Real-Time URL-Based Phishing Detection Using XGBoost and Browser Extension Integration

K.U.T. Siriwardhana, A. Fathima Sharfana, and R.M. Nayanajith Rathnayaka

Abstract Phishing attacks continue to pose a significant cybersecurity threat by misleading users into disclosing sensitive information through fraudulent websites. This study presents a machine learning–based approach for detecting phishing websites using URL-based features. A dataset comprising phishing and legitimate URLs was collected from trusted sources, including PhishTank, OpenPhish, and Cisco Umbrella's Top Sites. Following preprocessing, key discriminative features such as URL length, HTTPS usage, special character frequency, and randomness indicators were extracted. Several classification algorithms, Random Forest, Support Vector Machine, Logistic Regression, and XGBoost, were trained and evaluated. Experimental results indicate that the XGBoost model achieved the highest accuracy of approximately 90%, demonstrating balanced detection performance with low computational overhead. The trained model was deployed through a Flask-based API and integrated with a web browser extension to enable real-time URL analysis and phishing alerts. The proposed system enables real-time phishing detection with an average inference latency of approximately 300–500 ms by deploying a lightweight, URL-feature–based machine learning model within a browser extension, without relying on blacklist databases or computationally heavy analysis.

***Index Terms*— Cybersecurity, Machine Learning, Phishing Detection, Real-time Detection, XGBoost**

I. Introduction

In modern digital era, people use the internet every day. However, numerous problems accompany this widespread usage. One of the most significant threats is phishing attacks[1]. Attackers frequently create fake websites with user interfaces that look almost real[2][3]stealing private sensitive information such as passwords and bank account details. These attacks affect normal users, companies, and even government sites. The nature of phishing threats continues to evolve continuously[4] making detection increasingly challenging.

Traditional phishing detection tools that rely on URL blacklists and basic rule-based filters are unable to identify newly created and zero-day phishing sites[5][6]. These conventional safety methods only work when a site is already marked as malicious. Attackers, however, are fast at creating new URLs, altering domain patterns, and employing evasion methods that existing systems cannot match. When hackers create a new phishing site, it often slips through traditional defenses. Most available solutions rely on third-party phishing databases to recognize threats[7], leaving users vulnerable when a phishing site is novel or not yet listed in such databases. Furthermore, current research does not focus sufficiently on real-time URL-feature-based detection with browser extensions and high accuracy. The majority of existing literature implements machine learning models in an offline mode and fails to include lightweight and instant URL analysis integrated into the browsing process.

To address these limitations, a proactive detection system is needed that learns from URL patterns, domain names, and other subtle signals that normal users cannot see. Such a system can determine whether a link is safe or not, even if it is new and previously unseen. This research presents a browser extension that runs while people use the internet, monitoring the websites users visit and checking them with a trained machine learning model on the server. When a user opens any website, the extension reads the website URL and sends URL features to the backend, where a trained machine learning model checks the link and studies its patterns. The model examines URL features and compares the link with what it learned from thousands of real and phishing URLs. If it finds something strange or risky, it marks that website as phishing and the extension delivers a quick warning message to the user. Users do not need to do anything manually, as everything works in the background automatically, providing seamless, real-time protection against evolving phishing threats.

K.U.T. Siriwardhana is a graduate from the Department of ICT, South Eastern University of Sri Lanka, Sri Lanka.

A. Fathima Sharfana is a Lecturer (Prob) at the Department of ICT, South Eastern University of Sri Lanka, Sri Lanka. (Email: sharfana.atham@seu.ac.lk)

R.M. Nayanajith Rathnayaka is a demonstrator at the Department of ICT, South Eastern University of Sri Lanka, Sri Lanka. (Email: nayanajith@seu.ac.lk)

ISSN: 2961-5992

II. LITERATURE REVIEW

A. Phishing Attacks and Detection Techniques

Phishing attacks represent one of the most dynamic and pervasive cybersecurity threats, continuously evolving to exploit vulnerabilities in user behavior and system security. Recent surveys indicate that attackers persistently refine their deceptive techniques, rendering conventional security measures increasingly inadequate in providing comprehensive protection[5][4].

Phishing attacks manifest through multiple vectors, each exploiting different aspects of digital communication and web browsing. URL-based phishing remains the most prevalent approach, where attackers craft malicious links that redirect users to fraudulent websites designed to mimic legitimate services[8][9]. These attacks have become increasingly sophisticated, employing homograph attacks, subdomain manipulation, and URL shortening services to disguise malicious intent[2]. Email phishing leverages social engineering principles to manipulate recipients into divulging credentials or executing harmful actions. Spoofed websites represent particularly dangerous threats, replicating the visual appearance and functionality of legitimate platforms with remarkable precision[10]. Browser-based phishing attacks exploit vulnerabilities within web browsers themselves, utilizing techniques such as tab-nabbing, clickjacking, and cross-site scripting to compromise user security without requiring navigation away from legitimate domains.

