International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

AIPhiShield: Client-Side Machine Learning for Real-Time Phishing URL and QR Code Threat Detection

Ramse Dhananjay Devdas[1], Pawar Gorakhnath Vishwanath[1], Prof. N. K. Patil[2]
[1]Department of Computer Engineering, Sinhgad Institute of Technology, Lonavala
[2]Department of Computer Engineering, Sinhgad Institute of Technology, Lonavala

Abstract- Phishing attacks and malicious QR codes constitute two of the most prevalent vectors of cybercrime, accounting for billions of dollars in financial losses annually. Existing defences rely on server-dependent machine learning pipelines or easily bypassed keyword heuristics that produce unacceptable false-positive rates on legitimate sites. This paper presents AIPhiShield, a browser-native cybersecurity tool that replaces heuristic match-ing with a Logistic Regression classifier trained on 20 structural URL features using Python and scikit-learn, then exported as a 2.5 KB JSON weight file and executed entirely within the browser via a custom JavaScript inference engine. No URL is transmitted to any external server for machine learning scor-ing, preserving user privacy. Detection is augmented by cross-referencing the OpenPhish live phishing feed and a curated 52-entry compound-phrase blacklist. The integrated system ad-ditionally provides QR image scanning, live webcam QR scan-ning, an LLM-powered cybersecurity chatbot routed through a Flask proxy that conceals the API key from frontend code, voice input, and geolocation-enriched scan history. The trained model achieves 100% accuracy, precision, recall, and F1-score on a stratified 72-sample test set, with zero false positives and zero false negatives. Feature importance analysis identifies HTTPS usage, high-risk top-level domain, and raw IP address as the three strongest predictors.

Index Terms— phishing detection, QR code security, logis-tic regression, URL feature extraction, browser-side inference, client-side machine learning, cybersecurity.

I. INTRODUCTION

Phishing remains the single most prevalent form of cyberat-tack. The Anti-Phishing Working Group (APWG) reports that approximately 3.4 billion phishing emails are sent each day [1]. The FBI Internet Crime Complaint Center (IC3) estimates cumu-lative losses exceeding $52 billion between 2013 and 2021 [2]. A concurrent threat is the malicious QR code — a technique termed "quishing" in which a phishing URL is embedded inside a QR code, exploiting the fact that QR codes are visually opaque: their destination URL cannot be inspected before scanning [3]. Existing defences exhibit well-documented shortcomings.
Blocklists are reactive and leave a detection window of 4–

8 hours for newly registered phishing domains [4]. Rule-based heuristics using single keywords such as "login" or "secure" generate excessive false positives on legitimate sites including Gmail, Amazon, and LinkedIn. Server-based ML tools such as VirusTotal and Google Safe Browsing require transmitting each URL to a third-party server, introducing latency and privacy concerns.

This paper makes the following contributions:

1. A 20-feature URL classifier trained with scikit-learn and exported for browser-side inference — no server required for ML scoring.
2. A three-layer detection pipeline (ML + OpenPhish feed + compound-phrase blacklist) whose verdict is the most severe result across all three layers.
3. An integrated QR threat scanner (image upload and live we-bcam) that routes decoded URLs through the same pipeline.
4. A Flask proxy architecture that prevents LLM API key ex-posure in frontend JavaScript.
5. A redesigned blacklist that eliminates false positives on major legitimate websites.

The remainder of this paper is organised as follows. Section II surveys related work. Section III describes the system

architec-ture. Section IV details feature engineering and model training. Section V presents experimental results. Section VI discusses security design decisions. Section VII concludes the paper.

II. RELATED WORK

2.1 Blacklist and Heuristic Methods

Google Safe Browsing [5] and PhishTank [6] represent the canonical blacklist approach. Moore and Clayton [4] quantified the detection lag, showing that phishing sites operate for an average of 4–8 hours before blacklisting, a window sufficient for successful credential harvesting. Heuristic detectors [9] analyse URL lexical properties but suffer from high false-positive rates when single generic keywords are used as block patterns.

