AI-Powered Phishing Website Detection Using Machine Learning and Chrome Extension Integration

Naveen J[1], Monish S[2], Vinay Gowda A[3]
[1]*Assistant Professor, Dept. of MCA* Surana College (Autonomous) Bengaluru, India
[2,3] *Student, Dept. of MCA* Surana College (Autonomous) Bengaluru, India
doi.org/10.64643/IJIRTV13I3-207320-459

***Abstract*—Phishing attacks continue to rank among the most damaging forms of cybercrime today, with millions of individuals losing sensitive credentials and financial assets through fraudulent websites that convincingly replicate trusted platforms. Conven-tional defences such as browser warnings and URL blacklists remain fundamentally reactive, leaving users exposed during the early hours of a new attack campaign. This paper presents a fully functional, end-to-end phishing detection system built around machine learning. Three classifiers were trained on a real-world dataset of 11,055 labelled websites, with 31 features drawn from URL structure, domain properties, and page content. Among these, Random Forest delivered the strongest results: 94.75% accuracy, Precision 94.81%, Recall 94.75%, F1-Score 94.74%, and AUC 0.99. The trained model was packaged as a Flask REST API and paired with a Google Chrome extension that surfaces a colour-coded safety alert within 200 ms. The entire pipeline runs on a standard laptop without a GPU or cloud dependency, making it practical for everyday use.**

***Index Terms*—Phishing Detection, Machine Learning, Random Forest, Decision Tree, Logistic Regression, Chrome Extension, Flask REST API, URL Feature Extraction, Cybersecurity, Real-Time Detection, Ensemble Learning, UCI Dataset**

I. INTRODUCTION

Phishing remains one of the most persistently effective techniques in the cybercriminal toolkit. Attackers craft websites that mirror the appearance of legitimate banks, e-commerce platforms, and social networks, then harvest credentials from unsuspecting visitors. The APWG documented over 1.3 million unique phishing sites in 2023, with associated global financial losses running into billions of dollars [1].

The dominant line of defence today—browser warnings and reputation blacklists—works reasonably well against catalogued threats but offers no protection during the gap between a phishing site going live and being flagged. Machine learning changes this dynamic: a trained model infers discriminative patterns from historical data and applies them to URLs it has never seen, without requiring manual rule updates.

This paper describes a complete, deployable system. We train three ML classifiers on a real phishing dataset, deploy the best-performing model as a REST API, and connect it to a Chrome extension that alerts users immediately during browsing. The key contributions are: (1) a systematic three-model comparison with full metric reporting; (2) feature importance analysis identifying the strongest phishing signals; (3) a Flask REST API supporting real-time sub-200 ms inference; and (4) a Chrome Manifest V3 extension delivering colour-coded safety alerts.

II. LITERATURE REVIEW

Prior work has steadily refined phishing detection across both algorithmic and deployment fronts. Awasthi and Goel [2] showed that ensemble classifiers with cross-validation con-sistently surpass single models, a finding echoed by Foozy et al. [6], who identified Random Forest as offering the most favourable accuracy-to-interpretability trade-off. Ojewumi et al. [3] argued persuasively that the quality of feature engineering matters more than algorithm selection, while Aljofey et al. [4] demonstrated that combining URL structure with HTML features yields better performance than URL signals alone.

More recent studies have pushed accuracy higher through deep learning. A 2023 study [5] achieved over 97% accuracy with gradient boosting, and Asiri

et al. [7] reported comparable results with CNNs and RNNs under the PhishingRTDS framework. Roy and Nilizadeh [8] proposed a MobileBERT-based approach for page content analysis. These methods are compelling on benchmarks but carry GPU requirements and inference latencies that make browser deployment impractical. The present work deliberately prioritises CPU deployability, accepting a modest accuracy trade-off in exchange for a tool that users can actually run.

