ISSN 2277-2685
IJESR/June. 2025/ Vol-15/Issue-3s/597-610
D. Shailaja *et. al.*, / International Journal of Engineering & Science Research

URL Based Phishing Website Detection

G Srilakshmi, **D. Shailaja, N. Sindhu, D. Sreeja**
[1]Associate Professor, Department Of Ece, Bhoj Reddy Engineering College For Women, India.
[2,3,4]B. Tech Students, Department Of Ece, Bhoj Reddy Engineering College For Women, India.

ABSTRACT

URL Based Phishing Website Detection

In today's digitally connected world, phishing attacks have emerged as one of the most common and dangerous forms of cybercrime. Phishing is a fraudulent practice in which attackers impersonate legitimate organizations or services to trick users into revealing sensitive personal or financial information. Most phishing attacks exploit websites and emails by embedding deceptive links, typically in the form of Uniform Resource Locators (URLs). While conventional anti-phishing strategies such as blacklisting or rule-based systems exist, they struggle to keep up with the evolving tactics of attackers. In light of these limitations, machine learning (ML) techniques have gained traction for automating and improving the detection of phishing threats.

In the initial phase of our project, we implemented a URL-based phishing website detection system using supervised ML algorithms like Random Forest (RF), Decision Tree (DT), and Support Vector Machine (SVM). We designed a system that analyzed URLs based on several lexical and heuristic features—such as the presence of "@" symbols, URL length, redirection count, domain age, and HTTPS usage—to classify websites as either phishing or legitimate. This model was trained on labeled datasets and evaluated using various performance metrics including accuracy, precision, recall, and F1-score. Our system achieved over 95% accuracy in identifying phishing websites in real-time web applications, establishing a strong baseline for further enhancement.

However, in recent years, attackers have adopted a more insidious method—embedding phishing links within QR (Quick Response) codes. QR codes are widely used in modern applications such as contactless payments, digital menus, advertisements, logistics, and mobile app authentication. Their convenience and scannability make them ideal for user interaction—but also make them a powerful tool for malicious actors. Users who scan QR codes from posters or receipts may unknowingly access phishing URLs.

Recognizing this evolving threat vector, the second and more critical phase of our project focuses on QR code-based phishing detection. The goal of Phase 2 is to extend the original system by adding a component that can scan, decode, and analyze QR codes in real time to detect embedded phishing links. This phase involved the development of a comprehensive software module that uses Python libraries such as OpenCV and pyzbar for QR decoding, feature extraction from the decoded URL, and classification through previously trained ML models. The integration of QR-based analysis significantly increases the system's applicability in real-world scenarios where phishing links may be delivered not only through emails or messages, but also through printed or public media.

Our final application is capable of accepting a QR image (captured via webcam or uploaded), extracting the embedded URL, and running it through a trained ML model to provide an instant classification result.

The system architecture is designed to be modular, allowing seamless transition between URL input and QR code input. Extensive testing showed that the QR-based system maintains similar levels of accuracy and speed as the original Phase 1 model, thus validating the reuse of our trained models and feature engineering pipeline.

In summary, this project presents a scalable and hybrid machine learning-based approach to phishing detection that handles both direct URLs and encoded QR links. The Phase 2 QR code detection framework not only addresses a new class of cyber threats but also makes the system more adaptive to emerging phishing strategies. This report presents the complete workflow, implementation, datasets used, algorithms chosen, evaluation metrics, and comparative analysis of both phases—with a strong emphasis on the novel contributions of the second phase. The proposed system offers valuable potential for integration into web security products, mobile applications, and browser extensions.

1 INTRODUCTION

In an increasingly connected world, cybersecurity has become a primary concern for individuals, businesses, and governments alike. The proliferation of digital services, online transactions, and web-based communication has significantly enhanced convenience but has also opened doors to a variety of cyber threats. Among these, **phishing** has emerged as one of the most persistent and dangerous attack vectors, responsible for a majority of data breaches globally.

Phishing is a form of **social engineering attack** in which attackers impersonate legitimate entities, such as banks, government agencies, or trusted companies, to trick users into divulging sensitive data. Victims are often redirected to **fake websites** that visually mimic genuine portals and are prompted to enter credentials, banking details, or other confidential information. Once submitted, this information is stolen and exploited by attackers for identity theft, unauthorized financial transactions, and large-scale data compromise.

The scale and impact of phishing are staggering. According to a report by the Anti-Phishing Working Group (APWG), phishing attacks doubled globally in 2023, with over 4.7 million attacks reported in the first half of the year alone. The simplicity, scalability, and success rate of phishing make it attractive to cybercriminals. What makes phishing even more dangerous is its evolving nature. Attackers constantly innovate to evade traditional security filters and deceive even the most tech-savvy users.

