MACHINE LEARNING-BASED CLIENT-SIDE DEFENSE AGAINST WEB SPOOFING ATTACKS

Dr. Thanveer Jahan[1*], Ch. Vyshnavi[2], P. k. Pranay sai[2], Pavan kalyan[2], Surya Teja[2], Kandikonda Kamal[2]

[1]Associate Professor & Head, [2]UG Student, [1,2]Department of CSE(AI&ML)

[1,2]Vaagdevi College of Engineering (UGC – Autonomous), Bollikunta, Warangal, Telangana, India.

*Corresponding Email: Dr. Thanveer Jahan (thanvivcecse@gmail.com)

ABSTRACT

Web spoofing attacks pose a significant threat to the security and trustworthiness of online communications and transactions. These attacks involve cybercriminals impersonating legitimate websites to deceive users into disclosing sensitive information or performing unintended actions. Traditional defenses against such threats primarily rely on server-side solutions, such as SSL/TLS encryption and domain validation techniques. However, these methods are often reactive and suffer from key limitations. They typically detect spoofing only after an attack has begun, exposing users during a critical vulnerability window. Moreover, distinguishing between authentic and spoofed websites remains a challenge, leading to false positives and negatives.

To address these shortcomings, there is a pressing need for proactive, client-side defense mechanisms capable of identifying spoofing attempts in real-time. In response, we propose PISHCATCHER, a novel, machine learning-driven solution designed to operate on the client side. By analyzing a combination of web page attributes such as HTML structure, CSS styling, and JavaScript behavior PISHCATCHER leverages advanced machine learning algorithms to accurately differentiate between genuine and spoofed websites. Furthermore, the system continuously learns and adapts to evolving spoofing tactics, ensuring robust and dynamic protection for users. This approach represents a significant advancement in defending against web spoofing threats and filling critical gaps in current cybersecurity infrastructure.

Keywords: Web spoofing, Client-side protection, Real-time detection, Phishing prevention.

1. INTRODUCTION

Web spoofing attacks pose a significant threat to online security by allowing malicious actors to impersonate legitimate websites, deceiving users into divulging sensitive information or engaging in harmful actions. To combat this threat effectively, robust defense mechanisms are essential to identify and thwart spoofing attempts in real-time. While current approaches primarily rely on server-side defenses like SSL/TLS protocols and domain validation techniques, they suffer from limitations such as reactivity and difficulty in accurately differentiating between legitimate and spoofed websites.

To address these challenges, the proposed PISHCATCHER system offers a novel approach to combat web spoofing attacks. Operating at the client-side, PISHCATCHER leverages machine learning techniques to analyze various features extracted from web pages, including HTML structure, CSS styles, and JavaScript behavior. By employing advanced machine learning algorithms, PISHCATCHER accurately distinguishes between legitimate and spoofed websites, enabling real-time detection and prevention of spoofing attempts.

2. LITERATURE SURVEY

doi: 10.48047/ijiee.2025.15.4.60

W. Khan et al. [1] proposed SpoofCatch, a client-side protection tool designed to combat phishing attacks. Their research focused on developing a tool that operates on the client side to detect and prevent phishing attempts by analyzing and filtering out potentially harmful content. This approach aimed to enhance user security by providing an additional layer of protection against phishing threats. B. Schneier [2] discussed the limitations of two-factor authentication (2FA) in his article. He argued that while 2FA provides an extra layer of security, it is often insufficient in protecting against sophisticated attacks. Schneier's analysis highlighted the need for more robust security measures beyond 2FA to effectively combat modern cybersecurity threats. S. Garera et al. [3] introduced a framework for detecting and measuring phishing attacks. Their work provided a structured approach to identifying phishing threats by using various detection techniques and metrics. This framework aimed to improve the accuracy and reliability of phishing detection systems.

