. Figure 3 shows the distribution of the data within the dataset. It shows that the dataset is nearly balanced, with type 0 comprising 52% of the dataset, and type 1 comprising the remaining 48%. This reduces the chances of biased ML results. Hence, the dataset can be used without resorting to any sampling techniques.

The dataset contains 10 new features in total, but two particular ones are the Shannon Entropies of the URL and the domain. Shannon Entropy is defined as the randomness or uncertainty of the domain. If P_i is the probability of the i^{th} character in the domain, Shannon Entropy is given by:

$$H(i) = -\sum P_i * \log_2(P_i) \quad (1)$$

The dataset was free from null or missing values, eliminating the need for any steps to handle missing values. However, the dataset contained three attributes that had no non-zero values:
1) having_hyphen_in_subdomain,
2) average_number_of_hyphens_in_subdomain
3) having_path.

To compare the performance of each ML algorithm, two versions of the dataset were created:
1) **dataset_A**: the original dataset, with all 41 attributes.
2) **dataset_B**: the dataset with the above three attributes removed, leaving 38 attributes.

Each dataset was split in a 70:30 ratio for training and testing tuples, leaving 74,385 testing tuples.

III.2 ML ALGORITHMS TESTED

On each of the two versions of the dataset, the following ML models were tested:

1) **Decision Tree Classifier**: This is a supervised classification algorithm that creates a tree-like structure to classify tuples. Each non-leaf node is a feature that is tested, each branch is the result of testing that feature, and each leaf node is a class label (in this case, 0 or 1). Since this dataset contains purely numerical values, the nodes are split based on a threshold value for each feature, such as $>$ $\geq$, $\leq$ or $<$.

2) **AdaBoost**: This is a popular variant of the Boosting algorithm; it combines together a number of 'weak learners', which are ML models trained incrementally on the dataset [12]. Each tuple is assigned a weight, which indicates how difficult it is to classify. All weights are equal at the start of the training. When a model has been trained and tested, the weights of misclassified tuples are increased, while those of tuples that were correctly classified are decreased. The tuples with updated weights are used to train the next model. Once all the base models have been trained, they will have a weight assigned to their vote based on how well they performed. To classify an unknown URL, each model will return a vote for its class label, which is combined with its weight. The class with the highest weighted sum will be labelled final. Here, the algorithm was used with 100 estimators and a learning rate of 1.

3) **Gradient Boosting**: It is another variant of the Boosting algorithm, where each new base model must reduce the errors produced by the previous model. Since there are only two class labels in this dataset, the loss function used is log-loss, determining the deviation of the predicted label from the actual label.

4) **Artificial Neural Network**: This supervised learning technique relies on Artificial Neurons, which are information processing systems analogous to biological neurons. The three main layers in an ANN are the input layer which accepts raw inputs, hidden layers which process the input to pass to subsequent layers, and output layers which yield the final class label. Here, three hidden layers were used, each using Rectified Linear Unit (ReLU) as their activation function, and having 40, 30, and 20 neurons respectively. The output layer used the Sigmoid activation function. The number of training epochs was chosen as 100.

5) **Random Forest Classifier**: It is a popular ensemble method for ensuring high accuracy while balancing the variances introduced by individual classifiers. It is a variation of the Bagging (Bootstrap Aggregating) algorithm. The base classifiers, which are decision trees, are trained in parallel on the dataset. To implement it for this dataset, an appropriate number of decision trees must be discovered so that the maximum accuracy can be achieved. Then the ensemble can be trained and tested to predict a label based on majority voting; each base decision tree returns a class label for the URL, and the label with the highest votes is the final label. Here, dataset_A was used to repeatedly train the algorithm using different numbers of estimators from 1 to 40 trees. The training and testing accuracies were then plotted to find the best number of estimators.