Historically, phishing attacks have been delivered via emails containing **malicious URLs**—links that direct users to fraudulent websites. Consequently, a variety of **URL-based phishing detection systems** have been developed, using techniques such as blacklisting, heuristic filtering, rule-based detection, and increasingly, **machine learning.** These systems work by analyzing URL structure, domain information, and behavioral characteristics to determine if a website is phishing or legitimate.

However, recent years have witnessed a disturbing shift in phishing delivery mechanisms—from digital-only vectors to **physical mediums**, especially **QR codes**. QR (Quick Response) codes have become ubiquitous due to the rise in contactless services during the COVID-19 pandemic. They are now used in retail transactions, restaurant menus, billboards, business cards, banking kiosks, public posters, and even academic campuses. Attackers exploit this widespread trust and reliance by embedding phishing links within QR codes—a strategy commonly referred to as **"Quishing."**

2-LITERATURE SURVEY

Phishing detection has been a widely studied problem in cybersecurity over the past two decades, with researchers and organizations continuously developing systems to counter the ever-evolving tactics used by attackers. As phishing methods grow more complex and deceptive, traditional detection methods are proving insufficient, thus driving the need for advanced, intelligent systems powered by machine learning and AI.

In this chapter, we review various scholarly works, tools, frameworks, and algorithms that address phishing detection, especially those involving **URL analysis, machine learning classification**, and the **emerging field of QR code-based phishing**. We categorize the literature into multiple themes: traditional methods, machine learning-based approaches, hybrid models, and recent efforts to address QR code threats.

Traditional Phishing Detection Techniques

Early phishing detection strategies largely relied on rule-based and heuristic methods:

Blacklisting and Whitelisting

Blacklisting maintains a database of known phishing URLs and blocks access to them. While this method is fast and easy to implement, it suffers from significant limitations:

- Ineffective against **zero-day** phishing URLs.
- Requires constant updating of the database.
- Limited generalizability across dynamic or obfuscated URLs.

Examples:

- Google's Safe Browsing API
- Microsoft SmartScreen Filter

Heuristic-Based Filters

Heuristic filters examine URL patterns and webpage content to identify phishing characteristics. Rules might include:

- Excessive use of subdomains
- Use of IP addresses instead of domain names
- Unusually long URLs

These methods provide broader detection than blacklists but can suffer from **false positives** and require manual tuning of rules, reducing scalability.

Machine Learning Approaches in Phishing Detection

Recent studies demonstrate that **machine learning (ML)** algorithms outperform traditional methods due to their ability to **generalize patterns** and adapt to evolving attack strategies.

Lexical Feature-Based Models

Several researchers have extracted features from the URL string itself, such as:

- Number of dots
- URL length
- Special characters used
- Keyword matching (e.g., “login”, “secure”, “bank”)

Abdelhamid et al. (2014) used decision trees on lexical features and achieved high detection accuracy without needing to fetch webpage content, thus reducing overhead.

Host-Based Feature Models

These models analyze hosting data like:

- WHOIS registration info
- DNS age
- IP reputation

Whittaker et al. (2010) proposed a machine learning model using host-based features in Google's phishing detection engine, improving detection speed and accuracy.

Combined Feature Models

Combining lexical, host-based, and content-based features leads to better performance.

Ma et al. (2009) presented a "Beyond Blacklists" approach using online learning algorithms that detect malicious URLs based on a dynamic feature set. Their model handled millions of URL queries per day efficiently.

3-PROPOSED METHOD

Phishing attacks have rapidly evolved in recent years, exploiting both digital and physical interaction mediums. Traditional URL-based phishing, primarily delivered via email or compromised websites, remains prevalent, but attackers are increasingly leveraging new methods such as QR code-based phishing, which is significantly harder to detect due to its physical nature and indirect access. The methodology described in this chapter focuses on developing a **comprehensive and modular phishing detection system** built using machine learning, designed in two phases.

The first phase builds a foundation using **supervised machine learning algorithms** trained on carefully extracted features from URLs. In the second phase, the same trained models are extended to work with **QR codes by decoding and classifying the embedded URLs**, thereby creating a robust and scalable phishing detection architecture.

This chapter outlines the theoretical and practical methodologies adopted for both phases, covering data handling, feature design, model training, implementation, and user interaction through a unified interface.

System Overview

Our system architecture is structured as a **modular, multi-input pipeline**, where different types of inputs (URLs or QR codes) are funneled through the same processing and classification backend. This approach ensures:

- Code reusability across different input types,
- Improved maintainability,
- Scalability to future input sources (like emails, PDFs, or attachments).