R. Oppliger and S. Gajek [4] presented methods for effective protection against phishing and web spoofing. They explored various strategies to safeguard users from phishing attacks and website spoofing, focusing on enhancing the security of web interactions and preventing unauthorized access. T. Pietraszek and C. V. Berghe [5] addressed the issue of injection attacks through context-sensitive string evaluation. Their research proposed a method to defend against these attacks by analyzing and evaluating strings in context, which aimed to reduce vulnerabilities in web applications and enhance overall security. M. Johns et al. [6] focused on providing reliable protection against session fixation attacks. Their work involved developing mechanisms to prevent attackers from exploiting session fixation vulnerabilities, thereby improving the security of web sessions and protecting user data. M. Bugliesi et al. [7] worked on automatic and robust client-side protection for cookie-based sessions. Their research aimed to enhance the security of cookie-based sessions by implementing client-side protection mechanisms that could automatically detect and mitigate potential threats. A. Herzberg and A. Gbara [8] discussed strategies for protecting web users from spoofing and phishing attacks. They proposed solutions to safeguard even less experienced users from these threats, focusing on improving the overall security of web interactions and enhancing user awareness.

N. Chou et al. [9] presented client-side defenses against web-based identity theft. Their research focused on developing methods to protect users from identity theft by analyzing and securing web interactions, thereby reducing the risk of unauthorized access to personal information. B. Hämmerli and R. Sommer [10] edited proceedings from the 4th International Conference DIMVA 2007, which covered various aspects of intrusion detection and malware vulnerability assessment. The conference proceedings included research on methods and techniques for detecting and mitigating security threats in diverse computing environments.

C. Yue and H. Wang [11] introduced BogusBiter, a transparent protection mechanism against phishing attacks. Their approach aimed to provide a seamless and effective solution for detecting and preventing phishing attempts, enhancing user security without requiring significant changes to existing systems.

Vijayalaxmi Gopu [12] M. Selvi Development of “A Novel Deep Neural Network Approach to Diabetic Retinopathy Diagnosis”, 2024 First International Conference on Innovations in Communications, Electrical and Computer Engineering (ICICEC),Page Numbers – (01-07) ,DOI: 10.1109/ICICEC62498.2024.10808527,IEEE 2024.

3. PROPOSED SYSTEM

doi: 10.48047/ijiee.2025.15.4.60

The project showcases a comprehensive approach to developing a robust system for detecting phishing URLs, leveraging advanced machine learning techniques and thorough data analysis. Here is the overview of the project:

Initialization and Dataset Handling: The project intiaited by importing necessary Python packages and libraries, including those for data handling (pandas, numpy), machine learning (sklearn, xgboost), visualization (matplotlib, seaborn), and URL parsing (urllib). This ensures all essential tools are available for subsequent tasks. The dataset, located in "Dataset/phish_tank_storm.csv", is loaded using pandas, which provides a robust framework for data manipulation. Missing values within the dataset are replaced with zeros to prevent any computational errors during feature extraction and model training. The dataset comprises URLs labeled as either legitimate (0) or phishing (1), serving as the foundation for model training and evaluation.

Exploratory Data Analysis (EDA): Exploratory Data Analysis is conducted to gain an initial understanding of the dataset's composition. The labels are grouped and counted to determine the number of legitimate and phishing URLs. This distribution is visualized using a bar plot, offering a clear picture of class balance within the dataset. Such visualization is crucial as it highlights any potential class imbalance, which can significantly impact model performance and necessitate techniques such as oversampling, undersampling, or class weighting during model training.

Feature Engineering: Feature engineering is a critical step where meaningful features are extracted from the raw URL data to improve model performance. A custom function, get_features(), is defined to extract various features from URLs. This function computes the length of different URL components (e.g., domain, path) and counts occurrences of specific characters (e.g., dots, hyphens, slashes) within these components. These features help capture patterns that can differentiate legitimate URLs from phishing ones. The extracted features are saved in a processed dataset file ("processed.csv") for consistency and to avoid redundant computations in future runs.