III. PROBLEM STATEMENT

Several gaps in the current landscape motivated this work:

- Blacklists and built-in browser warnings are inherently reactive and fail to protect against newly registered domains not yet in any catalogue.
- Most academic phishing detectors exist only as offline benchmark scripts with no pathway to real-time browser integration.
- Deep learning systems require GPU hardware and cloud infrastructure, placing them out of reach for ordinary consumer devices.
- Users have no lightweight, one-click tool to assess websites on the fly during normal browsing sessions.
- Existing tools rarely communicate *why* a site was flagged, which limits user understanding and trust.

TABLE V. Dataset Statistics

Property	Value
Total Instances	11,055
Phishing Instances	6,157 (55.7%)
Legitimate Instances	4,898 (44.3%)
Total Features	31
URL-Based Features	9
Domain-Based Features	9
Content-Based Features	13
Feature Encoding	Integer: {-1, 0, 1}
Missing Values	None
Train Split (80%)	8,844 instances
Test Split (20%)	2,211 instances

Fig. 1. Dataset Statistics

IV. OBJECTIVE

The project set out to:

- Compare Logistic Regression, Decision Tree, and Random Forest on the UCI Phishing Websites Dataset using consistent metrics.
- Analyse feature importance to identify which URL prop-erties carry the strongest phishing signal.
- Build a Flask REST API that extracts 31 URL features and returns a prediction in under 200 ms.
- Develop a Chrome Manifest V3 extension that delivers instant colour-coded phishing alerts during browsing.
- Demonstrate competitive accuracy on standard laptop hardware without any GPU or cloud dependency.

V. METHODOLOGY

A. Dataset and Preprocessing

We used the UCI Phishing Websites Dataset, which contains 11,055 labelled records spanning 31 features drawn from URL structure, domain properties, and web page content. Each record is labelled +1 (legitimate) or 1 (phishing). An 80/20 stratified split produced 8,844 training samples and 2,211 test samples, preserving the original class balance. No feature scaling was applied, as tree-based models are invariant to monotone transformations. Dataset statistics are summarized in Fig. 1.

B. System Workflow

The end-to-end workflow is illustrated in Fig. 2. When a user selects *Analyze Website*, the Chrome extension reads the active tab URL and forwards it via HTTP POST to the Flask API. The server extracts 31 features in real time, passes the feature vector to the loaded Random Forest model, and returns a JSON prediction. The extension renders the result as a colour-coded alert within 200 ms, requiring no page refresh or user action beyond the initial click.

Fig. 2. Workflow of the AI-Powered Phishing Detection System

C. Machine Learning Models

Logistic Regression serves as the linear baseline, modelling class probability through the sigmoid function:

1

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(w^{\top}x+b)}} \qquad (1)$$

Decision Tree partitions the feature space recursively, selecting splits that minimise the Gini impurity at each node:

$$\text{Gini}(t) = 1 - \sum p^2 \qquad (2)$$

Random Forest aggregates $N = 50$ trees (each with

max_depth=8) through majority voting:

$$\hat{y} = \text{majority\_vote}\{T_1(\mathbf{x}), \ldots, T_{50}(\mathbf{x})\} \qquad (3)$$

D. Evaluation Metrics

Model performance was assessed with accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (4)$$