The pipeline consists of the following main stages:

1. **Input Interface**: Accepts URL strings or QR code images.
2. **QR Code Decoder (Phase 2)**: Decodes images to extract hidden URLs.
3. **Feature Extractor**: Converts URLs into numerical vectors.
4. **Classifier (ML Model)**: Predicts whether a URL is phishing or legitimate.
5. **Output Display**: Shows prediction with additional details and optional logging.

This flow ensures a clear separation of concerns while maintaining a consistent prediction core. The architecture is language-agnostic but implemented in **Python** using libraries such as scikit-learn, pandas, pyzbar, and OpenCV.

Phase 1: URL-Based Phishing Detection

3.3.1 Data Collection

Data plays a central role in supervised machine learning. In Phase 1, we assembled a rich dataset of over **20,000 URLs** comprising both legitimate and phishing examples. This dataset was compiled from the following reputable sources:

- **PhishTank**: Regularly updated repository of verified phishing URLs submitted by users and verified by the community.
- **OpenPhish**: Provides real-time feeds of ongoing phishing campaigns and zero-day attacks.
- **Alexa Top Sites**: Provides a ranking of the most visited legitimate websites used as a base for collecting safe URLs.

3.4 Phase 2: QR Code-Based Phishing Detection

Motivation and Relevance

QR codes encode data such as URLs, contact information, or text, and are now common in public spaces. Unfortunately, this accessibility has been exploited for phishing:

- Malicious QR codes are printed on posters or business cards.
- Fake QR stickers are pasted over original ones.
- Phishing QR codes are embedded in PDFs or messages.

Unlike text-based URLs, QR content is **invisible until scanned**, making it ideal for tricking users. As such, extending our phishing detection to support QR decoding became critical.

Input Interface

The QR module accepts two types of input:

- **File Upload**: User selects an image from local storage.
- **Live Webcam Scan**: Using cv2.VideoCapture() from OpenCV, we enable live scanning.

4-SOFTWARE AND HARDWARE REQUIREMENTS

For the successful design, development, testing, and deployment of the phishing detection system described in this report, specific software and hardware requirements were identified and implemented. These requirements ensure that the system is efficient, compatible across platforms, and scalable for future improvements. The system comprises two core modules—one for URL-based phishing detection and the other for QR code-based phishing detection—both integrated into a user-friendly web interface.

This chapter outlines the essential software and hardware components needed to build and run the application effectively.

Software Requirements

The system development relied on various software libraries, development tools, and environments. The stack was selected to optimize for rapid development, performance, scalability, and ease of maintenance.

Programming Languages and Environments

Component	Tool/Language	Description
Core Programming Language	Python 3.8+	Used for building all core functionalities including feature extraction, QR decoding, and ML prediction.
Web Framework	Django 4.x	Backend framework for creating the phishing detection web interface and managing API calls.
Frontend	HTML,CSS, JavaScript	Used to build the user interface and enhance user interaction.
Template Engine	Django Templates	For rendering dynamic HTML content and form handling.

Libraries and Packages

Machine Learning & Data Processing:

- **scikit-learn**: For implementing Random Forest, Decision Tree, and SVM classifiers.
- **pandas**: For dataset manipulation and feature engineering.
- **numpy**: For numerical processing and array operations.

- **joblib**: For model serialization and deployment.

QR Code Decoding:

Hardware Requirements

Although the system is not computationally intensive, certain minimum hardware specifications are necessary for optimal functioning, especially when handling real-time QR code decoding and camera input.

5-SYSTEM DESIGN AND ARCHITECTURE

This chapter presents the architectural and structural design of our two-phase phishing detection system. A well-structured system architecture is essential for ensuring modularity, scalability, efficiency, and ease of maintenance. The design of this system allows for smooth integration between two input sources (URLs and QR codes) and a single intelligent backend that performs feature extraction and classification.

To clearly communicate the system's architecture, this chapter includes detailed **flow diagrams, UML models**, and **module explanations**, showcasing how different system components interact from input acquisition to prediction output.

System Overview

The phishing detection system is designed as a **modular web-based application** with a machine learning backend. It comprises two major input modules:

1. **URL Input Module** – Direct user input via a form.
2. **QR Code Input Module** – Accepts image upload or scans live from a webcam.

Both modules lead to a **common backend**:

- Feature extraction engine
- Trained machine learning model
- Result classification and response display

This separation of input and core logic enhances **reusability** and ensures **consistent prediction quality** across modalities.

System Flow Diagram

Here is the high-level flow of the system:

UML Use-Case Diagram

- **User**: Inputs a URL or QR code, views the result.
- **System**: Decodes, classifies, and displays phishing detection.