The cybersecurity community has developed several traditional methodologies to combat phishing threats. Blacklist-based detection maintains databases of known malicious URLs and domains, comparing accessed websites against these repositories to identify threats[5]. Organizations like Google Safe Browsing and PhishTank continuously update these databases, providing protection against previously identified threats. However, this reactive approach inherently suffers from temporal vulnerabilities. Heuristic-based methods attempt to overcome these limitations by analyzing website characteristics and behavioral patterns[11]. These approaches examine factors such as domain age, SSL certificate validity, presence of sensitive input forms, and similarity to known legitimate sites. Rule-based systems employ predefined criteria and decision trees to classify websites, considering elements like URL structure, HTML content patterns, and domain registration information[12].

Despite widespread adoption, traditional phishing detection techniques face significant limitations that compromise their effectiveness. The most critical weakness of blacklist-based approaches is the inevitable delay between phishing site deployment and blacklist updates. Conventional blacklist and rule-based systems struggle to keep pace with rapidly evolving attacker tactics[5][4]. Research indicates that phishing websites have an average lifespan of only 15-20 hours, yet blacklist propagation can take several hours or even days, creating substantial vulnerability windows during which users remain unprotected[6]. This temporal gap allows attackers to achieve their objectives before detection systems can respond.

High false positive rates present another significant challenge, particularly for heuristic and rule-based systems[1]. Overly aggressive detection parameters can flag legitimate websites as suspicious, leading to user frustration and security alert fatigue, where users begin ignoring warnings altogether. Conversely, conservative parameters may miss sophisticated phishing attempts that carefully crafted appearance to evade detection rules. The lack of real-time protection capabilities represents perhaps the most fundamental limitation. As phishing techniques evolve rapidly, incorporating artificial intelligence, dynamic content generation, and personalized social engineering, static rule-based systems struggle to adapt[13].

Recognizing these limitations, there has been a firm trend of shifting from static signature-based methods toward data-driven machine learning and deep learning models for detecting malicious URLs[5][13]. This transition reflects the cybersecurity community's acknowledgment that adaptive, intelligent systems are necessary to combat the dynamic nature of modern phishing threats. Traditional methods typically cannot analyze contextual website behavior, user interaction patterns, or real-time content modifications that characterize contemporary phishing attacks[14]. Furthermore, these approaches often operate independently of user browsing context, failing to consider factors such as typical browsing habits, previously visited sites, or the legitimacy of pathways leading to potentially malicious sites. The emergence of zero-day phishing attacks, which exploit previously unknown vulnerabilities or employ entirely novel deception techniques, further exposes the inadequacy of retrospective detection methods relying on historical data and predefined patterns[10]. This evolution toward intelligent, adaptive detection mechanisms forms the foundation for modern browser-based security solutions capable of providing real-time protection against increasingly sophisticated threats.

B. Machine Learning-Based Real-Time Phishing Detection in Browser Extensions

The integration of machine learning algorithms into browser-based phishing detection represents a paradigm shift from reactive security measures to proactive, intelligent threat identification. This data-driven approach leverages computational intelligence to analyze website characteristics in real-time, addressing the critical limitations inherent in traditional detection methodologies[15][16].

Early phishing detection research primarily employed lexical and domain features with classical machine learning algorithms including Support Vector Machines (SVM), Decision Trees, k-Nearest Neighbors (KNN), and Random Forests[1][9]. These foundational studies demonstrated that even basic URL attributes such as length, count of special characters, token composition, and protocol signature (HTTP/HTTPS) could achieve over 90 percent precision with appropriate classifiers[9][17].

Comparative analyses across diverse machine learning algorithms have revealed important performance distinctions. Abad & Gholamy[1] compared SVM, Decision Tree, Random Forest, and KNN on mixed malicious URL datasets, finding that SVM provided the highest accuracy with careful parameter tuning. However, subsequent research indicates that ensemble models like Random Forest and Gradient Boosting often outperform simple linear models when evaluated on public datasets from PhishTank,

ISSN: 2961-5992

Kaggle, and similar sources[18][17]. These findings confirm that nonlinear ensembles better capture complex URL feature patterns compared to simple classifiers[19].