After each model was trained on the datasets, their performance metrics were computed. Since the task performed is a binary classification, the tuples with phishing URLs (Type '1' in the dataset) are considered as the 'positive' tuples denoted by 'P'. The genuine URLs (Type '0' in the dataset) are considered 'negative' tuples, denoted by 'N'. There are four possibilities of a prediction of the model:

1) **True Positive (TP)**: The given URL was correctly labelled as 'Phishing' or '1'.
2) **True Negative (TN)**: The given URL was correctly labelled as 'Not Phishing' or '0'.
3) **False Positive (FP)**: A genuine URL was incorrectly labelled as 'Phishing' or '1'.
4) **False Negative (FN)**: A phishing URL was incorrectly labelled as 'Not Phishing' or '0'.

False negatives have potentially disastrous consequences, since a phishing URL may be marked as genuine. Hence, they should be minimized as much as possible. Hence, the choice of the final ML model will not just factor in accuracy, but also the false negative rate. A model of a small size is preferred to allow it to be loaded faster in the web page. In Python, when an ML model is used to give a prediction, there are two possible ways to get a prediction: a class label, achieved using the predict() method, and a probability-based prediction, achieved using the predict_proba() model. For this system, the latter is used.

III.3 DEVELOPING THE USER INTERFACE

Once the best model is obtained, it is to be used to predict a URL's nature based on its features. This means that for every URL being inputted by the user or obtained from the API, all features of the URL must be extracted, matching their order in the original dataset. Hence, a Python program was written to perform the following:

1) **Compute Shannon Entropy of Input**: The formula from Equation (1) is applied to obtain the Shannon Entropy of the input URL or extracted domain.
2) **Check for Repeated Digits**: The URL is examined for repeated digits, returning '1' if they exist, otherwise '0'.
3) **Extract all URL Features**: Each of the 40 features are extracted from the URL; the first two functions are called to return their respective features. These are formed into a Python dictionary, i.e. as key-value pairs. The values are appended into a Python list, and finally converted into a Pandas DataFrame.
4) **Fetch URLs from an API**: The OpenPhish API is used to obtain URLs in real-time to be run through the algorithm.

The interface web pages are developed using HTML, CSS, Bootstrap and JavaScript. The input-based page is developed as a simple input field, with the results of the prediction to be displayed below the field. The real-time detection page implements JavaScript and Bootstrap to render cards displaying the URLs and their nature.

The application is developed using Python Flask. First, the final ML model is loaded into the application. Then, as the user interacts with the application, it displays the input and output to the user.