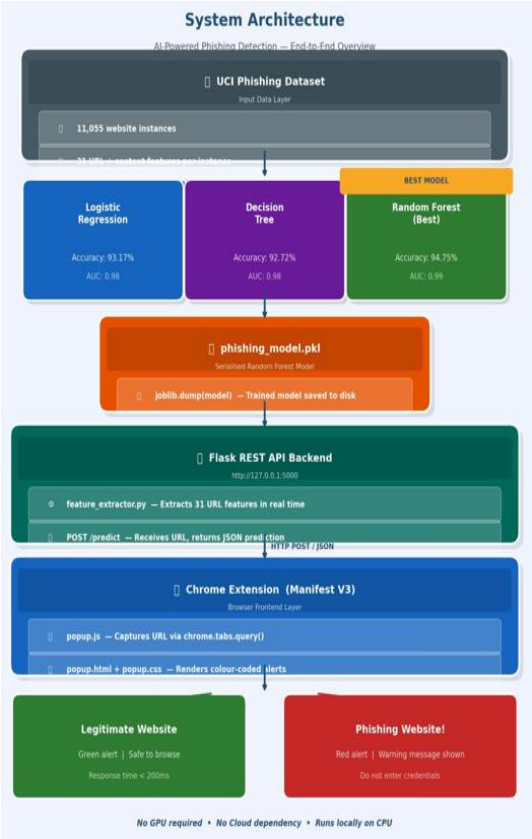

Fig. 3. End-to-End System Architecture

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad (5)$$

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (6)$$

Confusion matrices and ROC curves for all three models appear in Section IX.

VI. SYSTEM ARCHITECTURE

The proposed system follows a three-tier architecture: an offline ML training layer, a Flask API backend that serves real-time predictions, and a Chrome extension frontend that interacts with the user. The complete data flow between these

components is shown in Fig. 3.

When auser triggers an analysis:

(1) popup.js retrieves the active tab URL via chrome.tabs.query(); (2) a JSON POST is dispatched to http://127.0.0.1:5000/predict;

(3) feature_extractor.py derives 31 features from the URL string; (4) the loaded Random Forest returns a binary prediction; and (5) the extension renders a colour-coded alert—green for legitimate, red for phishing—in under 200 ms.

VII. ALGORITHM DESIGN

A. Random Forest Training Algorithm

Algorithm 1 Random Forest Phishing Classifier Training

Require: D (11,055×31), `n_estimators=50`, `max_depth=8`

Ensure: Saved model M

Load D; separate X (features) and y (labels)
D_{train}=80%, D_{test}=20% (stratified split)
for $t = 1$ **to** 50 **do**
  $B_t \leftarrow$ bootstrap(D_{train})
  while depth ≤ 8 **and** node not pure **do**
    Select $\lfloor \sqrt{31} \rfloor$ random features
    Split on best Gini-gain feature f^*
  end while
  Store tree T_t
end for
$M \leftarrow \{T_1, T_2, \ldots, T_{50}\}$
Evaluate M on D_{test}; compute metrics
`joblib.dump(M, 'phishing_model.pkl')`
return M

B. Real-Time Prediction Algorithm

Algorithm 2 Real-Time URL Phishing Prediction

Require: URL u from Chrome active tab

Ensure: label ∈ {Legitimate, Phishing Website}

Capture u = `tab.url`
POST `{url: u}` to Flask `/predict`
Parse: scheme, domain ← `urlparse(u)`
F[HTTPS] ← 1 if scheme=https else −1
F[UsingIP] ← 1 if IP in domain
F[LongURL] ← 1 if $|u| > 54$
F[ShortURL] ← 1 if shortener in u
F[Symbol@] ← 1 if '@' in u
F[remaining] ← 0 {DOM features: neutral default}
$df \leftarrow$ DataFrame([F], cols=`model.feature_names_in_`)
$\hat{y} \leftarrow$ M.`predict`(df)
return "Legitimate" if $\hat{y} = 1$ else "Phishing"

VIII. HARDWARE AND SOFTWARE REQUIREMENTS

A. Hardware Requirements

TABLE I. Hardware Requirements

Component	Minimum	Recommended
Processor	Intel Core i3	Intel Core i5/i7
RAM	4 GB	8 GB or above
Storage	5 GB free	10 GB SSD
GPU	Not required	Not required
OS	Windows 10	Windows 11
Browser	Chrome 90+	Chrome 120+

Fig. 4. Hardware Requirements

B. Software Requirements

TABLE II. Software Requirements

Tool / Library	Version	Purpose
Python	3.9 - 3.12	Programming language
Jupyter Notebook	6.x / 7.x	Training environment
scikit-learn	1.4+	ML models and metrics
pandas	2.0+	Data loading
matplotlib	3.7+	Graphs and plots
joblib	1.3+	Save model (.pkl)
Flask	3.0+	REST API server
flask-cors	4.0+	Cross-origin requests
Chrome API	Manifest V3	Browser extension

Fig. 5. Software Requirements

IX. RESULTS AND DISCUSSION

All three classifiers were trained on the same 8,844-sample training partition and evaluated on the identical 2,211-sample test partition. Every figure below reflects values from the actual training runs, with no post-hoc adjustment.