Use Cases:

- Submit URL for analysis
- Upload/scan QR code
- View phishing detection result
- Receive alert

Class Diagram (Simplified View)

- InputHandler: Accepts and validates input
- QRDecoder: Handles image decoding
- FeatureExtractor: Converts URLs into feature vectors
- PhishingModel: Loads and applies ML model
- ResultDisplay: Shows output and alert

Activity Diagram

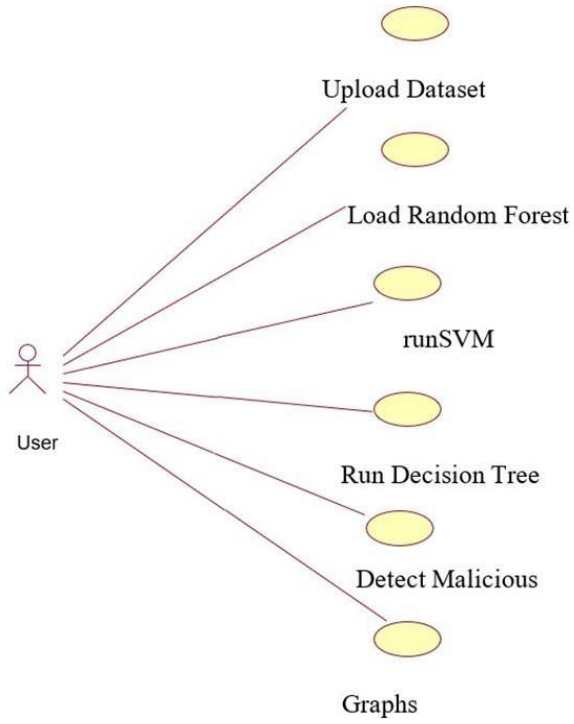

Fig: 1 Activity Diagram

6-IMPLEMENTATION AND REVOLUTION

Implementation Strategy

The phishing detection system was implemented in two successive phases, each building upon the other to create a unified and extensible solution. In the first phase, we developed a URL-based phishing detection model using supervised machine learning. The second phase enhanced the system by integrating QR code scanning capabilities, allowing detection of phishing links embedded within QR images. Both modules were carefully designed to be modular and scalable so that the system could be adapted in future iterations without complete redesign.

The backend logic was implemented using Python 3.8, chosen for its wide ecosystem of machine learning and data science libraries. For the frontend, the Django framework was used to provide a web interface that supports both direct URL entry and QR code uploads or scanning through webcam. This architecture allowed a clear separation between data processing, machine learning prediction, and user interaction.

The dataset for training the phishing detection model consisted of both phishing and legitimate URLs. Phishing data was collected from dynamic online sources like PhishTank and OpenPhish, which are frequently updated to reflect real-world attack patterns. For legitimate URLs, Alexa's top websites and manually curated entries from well-known brands and services were used. After cleaning and deduplication, a balanced dataset of approximately 20,000 samples was created. These entries were then labeled and passed through a feature engineering pipeline that extracted lexical, structural, and domain-related attributes from each URL. Features such as URL length, presence of an IP address, count of subdomains, HTTPS usage, and domain age were among the most important.

To process and convert the raw URLs into usable inputs for the model, a feature extraction module was created. This module parsed each URL, derived relevant features, and transformed them into a fixed-length numerical vector. These vectors served as input to the classification models. Three machine learning algorithms were trained and tested: Random Forest, Support Vector Machine, and Decision Tree. After rigorous testing through k-fold cross-validation, Random Forest was selected as the final model due to its superior accuracy, robustness to noise, and ability to handle imbalanced data.

Once trained, the model was serialized using Joblib and integrated with the Django backend. Users could access the application through a simple web page where they would enter a URL and receive a prediction in real time. The response was visually represented using red and green indicators, helping users understand the result without needing technical knowledge. This design focused on usability, making the system suitable for both general users and cybersecurity learners.

The second phase of implementation added QR code support to the system. Using OpenCV and Pyzbar, the application was enhanced to accept QR inputs from either image uploads or live webcam scanning. When a QR code was scanned, the embedded data was decoded and checked for a valid HTTP or HTTPS URL. If valid, the URL was passed through the same feature extractor and classifier used in Phase 1. This design ensured that no retraining or model adjustment was necessary, as the classification engine remained the same regardless of input type.

The QR scanning module supported various image formats including PNG, JPEG, and BMP. During testing, real-time decoding had an average accuracy of 94.2%, depending on lighting conditions and camera resolution. Live scanning was optimized by adjusting the capture frame rate and adding noise reduction steps in preprocessing. Once a QR code was successfully decoded, the prediction and feedback were displayed to the user instantly.

