PhishGuard AI: A Hybrid Machine-Learning Approach for SMS Phishing and Spam Risk Detection

Mr. Dinesh kumar[1], Ms. Deiva thirumagal S[2], Mr. Maheer ahamed R[3]
[1]*Assistant Professor, Department of Computer Science and Engineering (Cyber Security), Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India*
[2,3]*Student, Department of Computer Science and Engineering (Cyber Security), Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India*

***Abstract*—Phishing and spam messages often use urgency, account-related claims, and suspicious links to persuade recipients to disclose information or take unsafe actions. This paper presents PhishGuard AI, a prototype message-screening system that combines a text-classification baseline with transparent rule-based indicators to communicate a message label and risk level. The machine-learning component uses Term Frequency–Inverse Document Frequency (TF–IDF) features and Logistic R egression trained on the UCI SMS Spam Collection. The dataset contains 5,572 labeled messages and is evaluated using a stratified 80:20 train–test split. The recorded baseline test accuracy is 97.58% on this SMS spam dataset; this result should not be interpreted as phishing-specific accuracy or as evidence of protection against all malicious URLs. A Flask API exposes prediction functionality to a React/Vite interface. Additional heuristic checks inspect message wording and URL patterns, and a configurable score maps findings to Low, Medium, High, or Critical risk bands. The prototype also includes scan history, dashboard summaries, and user-facing alerts. The approach is intended as a practical decision-support demonstration, with limitations around dataset domain, heuristic coverage, and the absence of live URL reputation services.**

***Index Terms*—Cybersecurity, Logistic Regression, phishing detection, SMS classification, TF–IDF, risk scoring.**

I. INTRODUCTION

Electronic messaging is routinely used for personal, academic, and financial communication. Attackers exploit this familiarity by sending messages that imitate trusted organizations, create a sense of urgency, or direct recipients to deceptive web pages. Such messages can lead to credential theft, financial loss, and unauthorized disclosure of personal information. A useful screening tool should therefore provide an understandable warning while avoiding claims that a prediction alone proves a message is malicious.

PhishGuard AI is a prototype that classifies message text and supplements the model output with simple heuristic checks. The system is designed as an accessible demonstration: users submit a message, the server processes it, and the interface presents a predicted class, confidence information, and a risk band. Its goal is to combine a reproducible machine-learning baseline with understandable indicators rather than to claim a production-grade security gateway.

The principal contributions are: (1) integration of TF–IDF and Logistic Regression for message classification; (2) a Flask prediction API connected to a React/Vite interface; (3) a rule-based risk indicator for suspicious wording and URL patterns; and (4) a dashboard and local scan history intended to support demonstration and review.

II. RELATED WORK

Text classification is a common baseline for filtering unwanted messages. TF–IDF represents documents using term weights that reflect how informative a word is within a document collection, while linear classifiers such as Logistic Regression provide a computationally efficient decision boundary. Public corpora, including the UCI SMS Spam Collection, support repeatable experiments on labeled SMS spam and legitimate messages [1].

However, SMS spam detection and phishing detection are related but distinct tasks. A message dataset

labeled as ham or spam does not necessarily provide verified phishing labels, nor does text classification alone establish whether a destination website is safe. Phishing detection systems may use additional URL, domain, page-content, or reputation signals [4]. The current prototype uses limited pattern-based URL and language checks; it does not query live domain registration, DNS, browser reputation, or commercial threat-intelligence services.

III. SYSTEM DESIGN AND METHODOLOGY

A. System architecture

The front end is implemented with React and Vite. Users enter or paste message text through the interface, which sends a request to the Flask backend. The backend applies the trained text vectorizer and classifier, then combines the prediction with heuristic indicators to generate a risk score and explanatory result. The front end displays the response and can store scan history in browser local storage. Dashboard values are derived from the locally available history rather than from a centralized analytics service.