III.4 SYSTEM FLOW

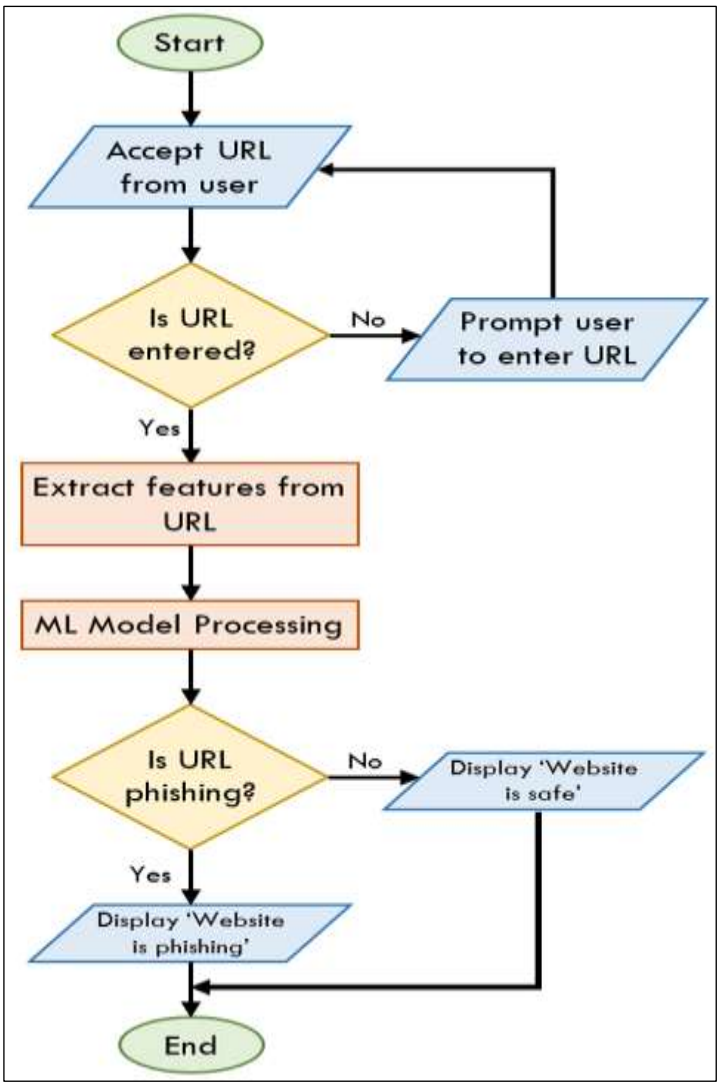

Figure 4: System Flowchart.
Source: Authors, (2025).

Figure 4 shows the system flow for the input-based phishing detection. First, a user has to enter a URL that they think is suspicious. The system checks for the field being filled, sending an alert to fill the field if left empty. The URL entered is

forwarded to the feature extraction process, which creates a complete list of URL features. These are passed to the ML model, which returns a prediction. This prediction is represented as two probabilities: the probability of the website being genuine, and the probability of the website being phishing. The prediction is based on the probability that is greater; the user is shown the prediction and the two probabilities.

IV. RESULTS AND DISCUSSIONS

IV.1 PERFORMANCE METRICS

All six ML algorithms were tested on the two created datasets to determine the one with the best classification metrics. The metrics computed for each model were as follows:

1) **Accuracy**: This is the proportion of test tuples that were labelled correctly, i.e. out of the total test tuples, how many of them were correctly labelled 0 or 1. Accuracy is given by:

$$Accuracy = \frac{TP + TN}{P + N} \quad (2)$$

2) **Recall**: This is the proportion of positive tuples that were labelled correctly, i.e. out of the total phishing URLs, how many of them were correctly labelled 1. Recall is given by:

$$Recall = \frac{TP}{P} \quad (3)$$

3) **Precision**: It is a measure of exactness, and is the proportion of tuples labelled positive by the model that are actually positive. In other words, since the ML model labels TP+FP tuples as 1, precision measures the proportion of TP tuples. Precision is given by:

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

4) **F1 Score**: This measure combines the Recall and Precision to give a single result. It is the harmonic mean of the Recall and Precision, and is given by:

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5)$$

As the classification task involved two classes, a confusion matrix was developed after each algorithm was tested. This structure compares the actual class of the tuple to the labels predicted by the model. It includes each of the four labelling types, which are TP, FP, FN and TN.

Table 1: Performance Metrics of Tested Algorithms.

Algorithm	dataset_A				dataset_B			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
AdaBoost	80.98%	81.94%	77.6%	79.71%	80.98%	81.94%	77.6%	79.71%
Gradient Boost	86.31%	88.1%	82.75%	85.34%	86.31%	88.1%	82.74%	85.34%
ANN	90.41%	93.01%	86.57%	89.68%	90.08%	91.15%	87.95%	89.52%
Decision Tree	94.55%	94.5%	94.16%	94.33%	94.57%	94.56%	94.14%	94.36%
Random Forest	96.12%	96.56%	95.33%	95.94%	96.1%	96.59%	95.27%	95.93%

Source: Authors, (2025).

Table 1 shows the final performance metrics of all the tested algorithms. Out of the five, the only algorithms to give an accuracy of 90% or higher were ANN, Decision Tree, and Random Forest with 27 estimators.

The Random Forest Classifier was initially tested with 10 estimators. The accuracies shown for dataset_A and dataset_B were 95.52% and 95.61% respectively. Given the high initial accuracy, the model was then tested using dataset_A with varying numbers of estimators. The graph of accuracies for the tested classifier is shown in Figure 5. The testing accuracy peak occurs between 25 and 30 estimators, and the position of the peak implied that the best number of estimators is 27.