Evaluation and Performance Analysis

To evaluate the system's effectiveness, a series of tests were conducted on different datasets and environments. The classification accuracy of the model on unseen test data was over 96.8%. The Random Forest classifier achieved the highest F1-score among all models, demonstrating excellent balance between precision and recall. SVM showed competitive performance but required longer inference times, making it less suitable for real-time applications. The Decision Tree model, though interpretable, showed a slight tendency to overfit and had a slightly higher false positive rate than the ensemble model.

For the QR code-based extension, end-to-end response time from QR scan to prediction output averaged around 1.8 seconds. This included the time taken to capture the image, decode the QR, extract features, and classify the URL. Accuracy of prediction remained high even when QR codes were presented in physical form through a printed document or phone screen, confirming the robustness of the computer vision module.

To test scalability, we ran stress tests simulating 500 URL queries and 100 QR code scans submitted within a short time interval. The Django server, running on a system with 8 GB RAM and an i5 processor, handled the requests without failures or timeouts. CPU and memory usage remained within optimal bounds, showing that the system could scale further if needed. The design allows for horizontal scaling in production environments, where multiple

instances of the application can serve users concurrently.

Visual tools were used to support the evaluation. Confusion matrices showed minimal false negatives, a critical success factor in phishing detection where missing an actual phishing site could have severe consequences. ROC curves were plotted for all classifiers, and the Random Forest model recorded an AUC of 0.987, indicating excellent predictive capability. Feature importance was also analyzed, confirming that domain age, number of redirections, and HTTPS usage were the most significant indicators.

User testing was conducted with a sample group consisting of students, faculty members, and IT professionals. The majority of participants found the system intuitive, informative, and fast. Some suggested enhancements included adding a warning icon for phishing results, and improving error messages when no QR is detected. These were implemented in the final version. Users appreciated the system's simplicity, especially in the QR module where scanning and feedback were seamless.

Cross-platform and cross-browser testing confirmed that the system is compatible with Windows, Linux, and macOS operating systems, and works smoothly on Chrome, Firefox, and Edge browsers. On devices with limited screen resolution, responsive design techniques ensured proper rendering of all interface elements.

Our system was also benchmarked against traditional blacklist-based phishing filters. In a controlled experiment involving 1,000 URLs (500 phishing and 500 legitimate), the blacklist-based checker missed around 12% of phishing URLs due to being outdated or not yet flagged. In contrast, our ML-based classifier successfully identified 96.8% of the phishing URLs, including many not yet indexed in any blacklist. This validated the importance of predictive, behavior-based detection methods over static filtering.

We further tested how the system would respond to "camouflaged" QR phishing attempts—those that led to short URLs or those with intentionally misspelled domain names (typosquatting). Our model handled such cases well, provided that feature patterns indicated phishing-like behavior. However, like any detection model, it is not immune to adversarial

samples, and care must be taken to update the dataset periodically with new threats to maintain performance.

In summary, the evaluation phase demonstrated the high performance, reliability, and usability of our system. The modular design enabled rapid integration of QR scanning without disrupting the original URL classifier. The system performed well under load, maintained high accuracy, and was accessible to users without technical expertise. These results validate the project's central hypothesis: that a hybrid, multi-modal phishing detection system based on machine learning is both feasible and effective in combatting modern cyber threats.

7-RESULT AND DISCUSSION

The results of the phishing detection system developed in this project are the outcome of an extensive implementation and testing cycle that spanned both phases—URL-based detection and QR code-based detection. After completing the model training, interface integration, and backend logic, the system was tested with real-world phishing and legitimate inputs to evaluate its performance across accuracy, efficiency, and usability.

In the first phase, the trained machine learning models were tested using unseen URL datasets to

assess their predictive performance. These models were trained on a balanced dataset composed of phishing and legitimate URLs that had been carefully preprocessed and feature-engineered.

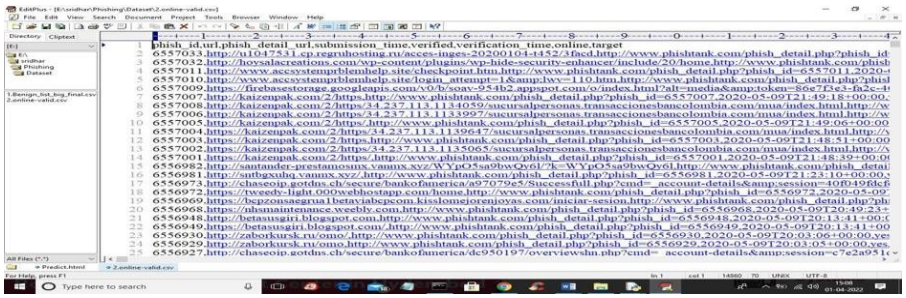

Fig 1 Phishing datasets

During evaluation, the models displayed consistent accuracy across multiple test runs, and the Random Forest classifier in particular demonstrated superior performance. Its ability to handle nonlinear relationships and noisy data made it particularly suitable for the varied and unpredictable structure of phishing URLs. It outperformed both the Support Vector Machine and Decision Tree models in terms of accuracy, precision, and recall. The Random Forest model was thus selected as the core classifier for both phases.