Recent developments emphasize ensemble learning and strategic feature selection as critical design choices in phishing URL detection systems. Wajid et al.[7] introduced an ensemble voting classifier with hybrid ensemble feature selection (HEFS) evaluated on DS-30 and DS-50 datasets, achieving accuracy between 96-98 percent using only approximately one-fifth of the original features. This demonstrates that intelligent feature selection not only improves computational efficiency but can also enhance model performance by reducing noise and overfitting[20]. Similarly, research comparing non-ensemble models versus ensemble classifiers across balanced phishing datasets examined Random Forest, Bagging, Stacking, AdaBoost, and Gradient Boost architectures[6]. These studies consistently show that ensemble approaches provide superior robustness and generalization capabilities compared to individual classifiers[19][21].

Subsequent research has built upon foundational approaches by incorporating larger datasets and additional engineered features, including domain age, number of subdomains, scale tokens, and entropy-based indicators[18][22]. The effectiveness of machine learning models fundamentally depends on the quality and relevance of extracted features. URL-based features constitute the primary category, encompassing characteristics such as URL length, presence of special characters, domain structure, and protocol usage[17]. Domain-related features provide crucial contextual information, including WHOIS registration details, DNS record analysis, domain reputation scores, and geographic location of hosting servers[8][22]. Webpage content features offer rich discriminative information by analyzing HTML structure, JavaScript behavior, external resource links, presence of login forms, and visual similarity to known legitimate sites[12][10]. Behavioral features, including page redirect patterns, popup behavior, and form submission destinations, provide additional classification dimensions[12].

Implementing machine learning models within browser extensions presents unique technical challenges demanding careful architectural consideration. Latency represents the most critical constraint, as users expect instantaneous page loading without perceptible delays introduced by security scanning[23]. Research indicates that detection latency exceeding 200-300 milliseconds significantly degrades user experience, creating pressure to optimize model inference speed without sacrificing accuracy. Browser extensions operate within strict resource limitations, including memory constraints, CPU allocation restrictions, and sandboxed execution environments[7]. These constraints necessitate lightweight model architectures and efficient feature extraction pipelines. The balance between security and usability requires sophisticated user interface design, presenting warnings that are informative yet non-intrusive[23]. Continuous model updating poses additional challenges, as phishing techniques evolve rapidly and models must adapt without requiring frequent extension updates that burden users[14]. Edge-based inference offers privacy advantages and reduces dependency on external services but amplifies resource constraints, while cloud-based approaches enable more sophisticated models but introduce latency and privacy concerns[24][22].

As summarized in Table I, recent phishing detection studies increasingly emphasize high classification accuracy through ensemble learning and deep learning architectures. Many of these approaches report accuracy levels between 96–99% in offline evaluations by leveraging computationally intensive feature sets, including HTML content, image analysis, word embeddings, and external services such as DNS or WHOIS queries. However, these design choices introduce significant inference latency, large model sizes, and dependencies on network-based lookups, which limit their suitability for real-time browser-based deployment. Moreover, several high-accuracy studies focus primarily on offline benchmarking and do not provide functional browser implementations or detailed latency analysis. This highlights a broader trend in the literature toward accuracy-driven evaluation, while practical deployment constraints such as latency, model size, and real-time usability remain comparatively underexplored.

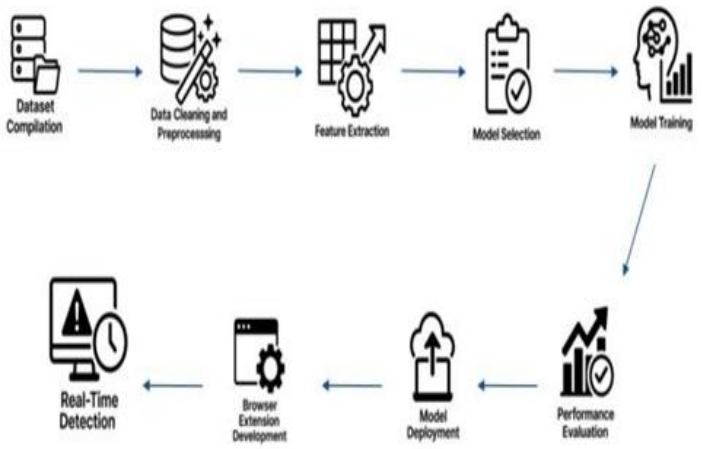

Fig. 1: Followed Methodology

III. METHODOLOGY

This section presents the comprehensive methodology employed to develop a real-time phishing detection system integrated with a browser extension. The proposed approach encompasses data collection, feature engineering, model development, hyperparameter optimization, and deployment through a Flask-based API. Figure 1 illustrates the complete workflow of the system architecture.