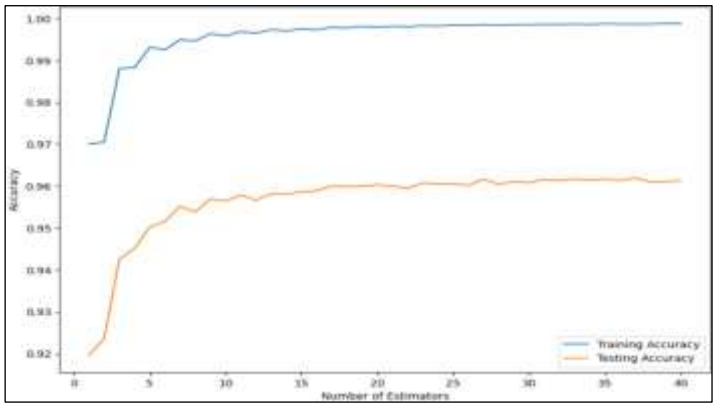

Figure 5: Accuracy Graph for Random Forest Classifier Measured from 1 to 40 Estimators
Source: Authors, (2025).

Using 27 estimators, a Random Forest Classifier was developed for each dataset. This was followed by testing the classifier and generating a confusion matrix.

Figure 6 shows the confusion matrix generated for the Random Forest for dataset_A, indicating that most tuples were correctly classified. There were 1,217 false positives and 1,672 false negatives.

Figure 7 shows the confusion matrix for dataset_B. It has a similar amount of false positives and false negatives, at 1,166 and 1,701 respectively.

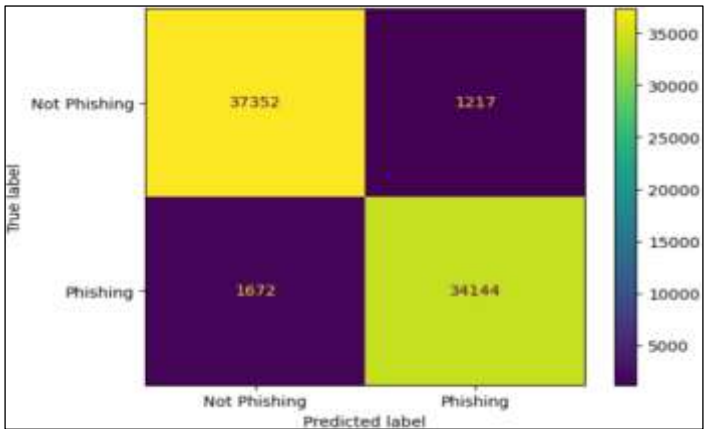

Figure 6: Confusion Matrix for Random Forest for dataset_A
Source: Authors, (2025).

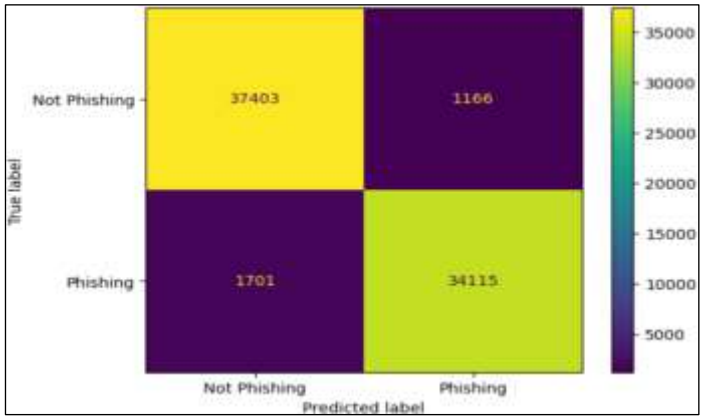

Figure 7: Confusion Matrix for Random Forest for dataset_B
Source: Authors, (2025).

For dataset_B, Random Forest showed a slightly better accuracy, precision and F1 score than for dataset_A. However, the Recall metric was slightly lower, at 95.27%, compared to 95.33% for dataset_A. This was also reflected in the confusion matrices, showing that the false negative rate was slightly lower for the original dataset. Based on the good performance and overall lower false negative rates, the final classification model chosen was Random Forest, using all 41 attributes and 27 base classifiers.