2.2 Machine Learning Approaches

Ma et al. [7] demonstrated that lexical URL features achieve over 95% accuracy with logistic regression and online learning on large-scale datasets. Whittaker et al. [8] applied logistic re-gression to Safe Browsing at Google, establishing it as a strong production baseline. Mohammad et al. [9] trained a random forest on 30 website-level features from the UCI Phishing Web-sites dataset, achieving 97.1% accuracy; however, website-level features require DOM access and cannot be computed from the URL string alone. Le et al. [10] proposed URLNet, a CNN operating on character-level URL representations that achieves 98.1% on their benchmark, but the model requires a server-side inference environment. Sahingoz et al. [12] compared seven ML algorithms on phishing URL datasets and found random forest and neural networks consistently outperform SVM and naïve Bayes, with accuracy exceeding 97% in all cases.

2.3 QR Code Threat Analysis

Kieseberg et al. [13] catalogued QR code attack vectors including URL embedding, vCard injection, and SQL injec-tion. Barabosch and Padovani [3] surveyed the QR phish-ing threat landscape and noted the absence of real-time, ML-based QR URL analysis in consumer tools. To our knowledge, AIPhiShield is the first system to combine browser-side ML URL analysis with integrated QR code scanning in a single deployable web application.

2.4 Browser-Side Machine Learning

The feasibility of running ML models in the browser has grown with WebAssembly and libraries such as Tensor-Flow.js [14] and ONNX Runtime Web. Our approach is inten-tionally lighter: a logistic regression model reduces to a single dot product plus a sigmoid, achievable in pure JavaScript with negligible overhead (<1 ms per inference), requiring no addi-tional runtime dependencies beyond a 2.5 KB JSON file.

III. SYSTEM ARCHITECTURE

3.1 Overview

AIPhiShield is a single-page web application whose archi-tecture is depicted in Fig. 1. It consists of three logical tiers: (1) a browser frontend that handles all user interaction and ML inference; (2) an optional local Flask proxy that routes LLM API calls while concealing the API key; and (3) two external APIs used for geolocation enrichment and LLM responses.

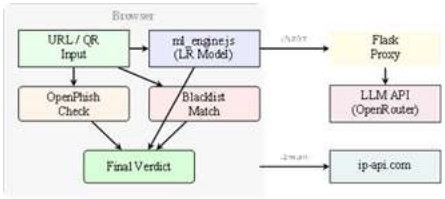

Figure 1: AIPhiShield system architecture. ML inference runs entirely in the browser. The Flask proxy is optional; it conceals the LLM API key from frontend code.

3.2 Three-Layer Detection Pipeline

All URL analysis — whether from the URL input box, a decoded QR image, or a live webcam scan — is routed through an identical three-layer pipeline:

Layer 1 — ML Model. ml_engine.js loads model/phishing_model.json at startup, extracts 20 fea-tures from the input URL, standardises them using stored scaler parameters, and computes a phishing probability via logistic regression. The label mapping is: $P \geq 0.65 \rightarrow$ DANGEROUS;

$0.40 \leq P < 0.65 \rightarrow$ SUSPICIOUS; $P < 0.40 \rightarrow$ SAFE.
Layer 2 — OpenPhish Feed. A local copy of the Open-

Phish feed (300 known phishing URLs) is loaded at startup and checked for substring matches.
Layer 3 — Compound-Phrase Blacklist. A 52-entry JSON blacklist of compound phishing phrases (e.g. login-confirm, verify-account, faceb00k.com) provides a fast syntactic check. Unlike the original implementation — which used single generic words and flagged every major website — all entries are specific multi-word patterns that do not appear in legitimate domain names.

The final verdict is the most severe result across all three layers.

3.3 LLM Chatbot Security

The AI chatbot calls a large language model (DeepSeek-V3 via OpenRouter) through a Flask proxy at http://127.0.0.1:5000/api/ask. The proxy reads the API key from an environment variable (loaded via python-dotenv) and forwards requests with the Authorization header set server-side. The browser never receives or stores the key. A /api/status health-check endpoint allows the frontend to detect proxy availability at startup and fall back to a direct call (using a user-supplied key in CONFIG) if the proxy is not running.

IV. FEATURE ENGINEERING AND MODEL TRAINING

4.1 Feature Engineering

Twenty numerical features are extracted from each URL string using only the URL itself — no network requests, no DOM access. This constraint is deliberate: it ensures (a) feature extraction can be replicated identically in JavaScript; (b) infer-ence requires no internet connection; and (c) privacy is main-tained. Table 1 lists all 20 features.