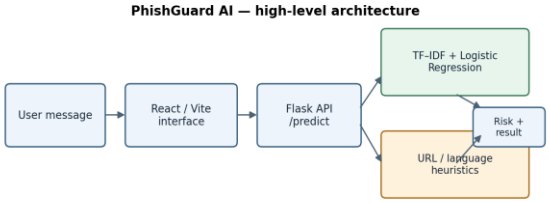

Fig. 1. High-level architecture of the PhishGuard AI prototype.

B. Dataset and preprocessing

The baseline model uses the UCI SMS Spam Collection, which contains 5,572 messages: 4,825 labeled ham and 747 labeled spams [1]. A stratified 80:20 split gives 4,457 training examples and 1,115 test examples, preserving the approximate class distribution in both subsets. Text is converted into TF–IDF features before classification. The split is used to estimate generalization on held-out examples from the same dataset; it does not measure performance on a separate phishing corpus or on live messages from current campaigns.

C. TF–IDF representation and classification

For a term t in document d, TF–IDF weights term frequency by inverse document frequency. In simplified form, the weight is TF (t, d) × IDF(t), where terms frequent in a document but uncommon across the corpus receive greater emphasis. Logistic Regression estimates class probabilities from the feature vector and predicts one of the dataset classes, ham or spam. In the application, the class label is presented as a classification result; confidence values should be interpreted as model outputs rather than calibrated guarantees of safety.

D. Heuristic risk indicators

A second layer checks for selected patterns associated with social engineering, including urgency-related wording, requests to verify or restore an account, and URL patterns such as shortened links or unusual link formatting. These checks are transparent rules, not a full URL reputation analysis. The risk score combines available indicators into a 0–100 scale. The configured bands are Low (0–29), Medium (30–59), High (60–79), and Critical (80–100). A high score signals that the message deserves caution; it is not proof of malicious intent.

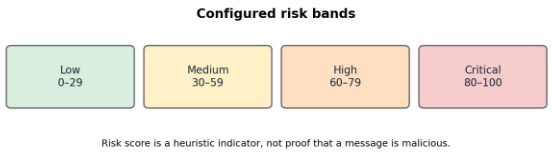

Fig. 2. Configured risk bands used by the prototype.

E. Request and response workflow

A user submits message text in the browser. The client sends a POST request to the /predict endpoint. The server validates the request, transforms the text using the saved vectorizer, obtains a class prediction, evaluates the heuristic indicators, and returns a structured result. The interface displays the predicted label, confidence information, risk score, and risk band. Scan-history and dashboard functions are intended for local demonstration and do not imply that the system monitors a phone inbox in the background.

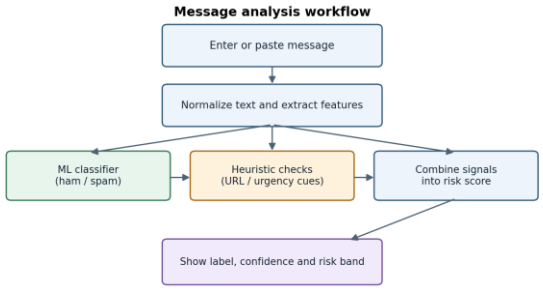

Fig. 3. Message screening workflow from submission to displayed result.

IV. IMPLEMENTATION

A. Backend service

The backend is implemented in Python using Flask. The root endpoint provides a basic service response, while POST /predict accepts message content and returns the model and risk-analysis result. The trained classifier and vectorizer are loaded by the application. Input validation and predictable response structure are important so the front end can handle empty or malformed submissions without confusing them with a safe prediction.

B. Web interface

The React/Vite interface provides a landing page, scan page, dashboard, history view, informational page, and login/sign-up screens. The scan page presents the classification and risk result. History search and filtering operate on locally stored scans, and dashboard summaries are calculated from that history. These account screens are part of the interface prototype; unless connected to a secure authentication service, they should not be described as production identity management.