IV.2 USER INTERFACE INTERACTION

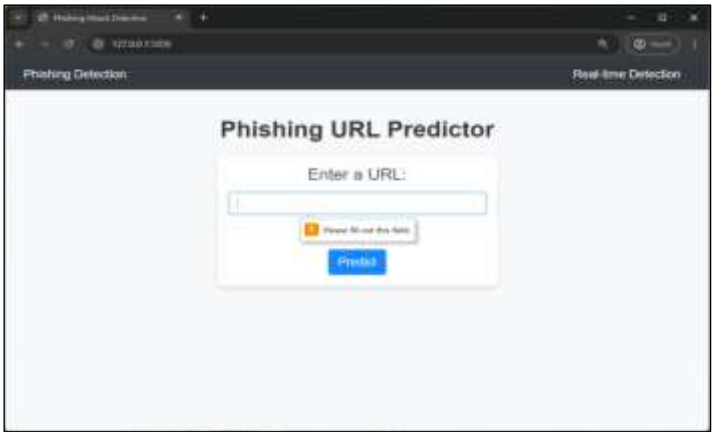

Figure 8: User Interface with a Prompt to Enter a Website URL.
Source: Authors, (2025).

Figure 8 shows the screen displayed upon opening the application, featuring a simple form-like window with an input field to accept a URL. Should the user leave the field blank and press 'Submit', they are prompted to fill out the field.

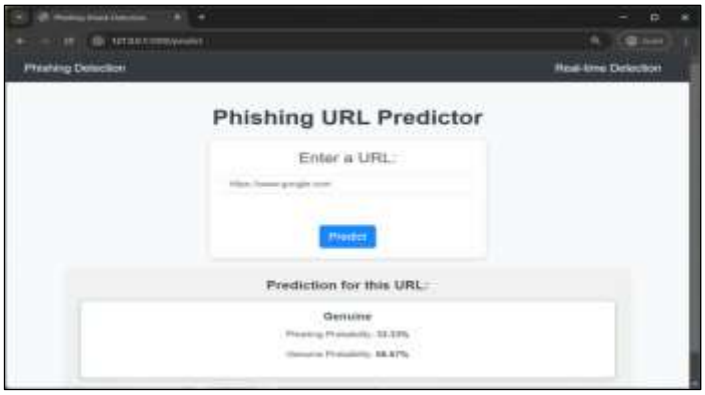

Figure 9: Prediction for a Genuine Website
Source: Authors, (2025).

Figure 9 shows the results shown when the ML model predicts a website as genuine, in this case, Google. The probability of the website being genuine is higher than its phishing probability, accordingly causing the 'Genuine' prediction to be shown.

In Figure 10, the prediction of a phishing website is shown. It displays a higher probability of the website being a phishing site, causing it to be labelled as such.

Figure 11 shows the real-time detection module, where URLs are fetched and then run through the ML model. It shows the predicted nature of the website detected and their probabilities of being phishing.

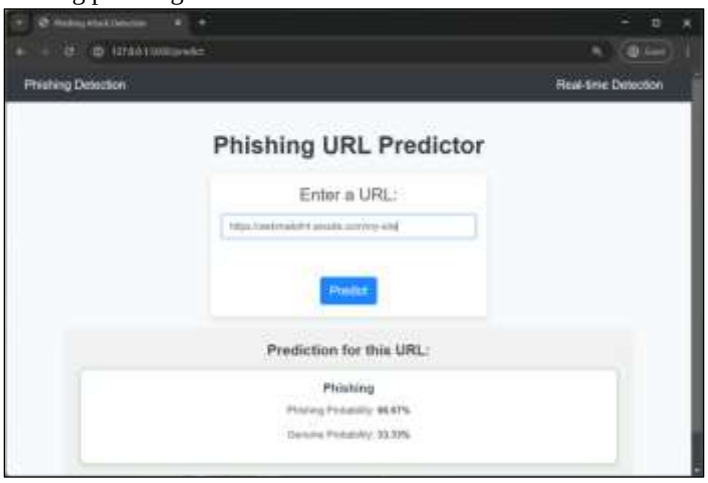

Figure 10: Prediction for a Phishing Website
Source: Authors, (2025).