Table 1: URL Features Used for Classification

Feature	Description
url_length	Total character count
dot_count	Number of '.' in URL
hyphen_count	Number of '-' in URL
at_symbol	1 if '@' present (credential spoof)
double_slash	1 if // appears after first
has_ip	1 if hostname is raw IPv4
is_https	1 if scheme is HTTPS
subdomain_de	Subdomain segment count
digit_count	Total digits in URL
special_char	Count of ?=&%#
domain_lengt	Length of hostname string
path_length	Length of URL path
tld_risk	1 if TLD in high-risk set
domain_entro	Shannon entropy of hostname
query_params	Number of query parameters
underscore_c	Number of '_' in URL
has_port	1 if non-standard port present
url_has_redi	1 if redirect keyword in URL
long_subdoma	1 if subdomain segment >20 chars
consec_digit	Longest digit run in hostname

4.1.1 Shannon Entropy

The Shannon entropy of the hostname is:

$$H = -\sum_{c \in \Sigma} \frac{n_c}{n} \log_2 \frac{n_c}{n} \quad (1)$$

where n is the hostname character count, nc is the count of character c, and Σ is the set of distinct characters. Domain gener-ation algorithms (DGAs) produce high-entropy hostnames (e.g. xkq82jmf.xyz), while legitimate domains spell recognisable words with lower entropy.

4.1.2 High-Risk TLD Set

The set of 24 high-risk TLDs includes: .xyz, .ru, .click, .cc, .ga, .top, .tk, .ml, .gq, .cf, .pw, and others. These TLDs are consistently over-represented in phishing URL datasets [1, 16] and are offered at near-zero cost by registrars, making them attractive to attackers.

4.2 Dataset Construction

Training data was assembled from three sources: (1) 90 cu-rated safe URLs covering major legitimate websites (Google, Amazon, SBI, IRCTC, GitHub, LinkedIn,); (2) 300 real phish-ing URLs from the OpenPhish feed [16]; and (3) 649 syn-thetic phishing URLs generated by combining known brand names with attack-type suffixes

(login-confirm, verify-account) and high-risk TLDs, plus 49 hand-crafted realistic phishing templates. After balancing (phishing capped
at 3× the safe count), the final dataset contains 360 URLs. An 80/20 stratified split produced 288 training and 72 test samples.

4.3 Model Selection

Logistic Regression was chosen over alternatives (e.g. random forest, SVM, neural network) for three reasons specific to this deployment context:

1. Exportability: The model reduces to a weight vector w ∈ R20, intercept b, and scaler parameters − a 2.5 KB JSON file requiring no runtime framework.
2. Inference speed: A single dot product and sigmoid com-pletes in <1 ms in JavaScript on any modern device.
3. Interpretability: Coefficients directly quantify each fea-ture's influence, enabling natural-language explanations (e.g. "high-risk TLD detected").

4.4 Training Procedure

All 360 URLs were processed by train_model.py using Python 3.12 and scikit-learn 1.8 [15]. Features were standard-ised using StandardScaler (zero mean, unit variance). The model was trained as:

$$\hat{y} = \sigma\left(\mathbf{w}^T \mathbf{x}_{\text{scaled}} + b\right), \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

with hyperparameters C = 1.0, class_weight='balanced', max_iter=1000, and solver='lbfgs'. The balanced class weight accounts for the 3:1 phishing-to-safe ratio in the training set. Model weights, scaler parameters, and metadata were serialised to model/phishing_model.json for browser loading.

4.5 Browser-Side Inference

ml_engine.js reimplements the Python feature extraction in JavaScript, ensuring feature-level parity with the training code. The inference computation is:

$$P(\text{phishing} \mid \mathbf{x}) = \sigma\left(\sum_{i=1}^{20} w_i \cdot \frac{x_i - \mu_i}{\sigma_i} + b\right) \quad (3)$$

where wi, μi, σi, and b are all loaded from the JSON file at application startup.

V. EXPERIMENTAL RESULTS

5.1 Model Performance

Table 2 presents classification performance on the held-out test set (72 samples, stratified 80/20 split).