Data Preprocessing: In this step, the dataset undergoes further preprocessing to ensure it is suitable for machine learning tasks. Non-numeric data such as URL components are transformed into numeric features using the previously defined get_features() function. The dataset is then reloaded, and any missing values are filled. Target labels are converted into numeric format to facilitate model training. The preprocessed dataset is split into feature variables (X) and target labels (Y), normalized using MinMaxScaler to scale the features within a specific range, and then shuffled to ensure randomness during model training.

Machine Learning Model Training: Three distinct machine learning models are trained on the preprocessed dataset: Support Vector Machine (SVM), Random Forest, and XGBoost (Extreme Gradient Boosting). Each model is trained on the features and corresponding labels to distinguish between legitimate and phishing URLs. The SVM model is trained using SVC from sklearn.svm, the Random Forest model using RandomForestClassifier from sklearn.ensemble, and the XGBoost model using XGBClassifier from xgboost. If trained models already exist, they are loaded from files to save computation time; otherwise, new models are trained and saved for future use.

Model Evaluation: The trained models are evaluated on a test set to measure their performance. Metrics such as accuracy, precision, recall, and F1-score are computed to provide a comprehensive assessment of each model's effectiveness. Confusion matrices are plotted to visualize the distribution of true positives, true negatives, false positives, and false negatives, offering insights into each model's strengths and weaknesses. These evaluations help identify the most effective model for phishing URL detection based on various performance criteria.

doi: 10.48047/ijiee.2025.15.4.60

Performance Comparison: Performance metrics of the three models—SVM, Random Forest, and XGBoost—are compared to determine the best-performing algorithm. The comparison is visualized using bar graphs, which provide a clear and concise representation of each model's precision, recall, F1-score, and accuracy. Additionally, a tabular format presents the detailed performance metrics for easy reference. This comparative analysis highlights the strengths and weaknesses of each model, aiding in the selection of the most suitable model for deployment.

Prediction on Test Data: The trained XGBoost model, identified as the most promising, is used to predict the nature of URLs from a separate test dataset ("Dataset/testData.csv"). Each URL in the test dataset undergoes feature extraction using the get_features() function, and the model predicts whether the URL is legitimate or phishing based on these features. This step demonstrates the model's practical application in real-world scenarios, providing a mechanism to evaluate new URLs for potential phishing threats.

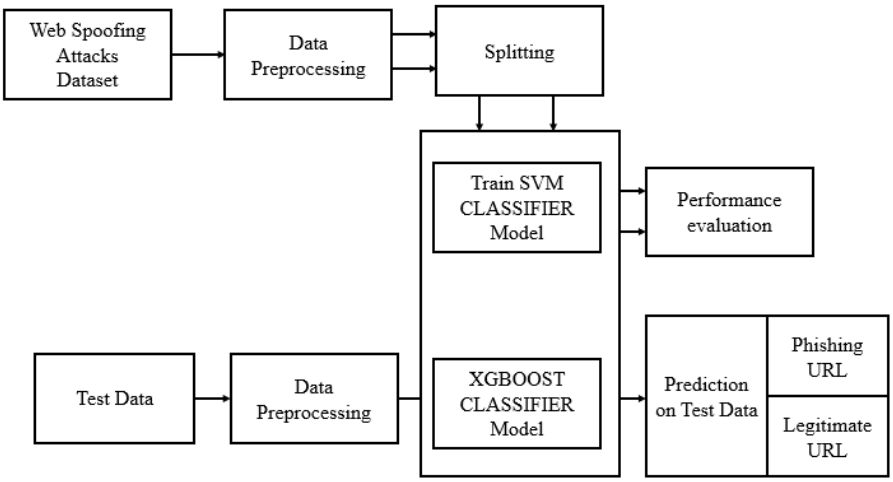


Fig. 1: Block Diagram of Proposed System

3.1 Data Preprocessing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while crea ting a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