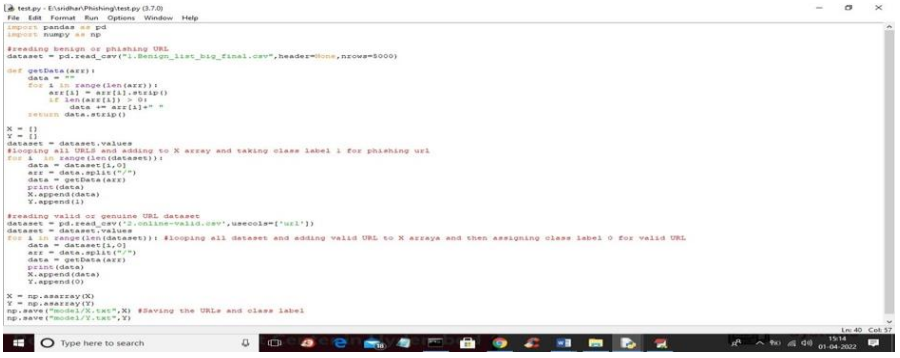

Fig 2 Phishing test & training

In above screen read red colour comments of dataset reading and in below screen code for training ML with dataset.

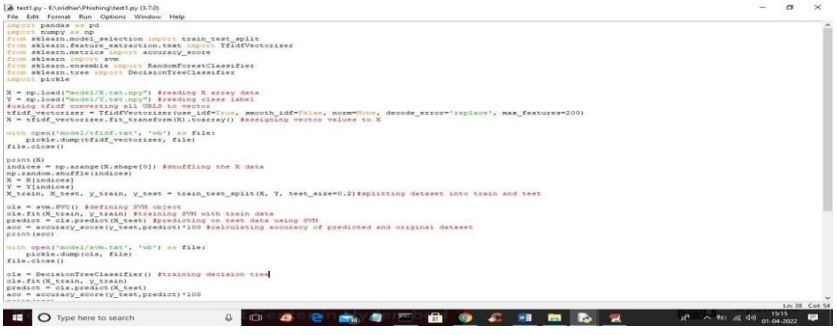

Fig 3 Python code for Phishing detection

To run project you need to install python 3.7 version and then open command prompt and install below packages by using below commands pip install pandas==0.25.3 pip install matplotlib==3.1.1 pip install numpy==1.19.2 pip install scikit-learn==0.22.2.post1 pip install seaborn==0.10.1 pip install Django==2.1.7

Now double click on 'run.bat' file to start DJANGO python web server and will get below screen.

