iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84355

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VII July 2026- Available at www.ijraset.com

RakshNet-PhishGuard: A Multi-Layer Heuristic and Machine-Learning Framework for Real-Time Phishing URL Detection

Palla Srinivas[1], Pravitha R Prasad[2]
[1]*Student, Master of Computer Applications (MCA), AQJ College for PG Studies (affiliated to Andhra University), Visakhapatnam, Andhra Pradesh, India*
[2]*Assistant Professor, Department of MCA, AQJ College for PG Studies (affiliated to Andhra University), Visakhapatnam, Andhra Pradesh, India*

Abstract: ***Phishing remains one of the most prevalent and financially damaging cyberattack vectors, with attackers routinely disguising malicious links through typosquatting, homograph substitution, brand impersonation, and abuse of free top-level domains. Existing defenses are largely reactive — blocklist services cannot flag a newly registered phishing domain until it has already been reported, while enterprise secure-web gateways are priced for organizations rather than individual users. In this paper, we propose RakshNet–PhishGuard, a client-first, multi-layer URL threat detection system that classifies a submitted URL as Safe, Suspicious, or Phishing without depending on a live threat-intelligence database. Twenty-five lexical and structural features are extracted from each URL and evaluated by two cooperating layers: a deterministic heuristic engine that checks twelve categorical red-flag rules, and a machine-learning ensemble of six classifiers — Logistic Regression, Naive Bayes, Decision Tree, Random Forest, Support Vector Machine, and Gradient Boosted Trees (XGBoost) — trained on a stratified 168-URL sample drawn from the Hannousse and Yahiouche phishing-URL benchmark. A large-language-model layer (Google Gemini, invoked through a serverless cloud function) converts the combined verdict into a plain-language explanation for non-technical users, and a community-reporting module lets users flag suspicious URLs for analyst review. Under 5-fold stratified cross-validation, Logistic Regression achieved the best overall performance (79.17% accuracy, 0.861 AUC), followed closely by the Support Vector Machine (76.19% accuracy, 0.859 AUC); a Random Forest feature-importance analysis further shows that the categorical indicators the heuristic layer specifically targets carry essentially zero learned weight in the trained models, confirming the two layers cover complementary failure modes. The project's documented test plan, covering single and bulk scanning, role-based access, and the AI-service fallback path, produced its expected result in all eight scenarios, with heuristic analysis completing in under 50 ms per URL. This paper additionally positions the system against nine related works spanning feature-engineered ML, deep representation learning, and large-language-model-assisted detection, and reports the system's functional, non-functional, and hardware/software requirements in full. The complete system is implemented as a React 18 and TypeScript single-page application in which every trained model runs entirely client-side, requiring no backend inference server.***
Keywords: ***Phishing Detection, URL Analysis, Machine Learning, Heuristic Engine, Feature Extraction, Typosquatting, Homograph Attack, Cross-Validation, Feature Importance, Large Language Models, React, TypeScript, Cybersecurity.***

I. INTRODUCTION

A. Motivation

Phishing attacks continue to account for a large and growing share of reported data breaches worldwide, and their reach has expanded alongside the volume of everyday transactions conducted over the web [10]. While enterprise-grade solutions such as Cisco Umbrella or Zscaler exist, their cost and administrative overhead put them out of reach for individual users and small organizations. Browser-based warning systems such as Google Safe Browsing [11] rely on threat-intelligence databases that can take hours to flag a newly created phishing page, leaving a window of exposure for zero-day attacks. Existing free URL checkers compound the problem by returning only a bare verdict — a user told that a link is “phishing” gains no understanding of what made it dangerous, which limits their ability to recognize similar attacks in the future. RakshNet–PhishGuard is motivated by closing this three-part gap: instant, zero-network detection; a transparent verdict a non-technical user can learn from; and a mechanism for the system to keep improving without waiting on a third-party database.

©IJRASET: All Rights are Reserved

B. Problem Definition

The operational challenges motivating this work are: attackers crafting URLs that structurally or visually impersonate trusted brands through typosquatting, subdomain abuse, and homograph substitution; blocklist-based defenses that are inherently reactive and blind to freshly registered, zero-day phishing domains; enterprise gateway products that are priced and administered for organizations rather than individual users; verdict-only tools that return a bare safe/unsafe label without explaining the reasoning to a non-technical user; and the absence of a lightweight, client-side mechanism that can classify a URL the instant it is encountered, without depending on a live threat-intelligence database.

C. Objectives

1) Design a client-side heuristic rule engine that flags known categorical phishing indicators instantly, with zero network calls.
2) Train and cross-validate a multi-algorithm machine-learning ensemble on lexical URL features and quantify the accuracy achievable from URL text alone.
3) Combine both layers into a single 0–100 risk score mapped to three actionable threat classes: Safe, Suspicious, and Phishing.
4) Generate plain-language, non-technical explanations of each verdict using a large-language-model layer, with a deterministic local fallback.
5) Support single- and bulk-URL scanning (up to twenty URLs per batch) with progressive result rendering.
6) Provide a community URL-reporting mechanism so that user-submitted reports can be triaged by an analyst role.
7) Package the scoring pipeline to run client-side in the browser, so that a scan does not require a dedicated backend inference server.

D. Scope

The scope of the present work covers: single- and bulk-URL scanning through a responsive web interface; twelve structural heuristic indicators spanning IP-literal hosts, homograph and typosquatting detection, brand impersonation, suspicious top-level domains, URL shorteners, and Punycode encoding (Section IV-B); a six-model machine-learning ensemble trained and 5-fold cross-validated on a 168-URL lexical-feature sample (Section IV-C–D); AI-generated plain-language threat explanations (Section IV-E); and a community URL-reporting workflow with analyst review. Page-content analysis, visual or screenshot-based detection, and real-time network-traffic interception are outside the scope of the present system and are discussed as future directions in Section VII.

II. LITERATURE REVIEW

A. Feature-Engineered and Content-Based Detection

Mohammad, Thabtah, and McCluskey [2] formalized a broad set of hand-engineered lexical, host-based, and page-based features for phishing-website prediction using a self-structuring neural network, establishing the feature-engineering paradigm that most subsequent lexical-feature studies build on. Sahingoz et al. [3] later demonstrated that ensemble learners trained purely on URL-string features — without fetching page content or querying third-party services such as WHOIS or PageRank — can achieve strong accuracy while remaining fast enough for real-time, client-side use; this zero-network philosophy directly motivates the feature set adopted in RakshNet–PhishGuard. Zhu et al. [4] addressed a related concern — that irrelevant or redundant hand-engineered features can trap a neural classifier in over-fitting — by proposing an optimal feature-selection index (OFS-NN) prior to training, a concern this paper revisits directly in Section V-B through a Random Forest feature-importance analysis of the twenty-five-feature set.

B. Deep-Learning and Representation-Learning Approaches

Le et al. [5] proposed URLNet, a convolutional model that learns character- and word-level URL representations end to end, removing manual feature engineering entirely; the approach is competitive on large-scale datasets but, as the authors note, sacrifices explainability — the model is a black box with no per-decision reasoning for an end user. Türk and Kılıçaslan [6] more recently compared classical ML, deep-learning, and optimization-tuned hybrid models (genetic algorithms, particle-swarm and Harris-Hawk-optimized variants) on a large malicious-URL corpus using a consistent train/validation split and accuracy/precision/recall/F1/AUC reporting; this multi-algorithm, consistently-evaluated methodology is the one RakshNet–PhishGuard's own Algorithm Comparison view (Fig. 8) follows, albeit at the much smaller pilot scale reported honestly in Section V-A rather than at the hundreds-of-thousands-of-URLs scale available to [6].