C. Alerts and demo mode

The interface can show warnings when a result is classified as suspicious or receives a high or critical risk score. The browser-based “real-time demo” is a simulation of message submission and analysis. It does not intercept SMS messages on a user’s phone. The Android source-level foundation may support future mobile integration, but an APK build and device-level testing have not been verified as part of the reported evaluation.

V. RESULTS AND DISCUSSION

A. Baseline evaluation

On the held-out test portion of the UCI SMS Spam Collection, the recorded baseline accuracy is 97.58%. Accuracy is the fraction of correctly classified examples among all evaluated examples. This value is reported only for the SMS ham/spam classification task under the stated split. Precision, recall, F1-score, class-wise confusion counts, and evaluation on an independent phishing dataset are not reported here because verified measurements for those quantities are not available in the current project record.

Table I. Summary of baseline configuration

Dataset	UCI SMS Spam Collection
Total messages	5,572 (4,825 ham; 747 spam)
Train / test split	4,457 / 1,115 (stratified 80:20)
Feature representation	TF–IDF
Classifier	Logistic Regression
Recorded test accuracy	97.58% (SMS ham/spam baseline)

B. Illustrative examples

A routine message such as “Hey, are we still meeting for lunch today?” is an example of legitimate text and is expected to receive a low-risk result in the current demonstration. A message claiming that a bank account has been suspended and urging the recipient to follow a shortened URL illustrates the type of wording that may trigger the model and heuristic checks. These examples are illustrative demonstrations, not additional test-set measurements.

C. Interpretation

The baseline suggests that conventional text features can separate many spam and legitimate SMS examples in the selected corpus. Nonetheless, accuracy alone can hide errors on the minority class, so future evaluation should include precision, recall, F1-score, a confusion matrix, and testing on newer messages. The risk layer provides additional explainability but may produce false positives when legitimate messages contain urgent wording or links. Conversely, sophisticated phishing messages may avoid the patterns currently checked.

VI. LIMITATIONS AND FUTURE ENHANCEMENTS

The dataset represents SMS spam and legitimate messages rather than a curated phishing-only dataset. Results may not transfer to email, social-media direct messages, multilingual content, image-based lures, or newly emerging campaigns. The heuristic layer uses patterns rather than live URL inspection and can be evaded or triggered incorrectly. No live WHOIS, DNS, Google Safe Browsing, blacklist, or domain-reputation lookup is currently claimed. Browser local storage is not a secure central audit store, and the web demo does not passively monitor device messages.

Future work should evaluate a dedicated phishing dataset, report class-wise metrics and confidence intervals, test temporal and cross-domain generalization, and perform adversarial testing. URL analysis could be enhanced through safe redirect handling and trusted reputation APIs, with appropriate privacy and network controls. Additional work should include multilingual text, model calibration, explainable evidence for each warning, secure account management, automated tests, and verified Android builds on emulator and physical devices.

VII. CONCLUSION

This paper presented PhishGuard AI, a message-screening prototype combining TF–IDF and Logistic Regression with rule-based indicators and a configurable risk score. A Flask API connects the classification pipeline to a React/Vite interface with scan history, dashboard summaries, and warnings. The recorded 97.58% accuracy applies to a held-out split of the UCI SMS Spam Collection and must not be interpreted as phishing-specific performance. The prototype is a useful educational and demonstration platform, while broader security claims require independent phishing datasets, detailed metrics, live URL analysis, and further testing.

ACKNOWLEDGMENT

The authors may acknowledge their project guide, department, institution, and any data or infrastructure support here. Replace this text with an accurate acknowledgment before submission; do not include individuals or funding sources unless they actually contributed.

REFERENCES

[1] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, “Contributions to the study of SMS spam filtering: New collection and results,” in *Proc. 11th ACM Symposium on Document Engineering (DocEng ’11)*, Mountain View, CA, USA, 2011, pp. 259–262.

[2] F. Pedregosa *et al*., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2019.

[4] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond blacklists: Learning to detect malicious web sites from suspicious URLs,” in *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, 2009, pp. 1245–1254.