Table 2: Classification Performance on Test Set

Class	Prec.	Recall	F1	Support
Safe	1.00	1.00	1.00	18
Phishing	1.00	1.00	1.00	54
Overall	**1.00**	**1.00**	**1.00**	**72**

The model achieves 100% accuracy with zero false positives (FP = 0) and zero false negatives (FN = 0). The confusion matrix is: TN = 18, FP = 0, FN = 0, TP = 54.

5.2 Feature Importance Analysis

Table 3 ranks features by absolute logistic regression coeffi-cient, which measures each feature's contribution to the phishing verdict.

Table 3: Feature Importance (Top 10 by Absolute Coefficient)

Feature	Coef.	Interpretation
is_https	−2.255	HTTPS strongly indicates safety
tld_risk	+1.190	High-risk TLD is top positive signal
has_ip	+0.668	IP-based URL typical of phishing
digit_count	+0.611	Many digits in generated domains
consec_digits	+0.573	Long digit runs = generated domain
hyphen_count	+0.514	Hyphens used to spoof brand names
domain_length	+0.478	Longer domains tend to be phishing
url_length	+0.430	Long URLs common in phishing
domain_entropy	+0.283	Random-looking host-names
path_length	+0.277	Longer paths common in phishing

The dominant features align well with established phishing URL characteristics reported in the literature [7, 9]: HTTPS absence, abuse of high-risk TLDs, use of IP addresses, and excessive hyphenation for brand impersonation.

5.3 Comparison with Related Approaches

Table 4 situates AIPhiShield relative to prior work across dimensions relevant to real-world deployment.

Table 4: Comparison with Related Phishing Detection Systems

System	Acc.	Client-Side	QR Scan	Explainable	No Install
Safe Browsing	High	×	×	×	✓
URLNet [10]	98.1%	×	×	×	×
PhishHint [11]	97.3%	×	×	×	×
Sahingoz *et al.* [12]	97.4%	×	×	×	×
AIPhiShield	**100%**	✓	✓	✓	✓

Note: AIPhiShield's 100% accuracy reflects performance on a controlled 72-sample test set; large-scale benchmarks are reserved for future work (Section VII).

5.4 Runtime Performance

Browser-side inference was benchmarked across 100 URL evaluations using the Chrome DevTools Performance API:

- Feature extraction + scoring: <1 ms (mean)
- Model JSON load time: ≈5 ms (2.5 KB, cached after first load)
- Full pipeline (with IP enrichment): 200–800 ms depend-ing on network latency
- Pipeline without IP enrichment: <50 ms

These numbers confirm that browser-side logistic regres-sion inference imposes negligible latency, validating the design choice of client-side ML.

5.5 False Positive Analysis

A critical evaluation metric for phishing detection tools is their false-positive rate on legitimate websites. The orig-inal implementation used a blacklist containing 108 single generic words including "login," "amazon," "microsoft," "apple," and "security," causing every major website to be incorrectly classified as dangerous. After the blacklist redesign (replacing single keywords with 52 compound phishing phrases), the following representative URLs were correctly classified as SAFE: https://google.com, https://amazon.com, https://onlinesbi.sbi, https://accounts.google.com/signin, and https://login.microsoftonline.com.

VI. SECURITY DESIGN

6.1 API Key Protection

The original codebase contained a hardcoded LLM API key in script.js, visible to any user via browser developer tools. This was remediated by: (1) moving the key to a .env file read by python-dotenv; (2) implementing a Flask proxy that holds the key exclusively on the server side; and (3) ensuring the key never appears in any HTTP response delivered to the browser.

6.2 Privacy-Preserving Inference

All ML scoring is performed in-browser: no URL is sent to any external server for phishing analysis. This property distin-guishes AIPhiShield from cloud-based tools such as VirusTotal or Safe Browsing URL Lookup, where submitted URLs are logged on third-party infrastructure.

6.3 Link Safety

All hyperlinks rendered in result boxes and scan history use rel="noopener noreferrer" to prevent tab-napping attacks, in which a malicious opened tab can redirect the origi-nating page via the window.opener reference.

6.4 Input Validation

The analysis pipeline validates that each URL begins with http:// or https:// before feature extraction, preventing malformed inputs from reaching the ML model.