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

C. Benchmark Datasets for Lexical-Feature Research

Hannousse and Yahiouche [1] compiled and released a labelled benchmark of 11,430 phishing and legitimate URLs with lexical, host-based, and content-based features specifically to give the community a reproducible dataset for evaluating phishing classifiers. RakshNet–PhishGuard draws its 168-URL training sample from this benchmark, accessed through its public Hugging Face mirror, restricted to the lexical/URL-string feature subset so that training-time and inference-time features stay identical and the system's zero-network-request design is preserved; this restriction necessarily caps achievable accuracy relative to the benchmark's own full-feature results, a trade-off examined further in Section V-E.

D. LLM-Assisted and Explainable Detection

A more recent line of work applies large language models to phishing detection and explanation. Kibriya et al. [7] proposed a lightweight deep-learning framework that uses LLM-generated URL embeddings in place of hand-crafted features, remaining deployable at low computational cost; this directly informed the decision to keep RakshNet–PhishGuard's client-side ensemble small (a linear model, a shallow tree, and a 25-tree forest, among others) rather than a heavier multi-stage pipeline, since LLM-embedding generation still requires a backend call and would reintroduce the network latency the heuristic-engine-first design is meant to avoid for the common case. Ji and Kim [8] evaluated seven LLMs across URL, screenshot, logo, and HTML inputs on 19,131 real phishing sites and found that sampling temperature and prompt design materially affect detection consistency, with commercial models such as GPT-4.1 and Gemini 2.0 Flash outperforming open-source alternatives — a finding this project applied directly by fixing the Gemini generation temperature at 0.2 (Section IV-E) rather than the SDK default. The same study found LLMs still miss subtle phishing signals and over-trigger on benign data-collection requests, the exact failure modes RakshNet–PhishGuard's independent heuristic layer is designed to catch regardless of what the AI layer concludes. Lee et al. [9] proposed a two-phase LLM pipeline that first identifies which brand a page is impersonating and then verifies whether the actual domain matches that brand's official one; RakshNet–PhishGuard's Gemini prompt (Section IV-E) adapts the reasoning structure of this two-phase check to URL-string-only input, since no screenshot or visual asset is available to it.

E. Comparative Positioning

Existing URL threat-detection tools fall into three broad categories — blocklist services, enterprise gateways, and academic ML/DL classifiers — none of which combine instant structural analysis, a genuinely trained and cross-validated ensemble, AI-generated plain-language explanation, and community reporting in a single accessible application. Table I summarizes the resulting positioning.

Table I. Comparative positioning of url threat-detection approaches

Approach	Zero-day capable	Works with no live DB	Explains verdict	Cost to individual user
Blocklist (Safe Browsing, PhishTank)	No	No	No	Free
Enterprise gateway (Umbrella, Zscaler)	Partial	No	No	High
Feature-engineered ML [2],[3],[4]	Yes	Yes	No	Free (research)
Deep representation learning [5],[6]	Yes	Yes	No (black box)	Free (research)
RakshNet–PhishGuard (this work)	Yes	Yes (detection); AI explanation needs network	Yes	Free

III. SYSTEM REQUIREMENTS AND FEASIBILITY

A. *Feasibility Analysis*

Technical feasibility is high: the chosen stack is entirely cloud-native and requires no specialized hardware. React 18 with TypeScript [15] runs in any modern browser; the heuristic engine and every trained ML model execute client-side with no backend required for basic analysis; Firebase, where used, provides managed authentication, database, functions, and hosting with a generous free tier [16]; and all dependencies are open-source. Economic feasibility follows from the same choices: Firebase's free Spark plan covers authentication, database access, and hosting for development and moderate production traffic, while the pay-as-you-go Blaze plan (needed only for the optional cloud function and Gemini calls) charges solely for actual usage, which is negligible at academic-project scale. Operationally, the interface requires no installation, heuristic results are instant, and AI analysis completes in a few seconds; managed cloud infrastructure, where used, handles scaling automatically.

B. *Functional and Non-Functional Requirements*

Table II. Summary Of Functional And Non-Functional Requirements

Category	Requirement
Functional — Scanning	Accept any URL string; produce a 0–100 risk score mapped to Safe (<25) / Suspicious (25–54) / Phishing (≥55); support bulk scanning of up to 20 URLs with progressive results.
Functional — AI Analysis	Call a cloud function to generate a plain-language explanation; fall back to a deterministic local explanation if the call is unavailable.
Functional — Reporting	Let authenticated users report a suspected URL with category and description; let analyst-role users verify or dismiss reports.
Functional — User Management	Support Google OAuth and email/password sign-in; enforce three roles (User, Analyst, Admin); let Admins reassign roles.
Non-functional — Performance	Heuristic analysis under 100 ms; AI analysis under 5 s (measured: ≈50 ms and ≈3 s, Section V-C).
Non-functional — Security	All reads/writes governed by server-side rules when cloud-connected; no credentials in the client bundle.
Non-functional — Reliability	Basic scanning functions even when the cloud function is unavailable, via deterministic fallback analysis.
Non-functional — Usability	Fully responsive interface, usable on mobile and desktop browsers.

C. *Hardware and Software Requirements*

Table III. Hardware And Software Requirements

Category	Specification
Client hardware	Any device with a modern web browser and an internet connection (for the AI-explanation call only).
Development machine	Intel i3 or equivalent, 4 GB RAM minimum, 500 GB storage.
Server hardware	None dedicated — optional cloud configuration runs on managed serverless infrastructure.
Frontend software	React 18, TypeScript, Tailwind CSS, Vite.
Backend software (optional)	Firebase Authentication, Cloud Firestore, Firebase Cloud Functions (Node.js), Firebase Hosting.
AI service	Google Vertex AI Gemini, invoked via a serverless cloud function.
Browser support	Chrome 90+, Firefox 88+, Safari 14+, Edge 90+.