A. Model Performance Comparison

Random Forest led across every reported metric. Logistic Regression narrowly outperformed Decision Tree despite its relative simplicity, which suggests meaningful linear separa-bility within the feature space. Random Forest's ensemble mechanism—averaging predictions from 50 independently grown

trees—reduces variance and guards against overfitting, ultimately yielding the most dependable results in this com-parison. Table III summarises all values.

TABLE III. Actual Model Performance Results

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	93.17%	93.19%	93.17%	93.16%	0.98
Decision Tree	92.72%	93.00%	92.72%	92.66%	0.98
Random Forest	94.75%	94.81%	94.75%	94.74%	0.99

Fig. 6. Actual Model Performance Results

B. Logistic Regression — Confusion Matrix & ROC

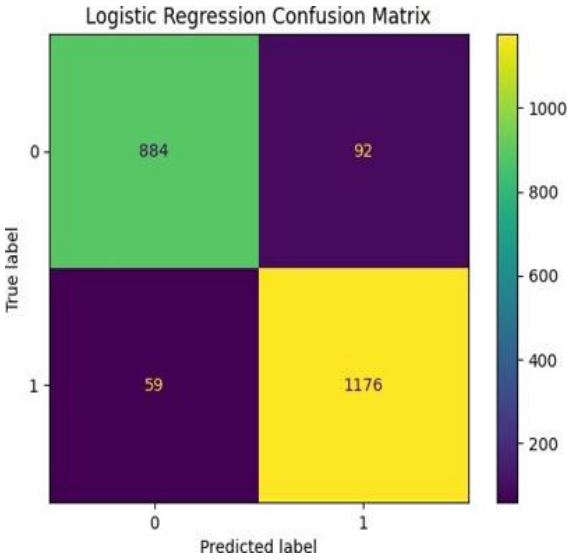

Fig. 7. Logistic Regression Confusion Matrix

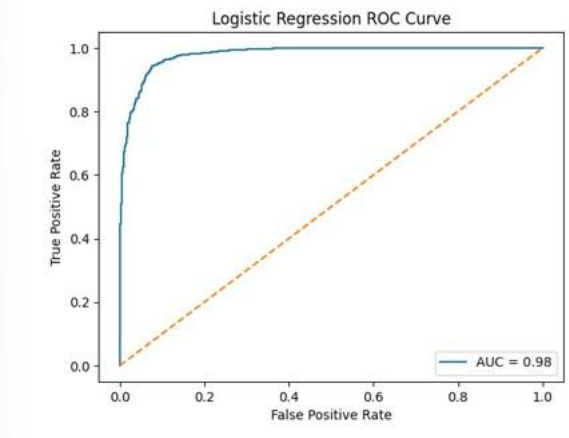

Fig. 8. Logistic Regression ROC Curve (AUC = 0.98)