A. Data Collection and Preprocessing Image Files

The dataset construction process involved aggregating URLs from multiple trusted public sources to ensure balanced representation of both malicious and legitimate websites. Phishing URLs were primarily sourced from PhishTank and OpenPhish repositories, while legitimate URLs were obtained from Cisco Umbrella Top Sites and DomCop databases. The compiled dataset comprised 13,716 URLs, maintaining an equal distribution with 6,858 phishing URLs and 6,858 legitimate URLs to prevent class imbalance bias during model training.

ISSN: 2961-5992

TABLE I
SUMMARY OF PHISHING DETECTION STUDIES AND RESEARCH GAPS

Study	Approach Category	Techniques / Models Used	Dataset Size / Type	Feature Scope
[9]	Machine Learning	URL-based ML	6,000 URLs	Lexical
[7]	Machine Learning	Ensemble (HEFS, RF)	DS-30, DS-50	URL + Feature engineering
[13]	Machine Learning	LR, SVM, RF, GB	UCI dataset	Feature-based
[20]	Machine Learning	Random Forest	Mendeley dataset	URL + Domain
[6]	Machine Learning	Ensemble ML	Two datasets	URL + DNS + HTML
[18]	Machine Learning	GBoost, RF, DT	Feature-based (30 features)	URL + Statistical
[19]	Machine Learning	LR, RF, SVM, Ensembles	Mixed dataset	Multi-feature
[21]	Machine Learning	SVM	Custom dataset	Feature-heavy
[14]	Deep Learning	CNN-LSTM, RF	11,430 URLs	URL + Embeddings
[22]	Deep Learning / Hybrid	Word2Vec, GloVe, XGB, CNN, LSTM	Mixed datasets	Text embeddings
[23]	Deep Learning	KD-ELECTRA (Chrome extension)	450k URLs	Transformer embeddings
[24]	Hybrid ML–DL	UPADM framework	Large dataset	Multi-modal
[25]	Hybrid Machine Learning	LR, SVC, DT, RF, GBM, KNN + proposed LSD (LR+SVC+DT voting)	~11,000 phishing & legitimate URLs (UCI-style)	URL-based + handcrafted features
[26]	Deep Learning (Hybrid)	CNN + LSTM (IPDS)	~1M URLs + >10k webpage images	URL + HTML text + images + frames
[27]	Deep Learning	RF, J48, FilteredClassifier, 24 ML models	Two datasets (11,055 instances; 1,353 instances)	Feature-based (30 & 9 engineered features)
[28]	Deep Learning (Survey)	CNN, LSTM, RNN, Attention-based DL	Multiple datasets (review paper)	Text, email structure, embeddings
[29]	Large Language Model (LLM)	LLaMA (character-level URL modeling)	>2 million URLs	URL character sequences (embedding-based)

ML = Machine Learning, DL = Deep Learning, ML–DL = Hybrid Machine Learning and Deep Learning, LR = Logistic Regression, SVM = Support Vector Machine, RF = Random Forest, DT = Decision Tree, GB = Gradient Boosting, XGB = Extreme Gradient Boosting, CNN = Convolutional Neural Network, LSTM = Long Short-Term Memory, NN = Neural Network, SEM = Structural Equation Modeling, HEFS = Hybrid Ensemble Feature Selection, KD-ELECTRA = Knowledge Distillation–based ELECTRA Model, UPADM = Unified Phishing Attack Detection Model, URL = Uniform Resource Locator, F1 = F1-score.

The preprocessing pipeline consisted of several critical steps to ensure data quality and consistency. Initially, duplicate entries were identified and removed using hash-based deduplication techniques. Subsequently, broken links and URLs with missing components were filtered out through accessibility validation. Character normalization was applied to standardize the URL format, removing unnecessary whitespace and special characters that could introduce noise in feature extraction. Each URL was assigned a binary label, where 1 denotes phishing and 0 represents legitimate URLs. The preprocessed dataset structure is depicted in Figure 2.