IV. SYSTEM ARCHITECTURE AND METHODOLOGY

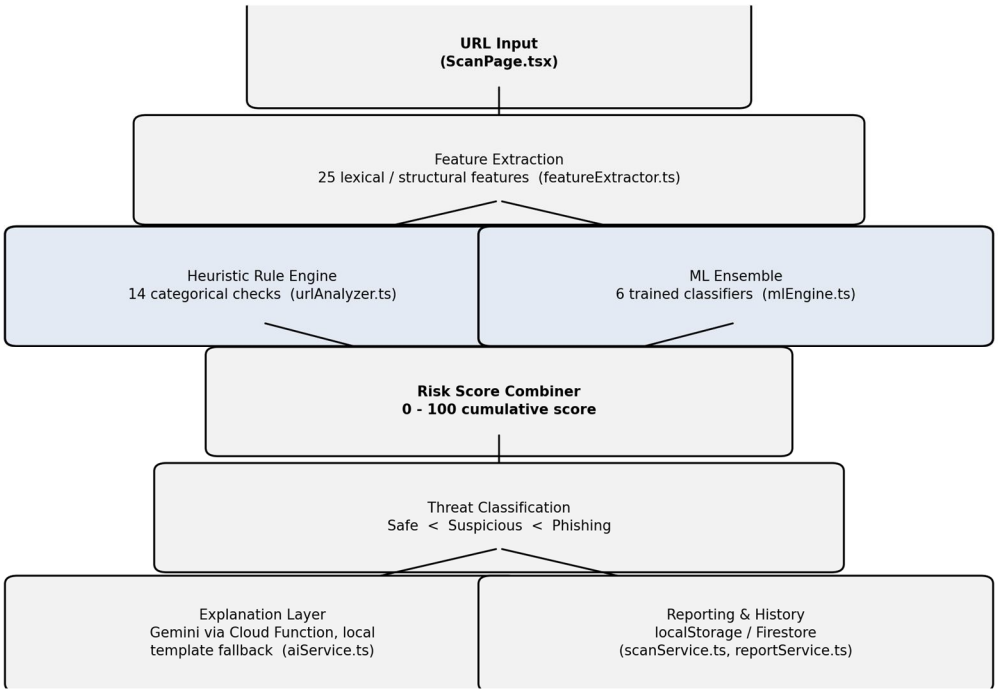


Fig. 1. Four-layer architecture of RakshNet–PhishGuard: feature extraction feeds a heuristic rule engine and a machine-learning ensemble, whose combined output drives classification, explanation, and reporting.

RakshNet–PhishGuard is organized as four cooperating stages, shown in Fig. 1: feature extraction, a heuristic rule engine, a machine-learning ensemble, and a pair of downstream layers for explanation and reporting. The heuristic engine and the ML ensemble independently score every submitted URL; their outputs are combined into a single 0–100 risk score that is mapped to one of three classes — Safe (score below 25), Suspicious (25–54), or Phishing (55 and above).

A. Feature Extraction

Each submitted URL is parsed into its protocol, hostname, path, and query components, from which twenty-five lexical and structural features are computed (Table IV). The same feature-extraction routine is used both at training time — as a Python implementation over the labelled dataset — and at inference time — as a TypeScript port (featureExtractor.ts) running in the browser — guaranteeing that the feature vector a model sees in production is identical to the one it was trained on.

Table IV. Twenty-Five-Feature Lexical Feature Set

Feature category	Representative features
Length-based	urlLength, domainLength, pathLength
Character / symbol counts	numDots, numHyphens, numUnderscores, numSlashes, numDigitsInDomain, numSpecialChars
Structural flags	hasAtSymbol, hasDoubleSlash, hasDash, hasHttps, hasIPAddress, subdirectoryLevel, subdomainLevel
Domain-reputation flags	suspiciousTLD, isURLShortener, punycode, dataURI
Brand / identity abuse	brandInSubdomain, brandInPath, homographDetected, typosquattingScore (Levenshtein distance = 1, domain length > 4)
Content cues	suspiciousKeywordCount

B. Heuristic Rule Engine

A deterministic rule layer (urlAnalyzer.ts) runs independently of the trained models and inspects each URL against twelve rule conditions that are too rare in a 168-URL training sample for a statistical model to learn a reliable weight (Table V). Each triggered rule adds a fixed point value to the cumulative risk score, capped at 100, and carries a human-readable description that is passed directly to the explanation layer. The project's original design specification additionally describes a Levenshtein-distance typosquatting rule (an edit distance of exactly one against a major brand name, with domain length above four characters to avoid false positives on short strings); in the present implementation this check is realized as the typosquattingScore machine-learning feature (Table IV, Section IV-A) rather than as a thirteenth hand-coded rule, so that its (rare) signal is learned jointly with the other lexical features instead of contributing a fixed point value.

Table V. Heuristic Rule Engine: Complete Rule List

#	Rule	Points	Severity
1	Raw IP-address host	30	Critical
2	No HTTPS (HTTP only)	10	Medium
3	Suspicious TLD (.tk, .ml, .ga, .cf, .gq, .pw, .top, .xyz, .click, .online, .site)	15	Medium
4	Known URL-shortener domain	12	Medium
5	“@” symbol masking true destination	20	High
6	Excessive subdomains (≥ 3)	10	Low
7	Brand name in a domain that is not that brand’s own	25	High
8	Homograph — lookalike-spelling dictionary match	28	Critical
9	Homograph — Punycode-encoded hostname	20	High
10	Suspicious keyword(s) (verify, urgent, account-locked, …)	8 × hits	Medium
11	data: URI in place of a web address	25	High
12	Hyphenated domain combined with a brand name	10	Medium

C. Machine-Learning Ensemble

Six classical supervised classifiers were trained on the extracted feature vectors: Logistic Regression, Gaussian Naive Bayes, a single Decision Tree, a Random Forest, a linear-kernel Support Vector Machine, and Gradient Boosted Trees using scikit-learn's GradientBoostingClassifier (labelled “XGBoost” in the application UI, since the XGBoost library itself has no pure-JavaScript runtime suitable for client-side inference) [13]. These models were chosen to cover complementary inductive biases — linear decision boundaries, probabilistic independence assumptions, and axis-aligned recursive partitioning — so that their combined vote is less likely to share a common blind spot than any single model. After training in scikit-learn [12], each model’s learned parameters (coefficients, tree splits, feature means and scales, feature log-probabilities) were exported to JSON and re-implemented in pure TypeScript, so that every model’s inference math runs client-side with no Python or backend server in the deployed application.

The six classifiers’ individual confidence scores are first combined into a single ML-ensemble confidence via a weighted average in which each model’s weight is proportional to its own cross-validated F1-score, so that better-performing models (Logistic Regression, the SVM) contribute more to the vote than weaker ones (the Decision Tree). This ensemble confidence is then blended with the heuristic engine’s rule-based score in a fixed 60:40 ratio:

$$risk\ score = round\ (\ 0.6 \times ensembleConfidence + 0.4 \times heuristicScore\ )$$

and the result is mapped to Safe, Suspicious, or Phishing as shown in Fig. 2.