Importing Libraries: To perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

doi: 10.48047/ijiee.2025.15.4.60

import numpy as nm

Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. Here, we have used pd as a short name for this library. Consider the below image:

```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

Handling Missing data: The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset. There are mainly two ways to handle missing data, which are:

- By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.
- By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc.

Encoding Categorical data: Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, Country, and Purchased. Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

Feature Scaling: Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no variable dominates the other variable. A machine learning model is based on Euclidean distance, and if we do not scale the variable, then it will cause some issue in our machine learning model. Euclidean distance is given as:

doi: 10.48047/ijiee.2025.15.4.60

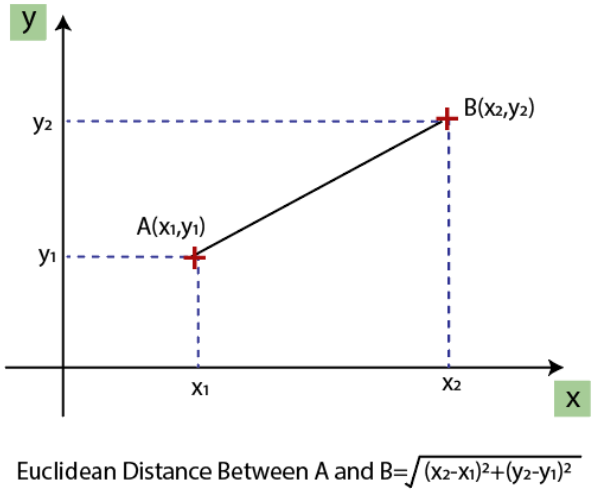


Figure 2: Feature scaling.

If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So, to remove this issue, we need to perform feature scaling for machine learning.

3.2 ML MODELS

3.2.1 PROPOSED SYSTEM XGBOOST Model

XGBoost is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "XGBoost is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the XGBoost takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

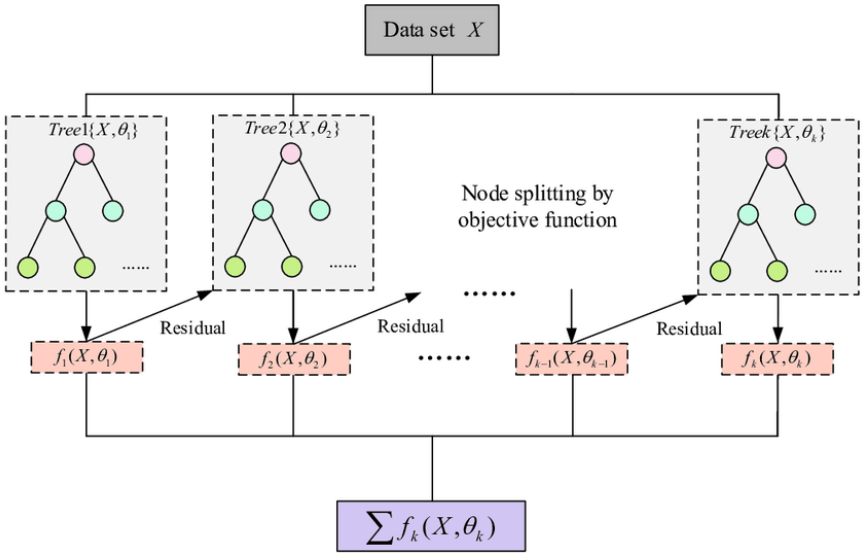


Fig. 3: XGBoost algorithm.

doi: 10.48047/ijiee.2025.15.4.60

XGBoost, which stands for "Extreme Gradient Boosting," is a popular and powerful machine learning algorithm used for both classification and regression tasks. It is known for its high predictive accuracy and efficiency, and it has won numerous data science competitions and is widely used in industry and academia. Here are some key characteristics and concepts related to the XGBoost algorithm:

Gradient Boosting: XGBoost is an ensemble learning method based on the gradient boosting framework. It builds a predictive model by combining the predictions of multiple weak learners (typically decision trees) into a single, stronger model.

Tree-based Models: Decision trees are the weak learners used in XGBoost. These are shallow trees, often referred to as "stumps" or "shallow trees," which helps prevent overfitting.