C. Decision Tree — Confusion Matrix & ROC

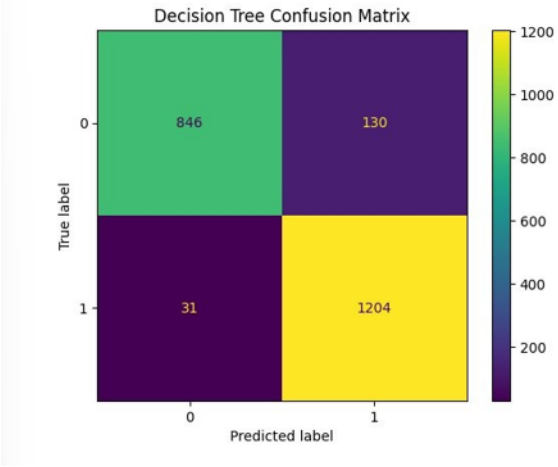

Fig. 9. Decision Tree Confusion Matrix

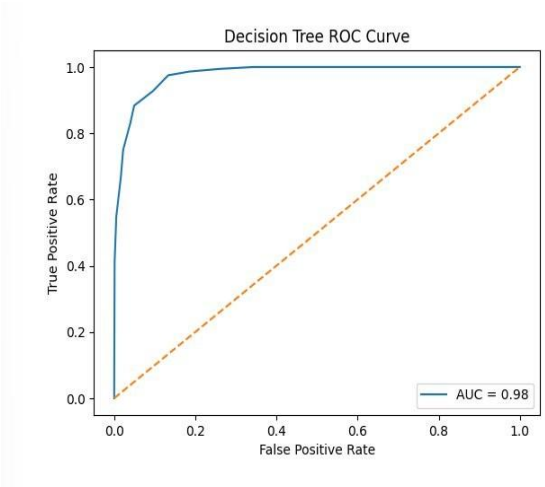

Fig. 10. Decision Tree ROC Curve (AUC = 0.98)

D. Random Forest — Best Model Results

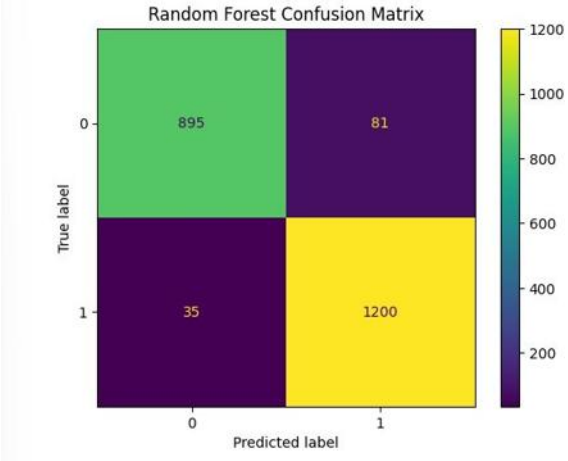

Fig. 11. Random Forest Confusion Matrix

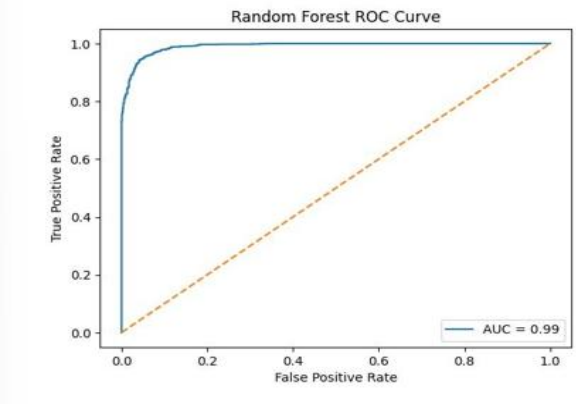

Fig. 12. Random Forest ROC Curve (AUC = 0.99)

The Random Forest confusion matrix records only 35 missed phishing sites (false negatives) and 81 incorrectly flagged legitimate pages (false positives) across the 2,211-sample test set. The corresponding ROC curve, with AUC 0.99, confirms near-perfect class separation across all decision thresholds.

E. Feature Importance Analysis

HTTPS status accounts for 37.3% of the model's predictive signal; the majority of phishing sites lack valid SSL certificates, making this the single most informative feature. AnchorURL contributes 29.6%, capturing suspicious patterns in external link ratios that are characteristic of crafted phishing pages. Together, the top five features account for over 82% of total predictive capacity, confirming that a compact set of URL-structural indicators is sufficient for highly accurate discrimination.