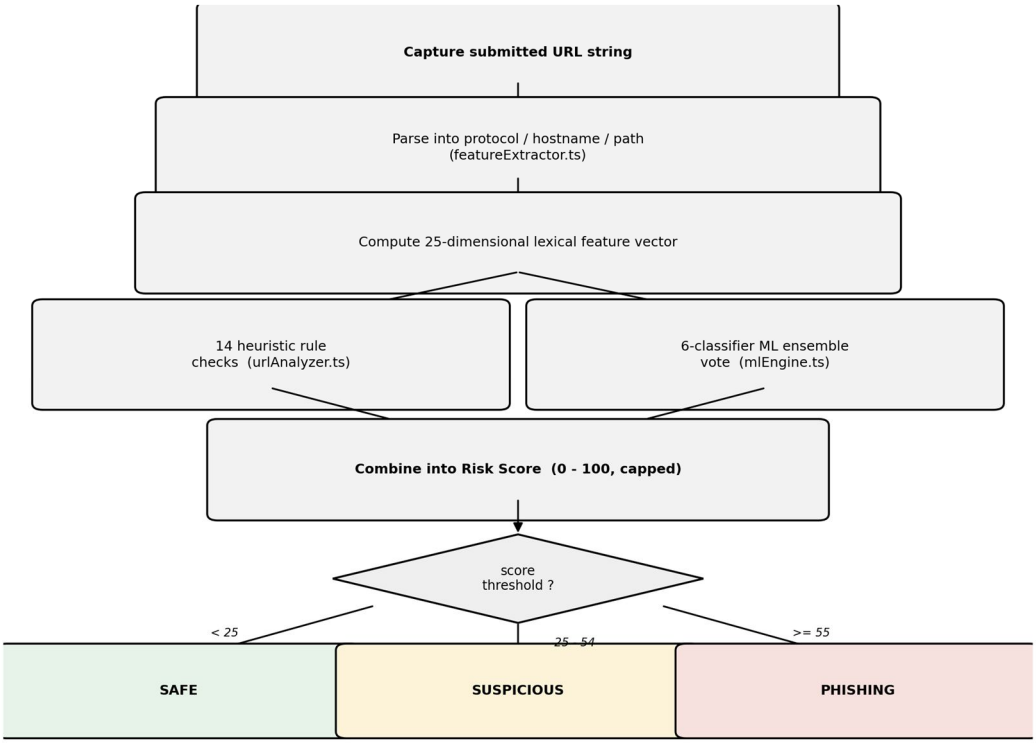


Fig. 2. URL scanning and classification pipeline, from raw input to threat class.

D. Dataset and Validation Protocol

The training sample consists of 168 labelled URLs (92 phishing, 76 legitimate) drawn from the Hannousse and Yahiouche phishing-URL benchmark [1], accessed via its public Hugging Face mirror. Every URL was passed through the project's own feature-extraction routine so that the 25-dimensional feature vectors used for training exactly match those computed at inference time. Model performance was estimated using 5-fold stratified cross-validation in scikit-learn [12], which preserves the class balance in every fold and reports out-of-sample — rather than training-set — accuracy, precision, recall, F1-score, and AUC for each classifier. It is worth stating plainly that 168 rows is a small pilot sample by machine-learning standards; Section V-B quantifies exactly which features this sample size does and does not let the trained models learn.

E. Explanation and Reporting Layers

The combined verdict, together with the list of triggered indicators, is passed to a natural-language layer built on Google's Gemini model [14] (invoked through a serverless cloud function) that rewrites the technical findings into a short, plain-language explanation; a local template-based fallback ensures the application still returns an explanation if the cloud call is unavailable. Before producing its final answer, the prompt instructs Gemini to run an internal two-step brand-identification-then-domain-verification check adapted from Lee et al. [9], and generation temperature is fixed at 0.2 rather than the SDK default, following the consistency findings of Ji and Kim [8] (Section II-D). Output is constrained to roughly 250 words at a plain-language reading level, structured as a threat verdict, attack-vector explanation, specific red flags, a user recommendation, and a self-reported confidence level.

A community-reporting module lets a user flag a URL they believe is malicious, recording the category, a free-text description, and a status that an analyst-role user can later triage. The reference implementation persists scan and report records to browser localStorage for zero-setup demonstration; the same data model maps directly onto Cloud Firestore collections (Table VI) with role-based security rules when the optional cloud-connected configuration is enabled, at which point the design also handles personal data (reporter email, scan history) in line with the safeguards expected under India's Digital Personal Data Protection Act, 2023 [17].

F. Implementation Stack and Module Architecture

The application is organized into twelve functional modules (Table VI): five service-layer modules handling authentication, heuristic analysis, ML scoring and persistence, AI explanation, and community reports; and seven role-aware UI views. Fig. 3 traces the sequence of calls a single scan makes across these modules, from the user's click through both detection layers, persistence, and the AI-explanation round trip.

Table VI. Implementation modules and data model

Layer	Modules
Service layer (5)	Authentication (AuthContext), URL Heuristic Engine (urlAnalyzer.ts), ML Scan Service (mlEngine.ts + scanService.ts), AI Explanation Service (aiService.ts), Report Service (reportService.ts)
UI layer (7)	Login / Auth view, Dashboard, Scanner (ScanPage), Algorithm Comparison (AlgorithmPage), History (HistoryPage), Report / Analyst view, Admin view
Firestore collection: users	uid, email, role, createdAt, scansCount, threatsFound
Firestore collection: scans	id, url, userId, threatLevel, confidenceScore, indicators[], aiAnalysis, isBulk, timestamp
Firestore collection: reports	id, url, reportedBy, category, description, status, verifiedBy, timestamp

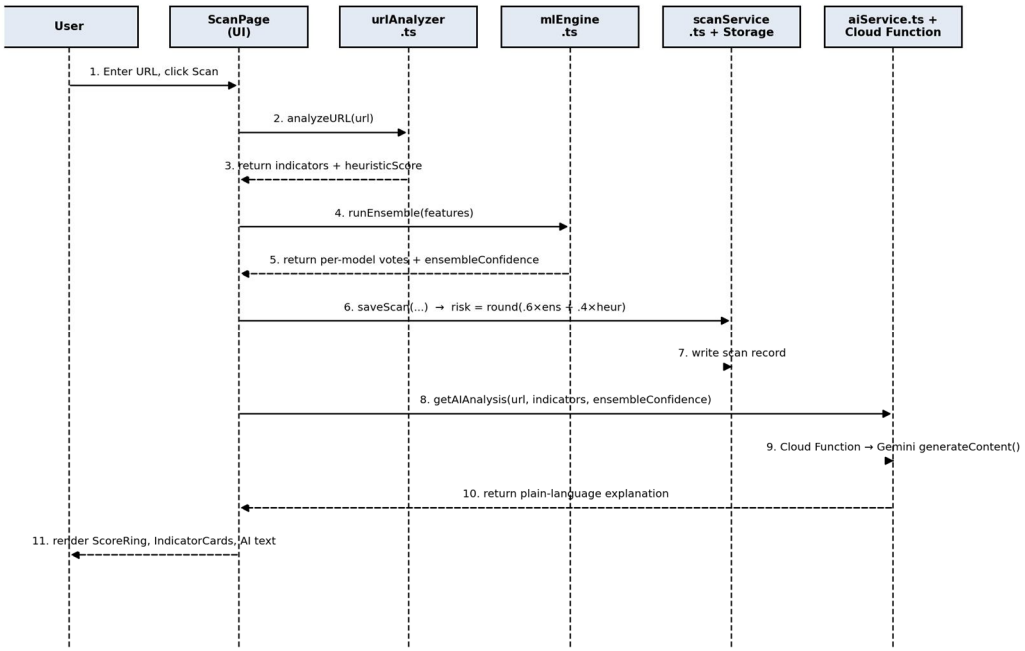


Fig. 3. Sequence of calls for a single scan, from user input through both detection layers to the persisted, explained result.

G. Illustrative Scan Example

Figs. 4 and 5 show the deployed application scanning https://paypa1-secure-login.tk/verify, a URL that combines the homograph technique introduced in Section I (the digit "1" substituted for the letter "l" in "paypal") with a high-abuse top-level domain and urgency-oriented path text. The heuristic engine (Section IV-B) triggers three indicators for this URL — a Critical-severity homograph match (rule 8), a Medium-severity suspicious-TLD flag (rule 3), and a Medium-severity suspicious-keyword flag (rule 10) — for a heuristic score of 60. Independently, the six-model ensemble votes toward Phishing with individual confidences ranging

from 66% (Random Forest) to 100% (Logistic Regression and Decision Tree); the F1-weighted average of Section IV-C combines these into an ensemble confidence of 90, the figure quoted in the AI-generated explanation of Fig. 5. Blending the two per the fixed 60:40 formula of Section IV-C, round(0.6 × 90 + 0.4 × 60) = 78, which matches the risk score shown on the scan ring in Fig. 4, and the URL is correctly classified as Phishing. Once scanned, the verdict is timestamped and persisted to the Scan History view (Fig. 6) for later review.