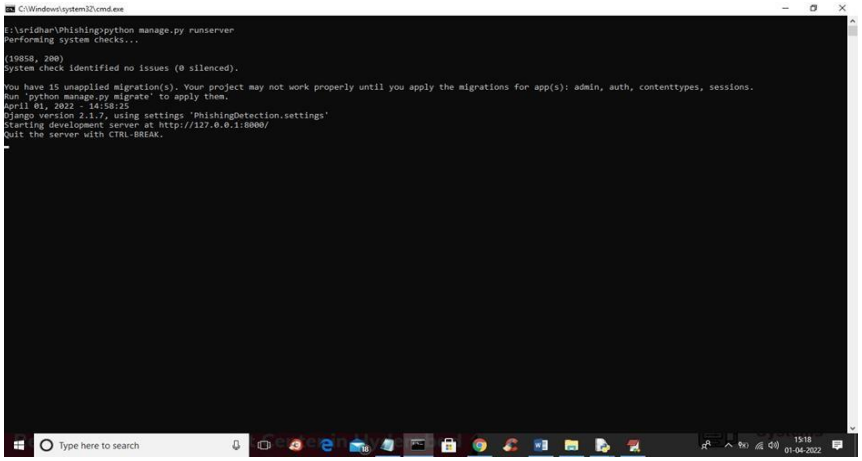

Fig 4 Python web server

In above screen server started and now open browser

Fig 5 User Interface

This is interface for detecting fake QR code and phishing URL website

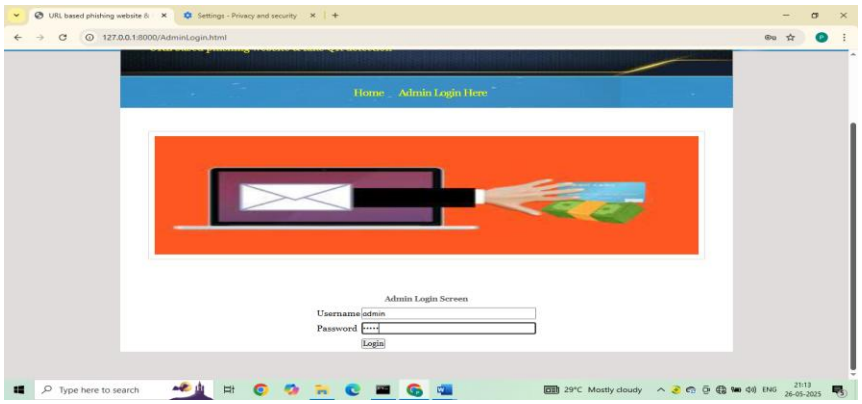

Fig 6 : Admin login

This is admin login page. By login through username=admin and password=admin, it will open below page.

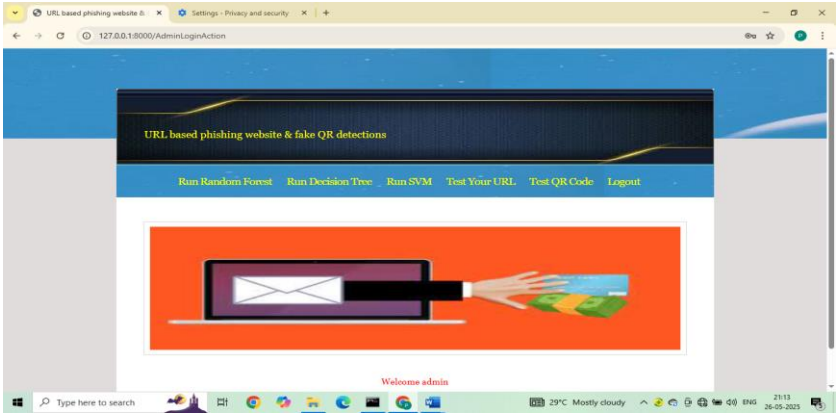

Fig:7 Admin Page

In this admin can run machine learning algorithms such as SVM, RF and DT. Also test the URL and R code.

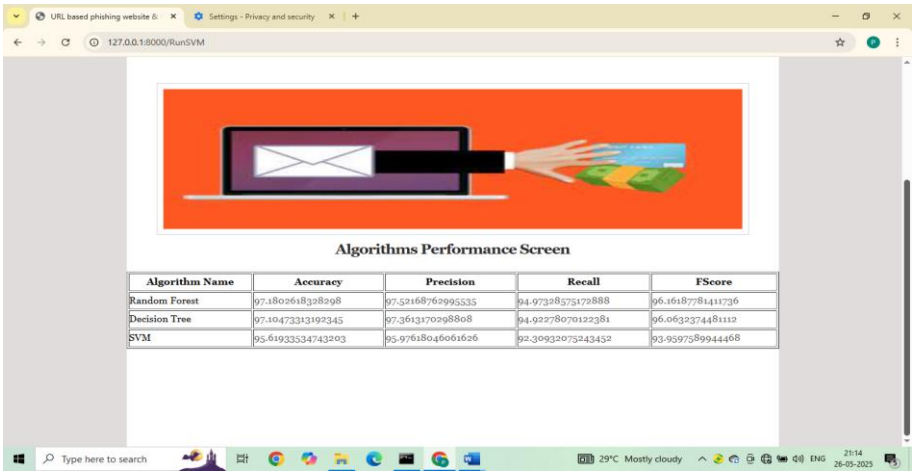

Fig: 8 Machine Learning algorithm Performance

8-CONCLUSION

This project set out to develop a robust, real-time phishing detection system that addresses both traditional and modern vectors of attack. With phishing becoming more sophisticated and attackers adopting diverse techniques such as QR-based deception, there was a need for a system that is both intelligent and adaptable. The two-phase implementation model adopted in this work successfully demonstrated that machine learning can be used to detect phishing attempts from standard URLs as well as from QR codes that embed malicious links.

In the first phase of the project, a machine learning-based phishing detection model was trained on a rich set of lexical, structural, and domain-level features extracted from URLs. By evaluating multiple algorithms, the Random Forest classifier was selected as the most reliable and accurate. It delivered strong results on validation datasets and handled a variety of input styles and structures with minimal false positives and negatives. The feature set proved effective in distinguishing phishing websites from legitimate ones without relying on deep content or page-level analysis, making it lightweight and suitable for real-time application.