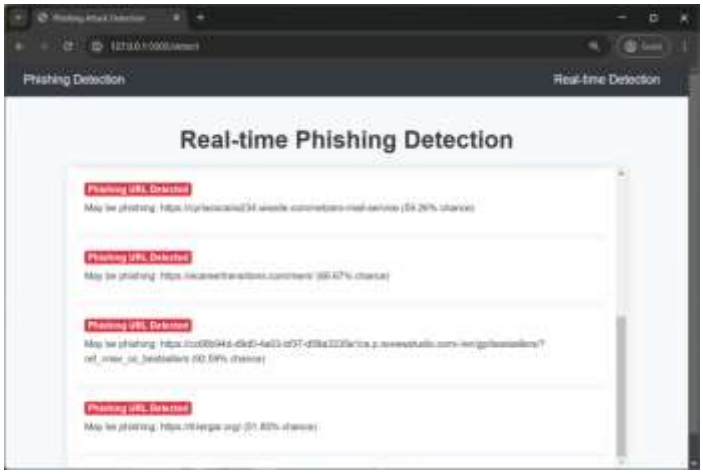

Figure 11: Real-time Phishing Detection Page
Source: Authors, (2025).

IV.3 COMPARATIVE ANALYSIS

Table 2: Comparative Analysis of Other Proposed Methodologies.

Sr. no.	Title of Paper	Proposed Methodology	Drawbacks
1	Application of machine learning for real-time phishing attack detection	Integrate the best ML algorithm into a Python Flask application, use probability-based prediction, and include a URL fetching module	May be affected by user's input, possibility of incorrect predictions
2	Machine learning approach for	Hybrid model combining well-performing ML	High False Positive rates, slower

	phishing website detection: A literature survey [1]	algorithms	in some cases
3	Phishing Detection in E-mails using Machine Learning [2]	Use only the most relevant email features and the best-performing ML algorithm	Poor adaption to evolving phishing emails
4	Phishing web site detection using diverse machine learning algorithms [3]	Find a combination of the best ML algorithms to create a stacked model	Potentially slow to execute
5	Phishing Detection Using Machine Learning: A Model Development and Integration [4]	Integrate the best-performing ML algorithm into a Django-based application	User errors may affect prediction results

Source: Authors, (2025).

The Python Flask application thus developed is capable of performing real-time detection of potentially harmful URLs. However, the ML model is not without its flaws. Given that the dataset was purely focused on the features of the URL, and that the ML model was not fully accurate, it is easy to miss potentially harmful websites and return the wrong prediction, or mislabel a known genuine website as phishing. The user may enter the same URL in many ways, and that can further lead to varying predictions. This would require mechanisms to validate user input, like checking for the presence of certain characters typical of a URL. The fetching of URLs is also limited based on the rate at which it can be performed and ethical considerations. It is possible to explore different sources to fetch URLs, but adhering to their terms of service is vital to ensure proper functioning of such applications while avoiding ethical issues.

V. CONCLUSIONS

The development of a phishing attack detection has numerous approaches, but usage of ML algorithms opens the possibility to automate such a process. A number of datasets were available online, including the one from [5] which was sufficiently large, nearly balanced and included novel attributes. The availability of updated datasets is very important in training any ML algorithm; outdated data leads to a much poorer accuracy. The final choice of the algorithm, the Random Forest Classifier, proved to have the best performance out of the five tested algorithms.