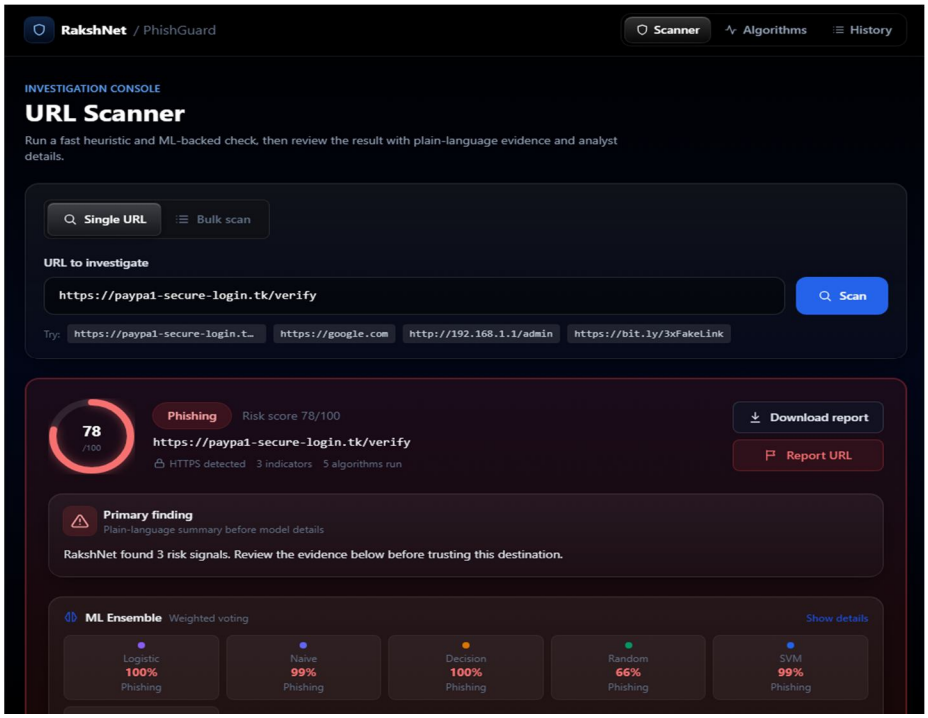


Fig. 4. URL Scanner view after analyzing a known-phishing test URL: risk-score ring, threat badge, and per-algorithm ML ensemble votes.

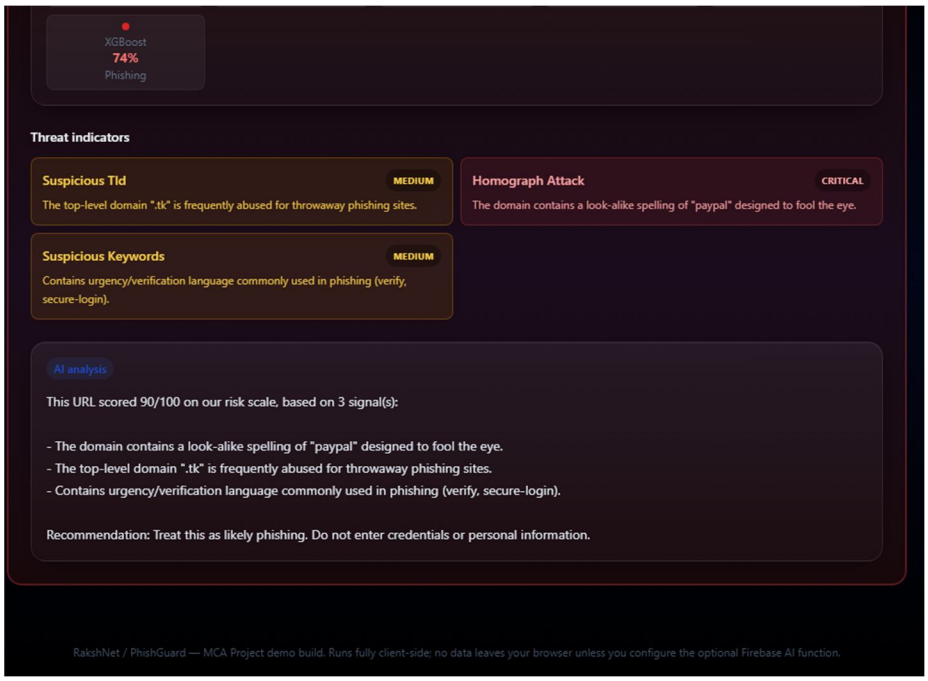


Fig. 5. Triggered threat indicators and the AI-generated plain-language explanation for the same scan.

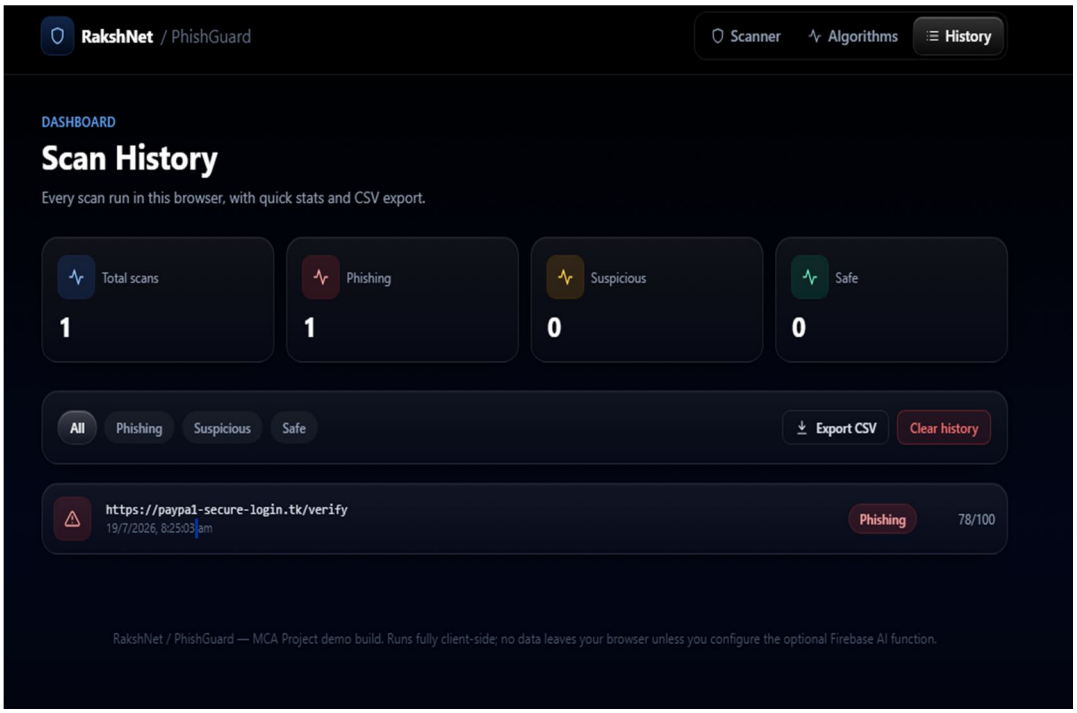


Fig. 6. Scan History view showing the example scan logged with its risk score, alongside aggregate session statistics.