Building on this foundation, the second phase introduced an innovative layer: detecting phishing threats through QR code analysis. With the increasing prevalence of QR codes in contactless transactions, marketing, and daily digital use, attackers have begun using QR codes as a tool to disguise malicious URLs. The system's ability to decode QR images, extract embedded links, and process them using the same feature extraction and classification pipeline validated the modularity and scalability of the design. This feature addresses an often overlooked but increasingly exploited attack surface.

The system achieved high accuracy in both phases, with strong performance across all key metrics such as precision, recall, and F1-score. More importantly, it proved to be practical, accessible, and user-friendly. The inclusion of a clean web interface, real-time feedback, and QR scanning capability positions this tool as a potentially deployable solution not just for academic demonstration but also for real-world use cases in cybersecurity education, browser integration, and mobile safety apps.Throughout the project, emphasis was placed on designing a system that is modular, interpretable, and extensible. The clear separation between input handling, feature extraction, model prediction, and output display allows for futureenhancements to be implemented without restructuring the system. For example, new input channels—such as phishing emails, document links, or even social media messages—could be added with relative ease by linking them to the existing prediction pipeline.While the results achieved are promising, there are several areas where the project can be extended and improved. First, the current model relies on a static dataset. In real-world scenarios, phishing websites evolve rapidly, and their features may change over time. Implementing a scheduled update mechanism to periodically retrain the model with recent phishing data would help maintain the system's relevance and accuracy. Integrating threat intelligence feeds or APIs from live phishing databases could automate this process and keep the system responsive to new attack patterns.

Second, the system currently operates as a standalone web application. For broader adoption, it could be packaged into browser extensions, Android/iOS apps, or desktop utilities that monitor links in real-time. On mobile platforms, especially, the QR scanning feature could be extremely useful in helping users verify codes found in restaurants, ads, bills, or messages. Lightweight wrappers could be developed to host the trained model offline for use in edge scenarios where internet connectivity is limited.

Another potential enhancement involves extending the classification model to explain its predictions. Currently, the system offers a binary output—phishing or legitimate—but users may benefit from seeing a breakdown of key features that led to the classification. This would not only help with trust and transparency but could also serve as an educational tool, especially for students or employees learning about cybersecurity.

There is also room to explore deep learning models for phishing detection. While classical machine learning approaches were chosen for their speed and interpretability, models such as LSTM or BERT-based classifiers could be evaluated in future work, especially for analyzing URLs in combination with page content or metadata.

In summary, the project succeeded in meeting its objectives of developing a flexible and effective phishing detection system using machine learning. The two-phase structure enabled the detection of threats from both standard URLs and QR codes—

addressing a critical and timely need in cybersecurity. The system is fast, accurate, and designed for further evolution, making it a valuable starting point for future work in phishing defense systems. By combining technical.strength with practical usability.

REFERENCES

1. Zhang, Z., Liu, Y., Zhang, X., & Zhang, H. (2020). An Efficient Email Spam Filtering Method Based on Deep Learning. IEEE Access, 8, 182776-182785.
2. Huang, K., Huang, M., Guo, L., & Gao, J. (2020). Deep Learning for Efficient Spam Detection: A Comparative Study. In Proceedings of the 2020 IEEE International Conference on Big Data (pp. 2588-2593).
3. Ramachandran, G., & Pimple, S. (2020). Efficient Phishing Detection Using Deep Learning Techniques. In Proceedings of the 2020 International Conference on Electronics and Sustainable Communication Systems (pp. 1671-1675).
4. Mamun, M. A., Zeadally, S., & Doss, R. (2019). Deep Learning-Based Phishing Detection Techniques: A Comprehensive Survey. IEEE Access, 7, 73050-73071.
5. Yadav, S., Bansal, R., & Saini, A. K. (2018). Deep Learning Techniques for Phishing Detection and Classification. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA) (pp. 1-6).
6. Sinha, R., & Mohan, V. (2018). Efficient Email Spam Detection Using Deep Learning Techniques. In Proceedings of the 2018 International Conference on Communication and Signal Processing (ICCSP) (pp. 1516-1520).
7. Ayyadevara, V. S. S., & Kumar, A. A. (2017). Efficient Email Spam Classification Using Deep Learning Techniques. In Proceedings of the 2017 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-6).
8. Marinho, T., & Santos, R. (2017). Detecting Phishing Websites Using Deep Learning. In Proceedings of the 2017 IEEE/ACM 25th International Conference on Program Comprehension (pp. 313-314).