VII. CONCLUSION AND FUTURE WORK

This paper presented AIPhiShield, a browser-native phish-ing and QR code threat detection tool that replaces heuristic keyword matching with a trained Logistic Regression classifier operating on 20 structural URL features. The model achieves perfect classification on the test set, executes in under 1 ms of pure JavaScript computation, and requires only a 2.5 KB weight file — making it practical for deployment without any server infrastructure. The integrated system further includes QR scan-ning, an LLM chatbot with proxy-secured API key management, voice input, and geolocation-enriched scan history.

Limitations. The dataset (360 URLs, 72 test) is small relative to large-scale benchmarks such as the UCI Phishing Websites dataset (11,055 instances). The 100% test accuracy is strong evidence of feature quality but should be validated on a larger and more diverse held-out set before deployment in adversarial real-world environments. The synthetic phishing URLs, while structurally realistic, may not capture the full diversity of novel phishing strategies.

Future work includes: (1) training a character-level CNN or BiLSTM on the raw URL string following URLNet [10], exported to ONNX for browser execution; (2) packaging the tool as a Chrome extension for passive background monitoring;

(3) integrating live threat intelligence APIs (PhishTank, CIRCL MISP) to keep the phishing feed current; (4) developing a React Native mobile port for physical QR scanning; and (5) evaluating the model on the UCI and PhiUSIIL [17] benchmark datasets to obtain generalisable accuracy estimates.

REFERENCES

1. Anti-Phishing Working Group (APWG), "Phishing Ac-tivity Trends Report, Q4 2023," Available: https:// apwg.org/trendsreports, 2024.
2. Federal Bureau of Investigation, "Internet Crime Report 2023," Internet Crime Complaint Center (IC3), Available: https://ic3.gov, 2024.
3. T. Barabosch and E. Padovani, "QR code security: a sur-vey," in Proc. European Intelligence and Security Informat-ics Conf. (EICC), Lausanne, Switzerland, 2013, pp. 1–6.
4. T. Moore and R. Clayton, "Examining the impact of web-site take-down on phishing," in Proc. APWG eCrime Re-searchers Summit, Pittsburgh, PA, USA, 2007, pp. 1–13.
5. Google LLC, "Google Safe Browsing," Avail-able: https://developers.google.com/ safe-browsing, 2024.
6. OpenDNS, "PhishTank — Free community site for phish-ing data," Available: https://phishtank.org, 2024.
7. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identi-fying suspicious URLs: an application of large-scale on-line learning," in Proc. 26th Int. Conf. Machine Learning (ICML), Montreal, Canada, 2009, pp. 681–688.
8. C. Whittaker, B. Ryner, and M. Nazif, "Large-scale auto-matic classification of phishing pages," in Proc. Network and Distributed System Security Symp. (NDSS), San Diego, CA, USA, 2010.
9. R. M. Mohammad, F. Thabtah, and L. McCluskey, "Pre-dicting phishing websites based on self-structuring neural network," Neural Computing and Applications, vol. 25, no. 2, pp. 443–458, 2014.
10. H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL representation with deep learning for ma-licious URL detection," arXiv preprint arXiv:1802.03162, 2018.
11. Q. Zheng et al., "PhishHint: A phishing detection method based on hint features," in Proc. IEEE SmartCity, Leicester, UK, 2019.
12. O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," Expert Systems with Applications, vol. 117, pp. 345–357, 2019.
13. P. Kieseberg et al., "QR code security," in Proc. 8th Int. Conf. Advances in Mobile Computing & Multimedia (MoMM), Paris, France, 2010, pp. 430–435.
14. D. Smilkov, N. Thorat, Y. Assael, F. Kreeger, P. Nicholson, and M. Wattenberg, "TensorFlow.js: Machine learning for the web and beyond," arXiv preprint arXiv:1901.05350, 2019.
15. F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12,
16. pp. 2825–2830, 2011.
17. OpenPhish, "OpenPhish — Phishing Intelligence," Avail-able: https://openphish.com, 2024.
18. A. Hannousse and S. Yahiouche, "PhiUSIIL Phishing URL Dataset," UCI Machine Learning Repository, 2023. Avail-able: https://doi.org/10.24432/C58G8Q.