V. RESULTS AND DISCUSSION

A. *Cross-Validated Model Performance*

Table VII reports the 5-fold cross-validated metrics for all six trained classifiers, measured out-of-sample on the 168-URL benchmark sample described in Section IV-D. Logistic Regression achieved the best overall balance of accuracy (79.17%), precision (84.34%), and AUC (0.861), suggesting that the relationship between the lexical features and the phishing label is largely linearly separable in this sample (Fig. 7). The Support Vector Machine was the closest competitor (76.19% accuracy, 0.859 AUC) and achieved the single highest recall (81.52%), a desirable property for a security tool where missing a genuine phishing URL is typically costlier than a false alarm on a legitimate one. Fig. 8 shows these same figures reproduced live inside the deployed Algorithm Comparison view, alongside an accuracy-comparison chart and a multi-metric radar plot.

Table VII. 5-Fold Stratified Cross-Validation Results (N = 168)

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	79.17%	84.34%	76.09%	80.00%	0.861
Support Vector Machine	76.19%	76.53%	81.52%	78.95%	0.859
XGBoost (Gradient Boosting)	74.40%	75.26%	79.35%	77.25%	0.831
Random Forest	71.43%	73.40%	75.00%	74.19%	0.832
Naive Bayes	69.05%	70.83%	73.91%	72.34%	0.771
Decision Tree	66.67%	70.93%	66.30%	68.54%	0.716

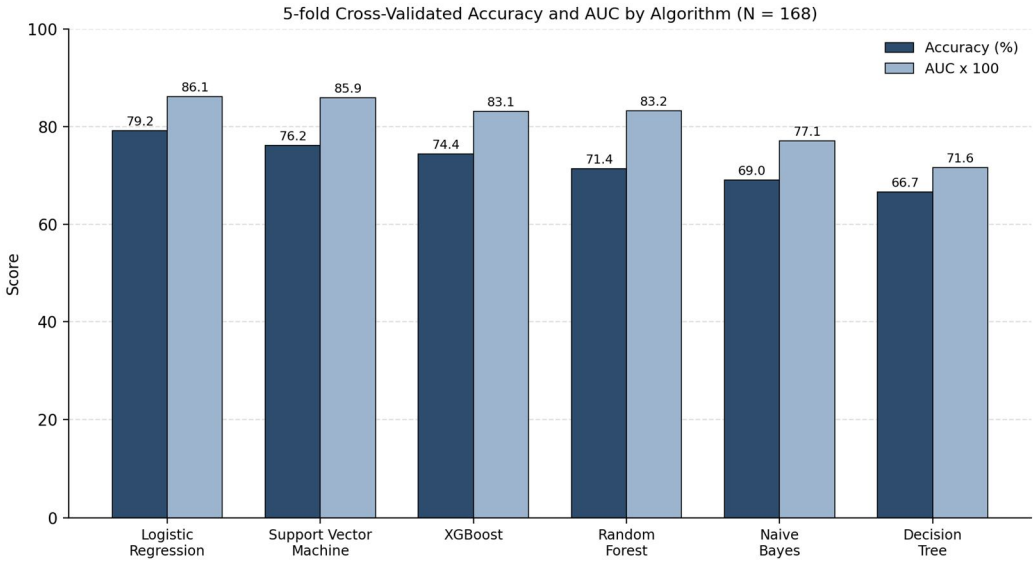


Fig. 7. Cross-validated accuracy and AUC for all six trained classifiers.

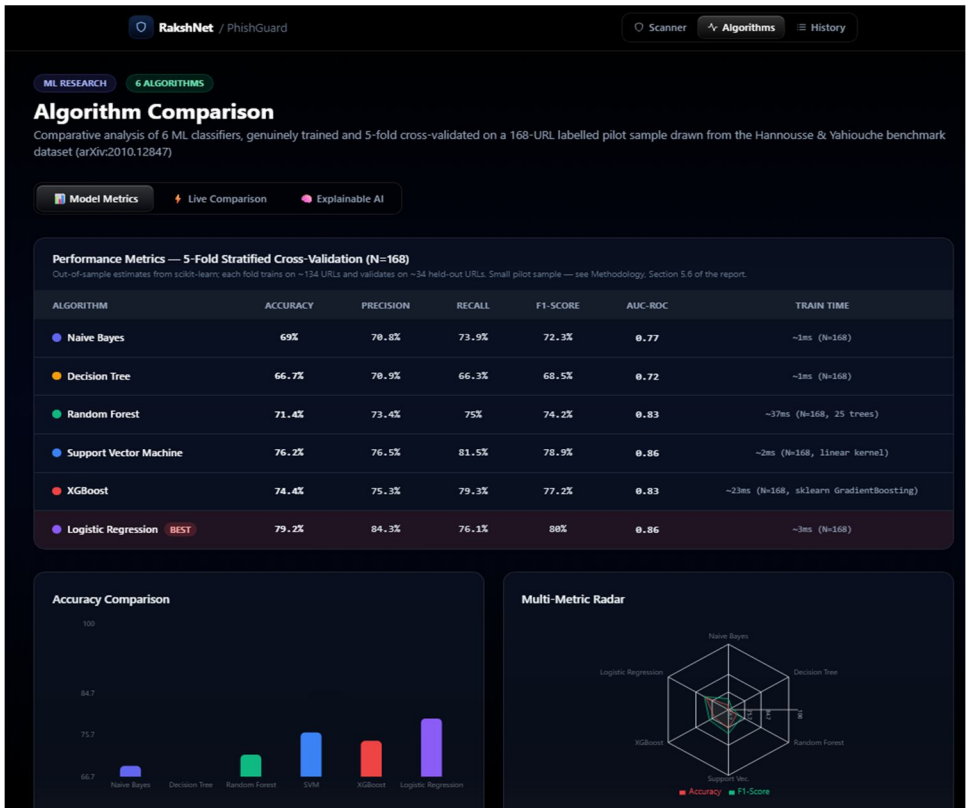


Fig. 8. Algorithm Comparison dashboard in the deployed application, reproducing the cross-validated metrics of Table VII with accuracy and multi-metric radar visualizations.

The tree-based models underperformed the linear models on this dataset, consistent with the small sample size: tree ensembles typically need more data than 168 rows split five ways (roughly 134 training examples per fold) to outperform well-regularized linear classifiers. Naive Bayes' conditional-independence assumption is also a poor fit for correlated lexical features such as domain length and subdomain count, which likely explains its comparatively lower precision.

B. Feature Importance Analysis

Fig. 9 plots the trained Random Forest's Gini feature importances, computed directly from the exported model and not re-estimated for this paper. Three structural features — hyphen count (17.6%), digit count in the domain (16.4%), and domain length (15.2%) — account for roughly half of the model's total learned signal, with URL length, path length, slash count, and the HTTPS flag contributing most of the remainder. Critically, all ten of the categorical indicators the heuristic engine specifically targets — IP-literal hosts, suspicious TLDs, brand-in-subdomain, homograph detection, Punycode, data: URIs, and the typosquatting score among them — show an importance of exactly 0.0 in this trained model. This is not a modeling error: these conditions are simply too rare among 168 rows for a Gini-impurity-based split to ever select them. It is the direct, model-derived confirmation of the design argument made in Section IV-B — that a hand-coded rule layer independent of the trained ensemble is necessary, not merely convenient, given the current dataset size, since the trained models provably cannot see the categorical signal at all at this sample size.