Objective Function: XGBoost uses a specific objective function that needs to be optimized during training. The objective function consists of two parts: a loss function that quantifies the error between predicted and actual values and a regularization term to control model complexity and prevent overfitting. The most common loss functions are for regression (e.g., Mean Squared Error) and classification (e.g., Log Loss).

Gradient Descent Optimization: XGBoost optimizes the objective function using gradient descent. It calculates the gradients of the objective function with respect to the model's predictions and updates the model iteratively to minimize the loss.

Regularization: XGBoost provides several regularization techniques, such as L1 (Lasso) and L2 (Ridge) regularization, to control overfitting. These regularization terms are added to the objective function.

Parallel and Distributed Computing: XGBoost is designed to be highly efficient. It can take advantage of parallel processing and distributed computing to train models quickly, making it suitable for large datasets.

Handling Missing Data: XGBoost has built-in capabilities to handle missing data without requiring imputation. It does this by finding the optimal split for missing values during tree construction.

Feature Importance: XGBoost provides a way to measure the importance of each feature in the model. This can help in feature selection and understanding which features contribute the most to the predictions.

Early Stopping: To prevent overfitting, XGBoost supports early stopping, which allows training to stop when the model's performance on a validation dataset starts to degrade.

Scalability: XGBoost is versatile and can be applied to a wide range of machine learning tasks, including classification, regression, ranking, and more.

Python and R Libraries: XGBoost is available through libraries in Python (e.g., **xgboost**) and R (e.g., **xgboost**), making it accessible and easy to use for data scientists and machine learning practitioners.

XGBoost, which stands for eXtreme Gradient Boosting, is a popular machine learning algorithm that is particularly effective for structured/tabular data and is often used for tasks like classification, regression, and ranking. It is an ensemble learning technique based on decision trees. Here's how XGBoost operates:

doi: 10.48047/ijiee.2025.15.4.60

Ensemble Learning: XGBoost is an ensemble learning method, which means it combines the predictions from multiple machine learning models to make more accurate predictions than any single model. It uses an ensemble of decision trees, known as "boosted trees."

Boosting: Boosting is a sequential technique in which multiple weak learners (usually decision trees with limited depth) are trained one after the other. Each new tree tries to correct the errors made by the previous ones.