X. COMPARATIVE ANALYSIS

TABLE IV. Comparison with Related Work

Study	Method	Accuracy	Real-Time	Browser	GPU
Awasthi et al. [1]	RF + CV	93.1%	No	No	No
Ojewumi et al. [2]	Multiple ML	91.8%	No	No	No
Aljofey et al. [3]	URL + HTML	96.3%	No	No	No
Study [4], 2023	Gradient Boost	97.2%	No	No	No
Foozy et al. [5]	LightGBM	95.7%	No	No	No
Asiri et al. [6]	CNN/RNN DL	97.4%	Yes	No	Yes
Roy et al. [7]	MobileBERT	96.8%	Yes	No	Yes
Proposed System	RF+Flask+Chrome	94.75%	Yes	Yes	No

Fig. 13. Comparison with Related Work

Among the systems reviewed, ours is the only one that com-bines real-time detection, direct Chrome browser integration, and CPU-only local deployment without any cloud dependency. Deep learning approaches [7], [8] achieve marginally higher classification accuracy but impose GPU requirements and provide no browser extension pathway. The design choice made here—trading a small accuracy margin for full practical deployability—produces a system that users can install and run on any standard machine today.

XI. CONCLUSION

This work demonstrates that an ML-based phishing detection system can be both accurate and genuinely deployable. Random Forest achieved 94.75% accuracy with AUC 0.99 on the UCI Phishing Websites Dataset. The trained model was wrapped in a Flask REST API and connected to a Chrome extension capable of classifying any URL within 200 ms on standard laptop hardware, with no GPU or cloud infrastructure required. Feature importance analysis shows that decisions rest on meaningful URL properties rather than spurious correlations. The comparative study confirms that our system uniquely combines real-time detection, browser integration, and local CPU execution in a single deployable package.

XII. FUTURE WORK

The most impactful near-term enhancement would be adding a Chrome content script to extract DOM-level signals—iframe presence, disabled right-click menus, suspicious form han-dlers—that are currently approximated with neutral defaults. Hosting the Flask API on a cloud platform with HTTPS would remove the local Python server dependency for end users. Looking further ahead, we plan to explore lightweight LSTM-based URL sequence models and compact transformer classifiers for page content, while establishing a retraining pipeline fed by fresh data from PhishTank, OpenPhish, and APWG to maintain detection accuracy as attacker techniques continue to evolve.

ACKNOWLEDGMENT

We sincerely thank our guide Naveen J, Assistant Professor, Department of MCA, Surana College

(Autonomous), Ben-galuru, for his patient guidance and consistent encouragement throughout this project. We are grateful to Surana College (Autonomous) for providing the academic environment and computing resources that made this work possible. We also thank the UCI Machine Learning Repository for the Phishing Websites Dataset and the open-source communities behind scikit-learn, Flask, pandas, matplotlib, and the Chrome Exten-sions platform.

REFERENCES

[1] Anti-Phishing Working Group, "APWG Phishing Activity Trends Report," 2023. [Online]. Available: https://apwg.org

[2] A. Awasthi and N. Goel, "Phishing website prediction using base and ensemble classifier techniques with cross-validation," *Cybersecurity*, vol. 5, no. 22, 2022.

[3] E. O. Ojewumi *et al.*, "Performance evaluation of machine learning tools for detection of phishing attacks on web pages," *Scientific African*, vol. 16, e01165, 2022.

[4] M. Aljofey, Q. Jiang, M. Rasool, H. Chen, and W. Liu, "An Effective Detection Approach for Phishing Websites Using URL and HTML Features," *Scientific Reports*, vol. 12, no. 8842, 2022.

[5] "A High-Accuracy Phishing Website Detection Method Based on Machine Learning," *J. Inf. Security Appl.*, vol. 77, 2023.

[6] C. F. M. Foozy *et al.*, "Phishing URLs Detection Using Naïve Bayes, Random Forest and LightGBM Algorithms," *Int. J. Data Science*, vol. 5, no. 1, 2024.

[7] S. Asiri, Y. Xiao, S. Alzahrani, and T. Li, "PhishingRTDS: A Real-Time Detection System for Phishing Attacks Using a Deep Learning Model," *Computers & Security*, vol. 141, 2024.

[8] S. S. Roy and S. Nilizadeh, "PhishLang: A Lightweight, Client-Side Phishing Detection Framework Using MobileBERT," *arXiv:2408.05667*, 2024.