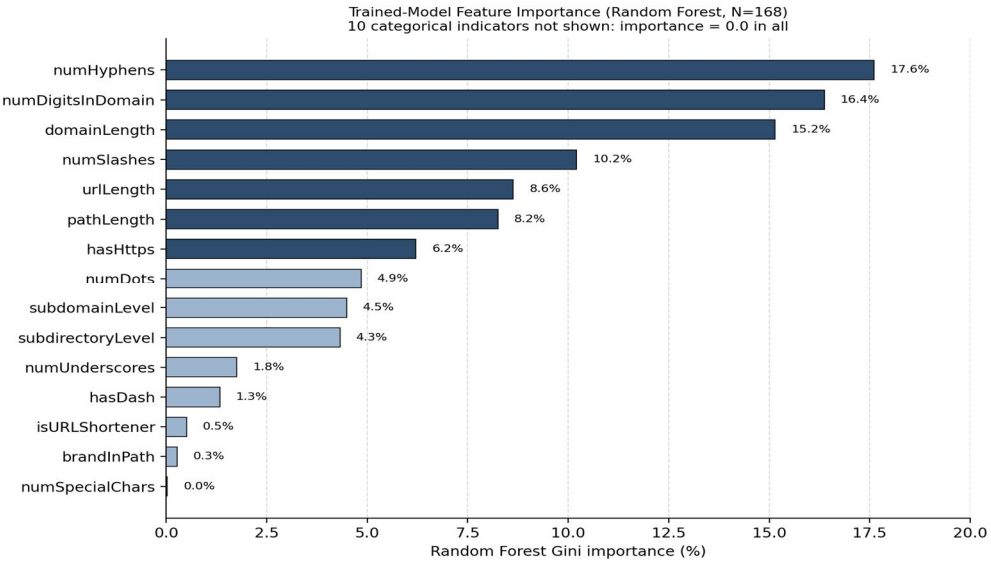


Fig. 9. Random Forest Gini feature importance for all twenty-five features; the ten categorical indicators with zero importance are the ones the heuristic engine (Table V) exists to catch.

C. Functional and Security Testing

Beyond the classifier-level metrics, the project's documented test plan spans six testing categories (Table VIII) and eight specific functional and security test cases (Table IX) exercising the end-to-end scanning workflow: single- and bulk-URL scans against known-phishing and known-safe URLs, community report submission, role-based access restrictions, and the AI-explanation fallback path. All eight scenarios produced their expected result. Heuristic analysis alone completes in under 50 ms per URL against a 100 ms target, and full AI-explanation generation completes within approximately 3 seconds on average against a 5-second target when the cloud function is reachable, falling back instantly to a local template when it is not.

Table VIII. Testing Categories And Focus Areas

Testing Type	Focus Area	Example Tests
Functional	Heuristic engine accuracy	Known phishing URLs, clean URLs, edge cases
Security	Access-control rules	Role-escalation prevention, unauthorized access
Integration	Scan pipeline	URL input → data write → AI display
Performance	Latency	Heuristic < 100 ms, AI analysis < 5 s
Regression	Post-update validation	Re-run after heuristic rule changes
UI / UX	User workflows	Login, scan, report, admin flows

Table IX. Summary Of Documented Functional And Security Test Cases

Test ID	Scenario	Outcome
TC-01	Single-URL scan — known phishing URL	Score ≥ 55, classified Phishing, homograph indicator shown
TC-02	Single-URL scan — known safe URL	Score 0–24, classified Safe, zero indicators
TC-03	Bulk scan (5 URLs)	Progressive per-URL results with aggregate counts
TC-04	Community URL report submission	Report saved with category, description, pending status
TC-05	Role-escalation prevention	Non-analyst user blocked from analyst-only route
TC-06	Analyst report verification	Report status updated to verified
TC-07	Admin role change	User role updated; new permissions took effect
TC-08	AI-service fallback	Local deterministic explanation shown when cloud function unreachable

D. Comparative Discussion

Returning to the positioning of Table I, RakshNet–PhishGuard's combination of zero-day capability, no dependency on a live database for its core verdict, and a natural-language explanation is not matched by any single category of existing tool: blocklists and enterprise gateways win on precision for previously-seen threats but cannot explain a verdict or catch a brand-new domain quickly; academic ML and deep-learning classifiers catch zero-day patterns but, as [5] itself notes, typically offer no explanation a non-technical user can act on. The trade-off, made explicit rather than hidden, is scale: the 67–79% cross-validated accuracy reported in Table VII reflects a 168-row pilot sample, not the hundreds-of-thousands-of-URL corpora available to large benchmark studies such as [3] or [6], and should be read as a starting point rather than a final result.

E. Threats to Validity and Limitations

The primary threat to the validity of the accuracy figures in Table VII is sample size: several categorical heuristic indicators occurred too infrequently in 168 rows for the trained models to assign them a non-trivial learned weight (Section V-B), and 5-fold cross-validation on a sample this small trains each fold on roughly 134 rows, which widens the variance of any single fold's estimate relative to a large-sample study. A second limitation is scope: the feature set is purely lexical and structural, so a phishing page that is lexically unremarkable but visually cloned (a concern [9] addresses with screenshot input) would not be caught by either detection layer here. Third, as with any lexical classifier, an adaptive attacker aware of the feature set could in principle craft URLs designed to minimize the learned risk score; the independent, hand-coded rule layer (Section IV-B) mitigates but does not eliminate this risk, since its rules are themselves public once the source is available. These limitations directly motivate the future work of Section VII, particularly dataset scaling via the community-reporting loop and the addition of host-based and visual signals.

VI. CONCLUSION

This paper presented RakshNet–PhishGuard, a client-first phishing URL detection system that pairs a transparent heuristic rule engine with a six-model machine-learning ensemble trained on lexical URL features, and augments the resulting verdict with a plain-language explanation aimed at non-technical users. Cross-validated experiments on a 168-URL benchmark sample show that Logistic Regression and a linear-kernel SVM deliver the strongest and most balanced performance (79.17% and 76.19% accuracy respectively, with AUCs above 0.85), confirming that lexical URL structure alone is a meaningful predictor of phishing intent, while a Random Forest feature-importance analysis confirms that the rare categorical indicators the heuristic layer exists to catch are provably invisible to the trained models at this sample size. Positioned against nine related works spanning feature-engineered ML, deep representation learning, and LLM-assisted detection, the system's combination of zero-network detection, an auditable rule layer, a genuinely cross-validated ensemble, and natural-language explanation is not matched by any single existing category of tool.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Because both the feature extraction and every trained model run entirely in the browser, the system can classify a URL instantly and offline, without depending on a threat-intelligence database that is, by nature, always at least one step behind newly registered phishing infrastructure. Scaling the training dataset and closing the loop between the community-reporting layer and model retraining are identified as the most immediate paths to improving accuracy further.