Gradient Boosting: XGBoost is a gradient boosting algorithm. It minimizes a loss function by adding weak models (trees) that minimize the gradient of the loss function at each stage. This is done by fitting a tree to the residuals (the differences between the predicted and actual values) of the previous model.

Regularization: XGBoost includes L1 (Lasso regression) and L2 (Ridge regression) regularization terms in the objective function to prevent overfitting. These regularization terms help control the complexity of individual trees and reduce the risk of overfitting the training data.

Tree Pruning: XGBoost uses a technique called "pruning" to remove branches of the trees that do not contribute significantly to the model's predictive power. This reduces the complexity of the trees and helps prevent overfitting.

4. RESULTS AND DISCUSSION

4.1 Dataset Description

The given dataset contains information about URLs and their characteristics, for the purpose of classifying them as either legitimate or phishing URLs. Here's a detailed description of each column in the dataset:

url: This column contains the URLs that are being analyzed. Each entry in this column is a string representing a web address.

ranking: This column contains a ranking value associated with each URL, which is based on factors such as traffic, popularity, or search engine ranking.

mld_res: This column contains a feature related to the main domain of the URL after some processing or resolution. "mld" stand for "main level domain."

mld.ps_res: Similar to mld_res, this column contains another processed or resolved feature related to the main domain, a secondary or more specific aspect.

card_rem: This column contain values representing a certain characteristic or feature of the URL after some form of "removal" or processing. "card" stand for "cardinality" or some metric related to unique elements in the URL.

ratio_Rrem: This column contains a ratio value related to the "Rrem" characteristic of the URL. It represents the ratio of a certain type of element removed or retained in the URL.

ratio_Arem: This column contains a ratio value related to the "Arem" characteristic, similar to ratio_Rrem, but focusing on a different aspect or type of element.

jaccard_RR: This column contains the Jaccard similarity coefficient between the "R" elements of the URLs, which measures the similarity between two sets.

doi: 10.48047/ijiee.2025.15.4.60

jaccard_RA: This column contains the Jaccard similarity coefficient between the "R" and "A" elements of the URLs.

jaccard_AR: This column contains the Jaccard similarity coefficient between the "A" and "R" elements of the URLs.