The Python Flask application integrated the algorithm, using it to predict the nature of the site by showing how likely it was to be a phishing website. Some improvements that can be made include improving input validation for the URL and experimenting with a variety of ML algorithms with different combinations of hyperparameters. The most reliable model not only provides highly accurate predictions on the dataset itself, but also on real-world data. This raises the need for development of large and updated datasets that also represent real-world websites to the highest accuracy.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Andrina Dsouza, Jimit Mehta and Prajwal Naik.
Methodology: Andrina Dsouza and Prajwal Naik.
Investigation: Andrina Dsouza, Jimit Mehta and Prajwal Naik.
Discussion of results: Andrina Dsouza, Jimit Mehta, Prajwal Naik and Akshay Agrawal.
Writing – Original Draft: Andrina Dsouza.
Writing – Review and Editing: Andrina Dsouza and Akshay Agrawal.
Resources: Andrina Dsouza.
Supervision: Akshay Agrawal.
Approval of the final text: Akshay Agrawal.

VII. REFERENCES

[1] R. R. Patil, G. Kaur, H. Jain, A. Tiwari, S. Joshi, K. Rao and A. Sharma."Machine learning approach for phishing website detection : A literature survey", Journal of Discrete Mathematical Sciences & Cryptography, Vol. 25 (2022), No. 3, pp. 817–827, April 2022, DOI: 10.1080/09720529.2021.2016224.

[2] S. Rawal, B. Rawal, A. Shaheen and S. Malik. "Phishing Detection in E-mails using Machine Learning", International Journal of Applied Information Systems (IJAIS), Vol. 12 – No. 7, October 2017, DOI:10.5120/ijais2017451713.

[3] A. Zamir, H. U. Khan and T. Iqbal, N. Yousaf, F. Aslam, A. Anjum and M. Hamdani. "Phishing web site detection using diverse machine learning algorithms", The Electronic Library, Vol. 38 No. 1, pp. 65-80, 2020, DOI: 10.1108/EL-05-2019-0118.

[4] V. A. Onih. "Phishing Detection Using Machine Learning: A Model Development and Integration", International Journal of Scientific and Management Research, Vol. 7, No. 04, pp. 27-63, April 2024.

[5] M. A. Tamal, M. K. Islam, T. Bhuiyan, A. Sattar. "Dataset of suspicious phishing URL detection", Front. Comput. Sci., Vol. 6, Mar 2024, DOI: 10.3389/fcomp.2024.1308634.

[6] G. Prasaath M, I. Khan I, M. Lingam K, R. Ramya. "Phishing website detection system using machine learning algorithms", IJCRT, Vol. 11, No. 11, November 2023.

[7] F. P. E. Putra, Ubaidi, A. Zulfikri, G. Arifin, R. M. Ilhamsyah. "Analysis of Phishing Attack Trends, Impacts and Prevention Methods: Literature Study", Brilliance Research of Artificial Intelligence, Vol. 4, No. 1, May 2024. DOI: https://doi.org/10.47709/brilliance.v4i1.4357

[8] S. S. M. Aldaham, O. Ouda, A. A. Abd El-Aziz, "Improved Detection of Phishing Websites using Machine Learning", International Journal of Intelligent Systems and Applications in Engineering, Vol. 12, No. 21, pp. 4619-4633, March 2024.

[9] N. N. Sakhare, J. L. Bangare, R. G. Purandare, D. S. Wankhede, P. Dehankar. "Phishing Website Detection Using Advanced Machine Learning Techniques", International Journal of Intelligent Systems and Applications in Engineering, Vol. 12, No. 12, pp. 329-, January 2024.

[10] A. Garje, N. Tanwani, S. Kandale, T. Zope, S. Gore. "Detecting Phishing Websites Using Machine Learning", IJCRT, Vol. 9, No. 11, pp. 243-246 November 2021.

[11] M. A. Saedi, N. A. Flayh. "Phishing Website Detection Using Machine Learning: A Review", Wasit Journal of Pure Sciences, Vol. 2, No. 2, pp. 270-281, June 2023, DOI: 10.31185/wjps.145.

[12] S. Alnemari, M. Alshammari. "Detecting Phishing Domains Using Machine Learning", Vol. 13, No. 8, DOI: https://doi.org/10.3390/app13084649.