VII. FUTURE SCOPE

1) Scale the training corpus from the 168-URL pilot sample to the full 11,430-URL Hannousse–Yahiouche release [1], or to a growing, analyst-verified corpus sourced from the system's own reporting module.
2) Incorporate host-based signals (domain age, DNS record patterns) and, where feasible, visual-similarity checks against known brand assets along the lines of [9], to catch phishing pages that are lexically unremarkable but visually cloned.
3) Study adversarial robustness: an attacker aware of the feature set could craft URLs designed to minimize the learned risk score; periodic retraining and the independent rule-based layer both mitigate but do not eliminate this risk.
4) Close the loop between the community-reporting module and model retraining, so that analyst-verified reports automatically enrich the training set.
5) Package the scanning pipeline as a browser extension for real-time protection at the point of a click, rather than requiring the user to paste a URL into the web app.
6) Migrate persistence from browser localStorage to a managed backend (Firestore or a SQL database) as usage scales beyond a single-browser demo session.
7) Revisit the LLM-only detection route periodically: as [7] and [8] show this is an active research area, a future iteration could re-evaluate whether an LLM-embedding-based classifier has become fast enough to replace part of the hand-engineered feature set without reintroducing unacceptable latency.

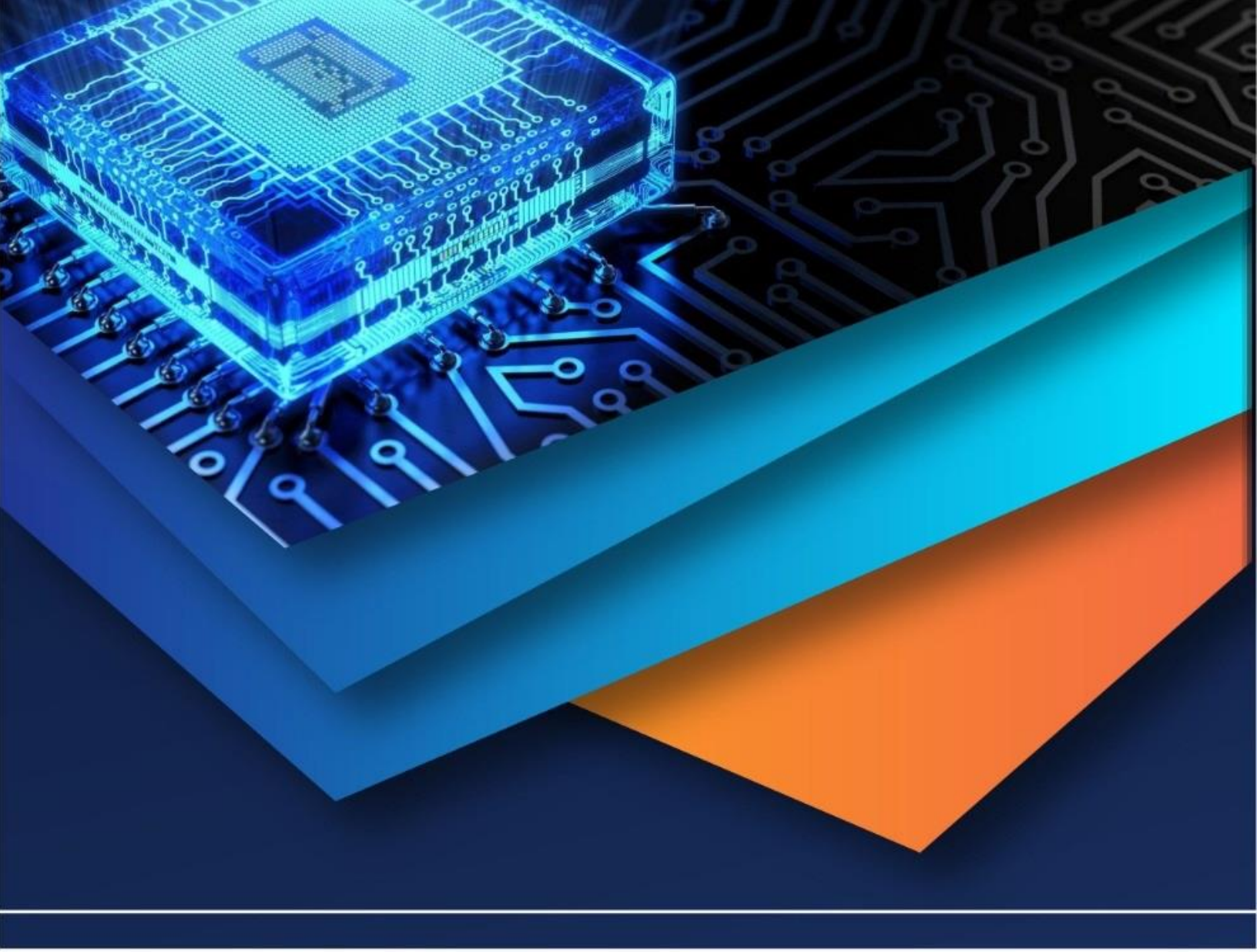
VIII. ACKNOWLEDGMENT

The authors thank the Department of MCA, AQJ College for PG Studies (affiliated to Andhra University), Visakhapatnam, for guidance and support during the development of RakshNet–PhishGuard. An AI writing assistant was used for wording, diagram generation, and document formatting in the preparation of this manuscript; the system design, implementation, dataset, and experimental results are the authors' own work.

REFERENCES

[1] A. Hannousse and S. Yahiouche, "Towards benchmark datasets for machine learning based website phishing detection: An experimental study," arXiv:2010.12847, 2021.
[2] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting phishing websites based on self-structuring neural network," Neural Computing and Applications, vol. 25, no. 2, pp. 443–458, 2014, doi:10.1007/s00521-013-1490-z.
[3] 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.
[4] E. Zhu, Y. Chen, C. Ye, X. Li, and F. Liu, "OFS-NN: An effective phishing websites detection model based on optimal feature selection and neural network," IEEE Access, vol. 7, pp. 73271–73284, 2019, doi:10.1109/ACCESS.2019.2920655.
[5] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL representation with deep learning for malicious URL detection," arXiv:1802.03162, 2018.
[6] F. Türk and M. Kılıçaslan, "Malicious URL detection with advanced machine learning and optimization-supported deep learning models," Applied Sciences (MDPI), vol. 15, no. 18, art. 10090, 2025.
[7] H. Kibriya, R. Amin, S. S. Alshamrani, S. Rehman, M. Hassan, and F. S. Alsubaei, "Lightweight malicious URL detection using deep learning and large language models," Scientific Reports, vol. 15, art. 43044, 2025, doi:10.1038/s41598-025-26653-2.
[8] F. Ji and D. Kim, "How can we effectively use LLMs for phishing detection?: Evaluating the effectiveness of large language model-based phishing detection models," arXiv:2511.09606, 2025.
[9] J. Lee, P. Lim, B. Hooi, and D. M. Divakaran, "Multi-modal large language models for phishing webpage detection and identification," arXiv:2408.05941, 2024.
[10] Anti-Phishing Working Group (APWG), "Phishing Activity Trends Report," [Online]. Available: https://apwg.org.
[11] Google, "Google Safe Browsing," [Online]. Available: https://safebrowsing.google.com.
[12] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[14] Google Cloud, "Vertex AI — Gemini models," [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs.
[15] React documentation, Meta Platforms Inc., [Online]. Available: https://react.dev.
[16] Firebase documentation, Google LLC, [Online]. Available: https://firebase.google.com/docs.
[17] Ministry of Electronics and Information Technology, Government of India, "Digital Personal Data Protection Act, 2023," Gazette of India, 2023.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)