jaccard_AA: This column contains the Jaccard similarity coefficient between the "A" elements of the URLs.

jaccard_ARrd: This column contains the Jaccard similarity coefficient between the "AR" elements after some form of reduction or processing (denoted by "rd").

jaccard_ARrem: This column contains the Jaccard similarity coefficient between the "AR" elements after removal or some form of processing (denoted by "rem").

label: This column contains the labels for the URLs, with 0 indicating legitimate URLs and 1 indicating phishing URLs. This is the target variable for the classification task.

4.2 Results Description

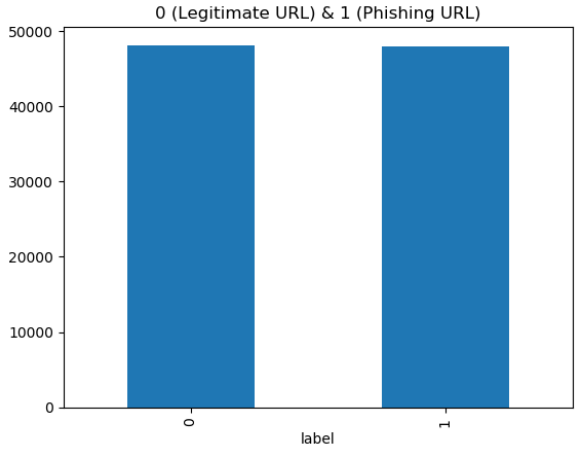
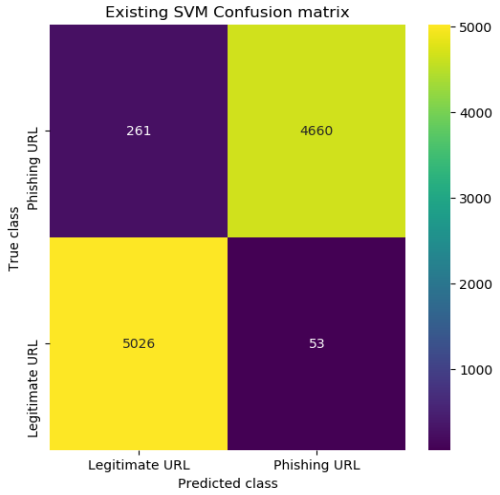


Fig. 4: Shows the Count plot of Phishing column in dataset.

The figure 4 illustrates the Count plot of the Phishing column, providing a visual representation of the distribution of phishing versus non-phishing instances in the dataset.



doi: 10.48047/ijiee.2025.15.4.60

Fig. 5: Confusion Matrix of SVM Classifier.

The figure 5 shows the Confusion Matrix for the SVM Classifier, detailing the true positive, true negative, false positive, and false negative values, which indicate the classifier's performance on the test data.

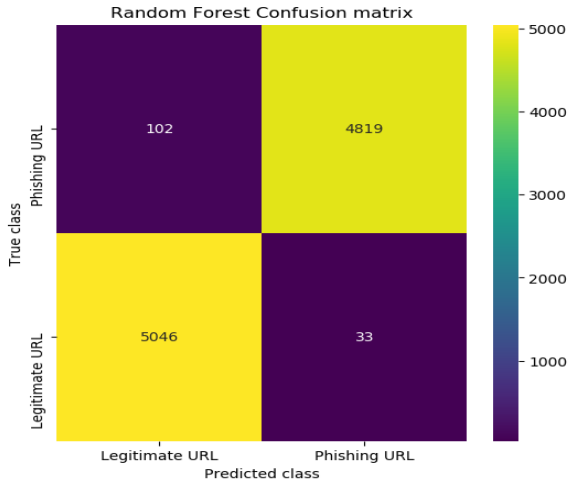


Fig. 6: Confusion Matrix of RFC Classifier.

The figure 6 displays the Confusion Matrix for the RFC Classifier, highlighting its accuracy by showing the distribution of correct and incorrect predictions on the test set.

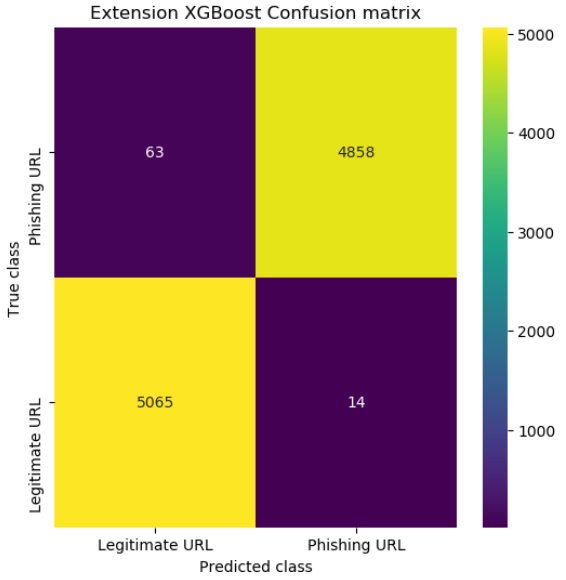


Fig. 7: Confusion Matrix of XGBoost Classifier.

The figure 7 presents the Confusion Matrix for the XGBoost Classifier, summarizing its predictive accuracy by indicating the number of true and false classifications made.

doi: 10.48047/ijiee.2025.15.4.60

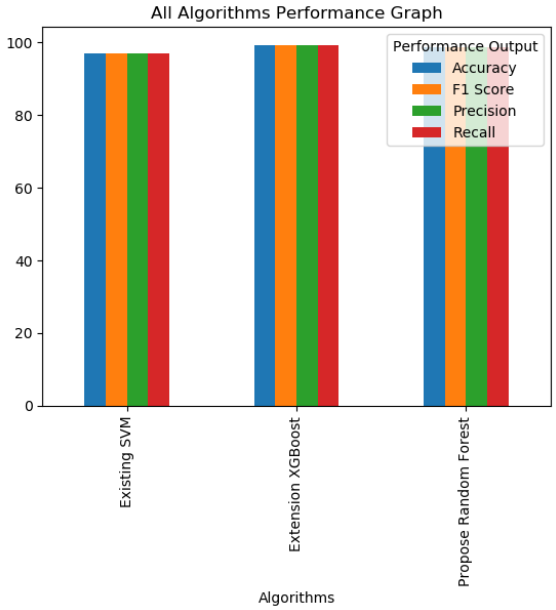


Fig. 8: Performance Comparison Graph of SVM, RFR, XGBoost Classifiers.

The figure 8 depicts a Performance Comparison Graph of the SVM, RFC, and XGBoost classifiers. It compares their performance metrics, providing a clear visual of their effectiveness in identifying phishing instances.

Table 1: Performance Metrics of SVM, RFR, XGBoost Algorithms

Algorithm Name	Precision	Recall	FScore	Accuracy
Existing SVM	96.97%	96.83%	96.86%	96.86%
Propose Random Forest	98.67%	98.64%	98.65%	98.65%
Extension XGBoost	99.24%	99.22%	99.23%	99.23%

The table 1 tells the performance metrics of three different machine learning algorithms: Existing SVM, Proposed Random Forest, and Extension XGBoost. The metrics evaluated include Precision, Recall, FScore, and Accuracy, all of which are expressed as percentages.

```
https://www.education-online.nl/Cliquez.ici.cas.inria.fr.cas.login/login.html ====> Predicted AS PHISHING

http://www.revistas-academicas.com ====> Predicted AS PHISHING

http://www.google.com ====> Predicted AS SAFE

http://www.yahoo.com ====> Predicted AS SAFE

http://www.horizonsgallery.com/js/bin/ssl1/_id/www.paypal.com/fr/cgi-bin/webscr/cmd=_registration-run/login.php?cmd=_login-run&
amp;dispatch=1471c4bdb044ae2be9e2fc3ec514b88b1471c4bdb044ae2be9e2fc3ec514b88b ====> Predicted AS PHISHING

http://www.phlebolog.com.ua/libraries/joomla/results.php ====> Predicted AS PHISHING

http://www.docs.google.com/spreadsheet/viewform?formkey=dE5rVEdSV2pBdkpSRy11V3o2eDdwbnc6MQ ====> Predicted AS PHISHING
```

Fig. 9: Proposed Model prediction on test data.

doi: 10.48047/ijiee.2025.15.4.60

Figure 9 illustrates the performance of the proposed XGBoost model in predicting the classification of URLs on a given test dataset. The figure provides a visual representation of how effectively the model distinguishes between legitimate and phishing URLs based on the features extracted from the dataset.

5. CONCLUSION

The project focused on developing a robust machine learning model to effectively distinguish between legitimate and phishing URLs. Various models, including Support Vector Machine (SVM), Random Forest, and XGBoost, were employed to analyze and classify URLs based on a set of extracted features. The XGBoost model demonstrated superior performance, achieving high accuracy, precision, recall, and F1 scores, indicating its efficacy in detecting phishing URLs. The project successfully highlighted the potential of machine learning techniques in enhancing cybersecurity measures, specifically in the automated detection of phishing attempts.

REFERENCES

[1] W. Khan, A. Ahmad, A. Qamar, M. Kamran, and M. Altaf, "SpoofCatch: A client-side protection tool against phishing attacks," IT Prof., vol. 23, no. 2, pp. 65-74, Mar. 2021.

[2] B. Schneier, "Two-factor authentication: Too little too late," Commun. ACM, vol. 48, no. 4, pp. 136, Apr. 2005.

[3] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," Proc. ACM Workshop Recurring malcode, pp. 1-8, Nov. 2007.

[4] R. Oppliger and S. Gajek, "Effective protection against phishing and web spoofing," Proc. IFIP Int. Conf. Commun. Multimedia Secur., pp. 32-41, 2005.

[5] T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," Proc. Int. Workshop Recent Adv. Intrusion Detection, pp. 124-145, 2005.

[6] M. Johns, B. Braun, M. Schrank, and J. Posegga, "Reliable protection against session fixation attacks," Proc. ACM Symp. Appl. Comput., pp. 1531-1537, 2011.

[7] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "Automatic and robust client-side protection for cookie-based sessions," Proc. Int. Symp. Eng. Secure Softw. Syst., pp. 161-178, 2014.

[8] A. Herzberg and A. Gbara, "Protecting (even naive) web users from spoofing and phishing attacks," 2004.

[9] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell, "Client-side defense against web-based identity theft," Proc. NDSS, 2004.

[10] B. Hämmerli and R. Sommer, "Detection of Intrusions and Malware and Vulnerability Assessment: 4th International Conference DIMVA 2007 Lucerne Switzerland July12-132007 Proceedings," vol. 4579, 2007.

[11] C. Yue and H. Wang, "BogusBiter: A transparent protection against phishing attacks," ACM Trans. Internet Technol., vol. 10, no. 2, pp. 1-31, May 2010.

doi: 10.48047/ijiee.2025.15.4.60

[12] Vijayalaxmi Gopu; M. Selvi Development of A Novel Deep Neural Network Approach to Diabetic Retinopathy Diagnosis.2024 DOI: 10.1109/ICICEC62498.2024.10808527.

[13]

doi: 10.48047/ijiee.2025.15.4.60