ISSN: 2961-5992

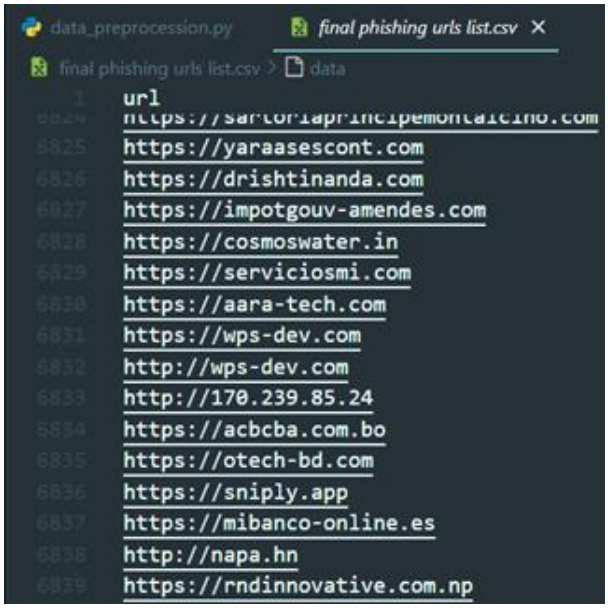

Fig. 2: Dataset after preprocessing

B. Feature Engineering

Feature extraction constitutes a crucial phase in transforming raw URL strings into quantifiable attributes suitable for machine learning algorithms. The feature engineering process encompassed four distinct categories, each capturing different behavioral characteristics of URLs.

- Lexical Features: This category extracts structural properties including URL length, count of digits, number of hyphens, presence of special characters, and token distribution. These features effectively capture obfuscation techniques commonly employed in phishing attacks.
- Character Ratio Features: Statistical measures such as vowel-to-character ratio, consonant-to-character ratio, digit-to-character ratio, and entropy-based randomness metrics were computed. These ratios help identify randomly generated domains and unusual character distributions characteristic of malicious URLs.
- Domain and Security Features: Security-related attributes including HTTPS protocol usage, top-level domain (TLD) classification, presence of IP-based domains, and suspicious domain patterns were extracted. These features leverage domain registration and security certificate information to assess URL legitimacy.
- Keyword and Brand Features: Semantic analysis was performed to detect phishing-indicative keywords such as "login," "verify," "update," and "secure." Additionally, brand imitation detection and brand similarity scoring algorithms were implemented to identify spoofing attempts targeting well-known organizations.

C. Model Selection and Training

1) Algorithm Evaluation

Twelve machine learning algorithms were evaluated to identify the optimal classifier for phishing detection. The evaluated models included Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, Naive Bayes, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and XGBoost, among others. All models were trained and evaluated under identical conditions to ensure fair comparison.

2) Data Partitioning

The dataset was partitioned using an 80-20 train-test split ratio. This ratio was selected to allocate sufficient training samples for model learning while retaining an adequate test set for unbiased performance evaluation. The stratified splitting technique ensured that both subsets maintained the original 50-50 distribution of phishing and legitimate URLs.

3) Baseline Training Protocol

The training protocol involved loading the engineered feature matrix (X) and corresponding label vector (y), followed by systematic training of each model on the training subset. Predictions were generated on the test subset, and comprehensive performance metrics were computed including Accuracy, Precision, Recall, F1-score, ROC-AUC, Log Loss, and Matthews Correlation Coefficient (MCC). Algorithm-specific parameters were configured as follows: maximum iterations set to 1000 for Logistic Regression convergence, probability estimation enabled for SVM to support probabilistic predictions, and log loss specified as the evaluation metric for XGBoost to suppress convergence warnings.

D. Hyperparameter Optimization

Following preliminary evaluation, XGBoost demonstrated superior performance and was selected for extensive hyperparameter tuning. The optimization process aimed to enhance predictive performance and generalization capability through systematic parameter space exploration.

1) Optimization Strategy

RandomizedSearchCV was employed for efficient hyperparameter search, executing 100 iterations with 5-fold cross-validation. The optimization process was conducted on Google Colab infrastructure to leverage computational resources. The dataset partitioning was modified to a 70-15-15 split for training, validation, and testing respectively, providing stable results while maintaining statistical representativeness in both validation and test sets.

2) Parameter Space

The hyperparameter search space encompassed the following ranges:

- n_estimators: [200, 1500] - controlling ensemble size
- max_depth: [3, 12] - limiting tree complexity
- learning_rate: [0.001, 0.3] - adjusting gradient descent step size
- subsample: [0.6, 1.0] - fraction of samples per tree
- colsample_bytree: [0.6, 1.0] - fraction of features per tree
- min_child_weight: [1, 10] - minimum sum of instance weight in child
- gamma: [0.0001, 1.0] - minimum loss reduction for splitting
- reg_alpha: [0.000001, 0.1] - L1 regularization term
- reg_lambda: [0.001, 1.0] - L2 regularization term

ISSN: 2961-5992

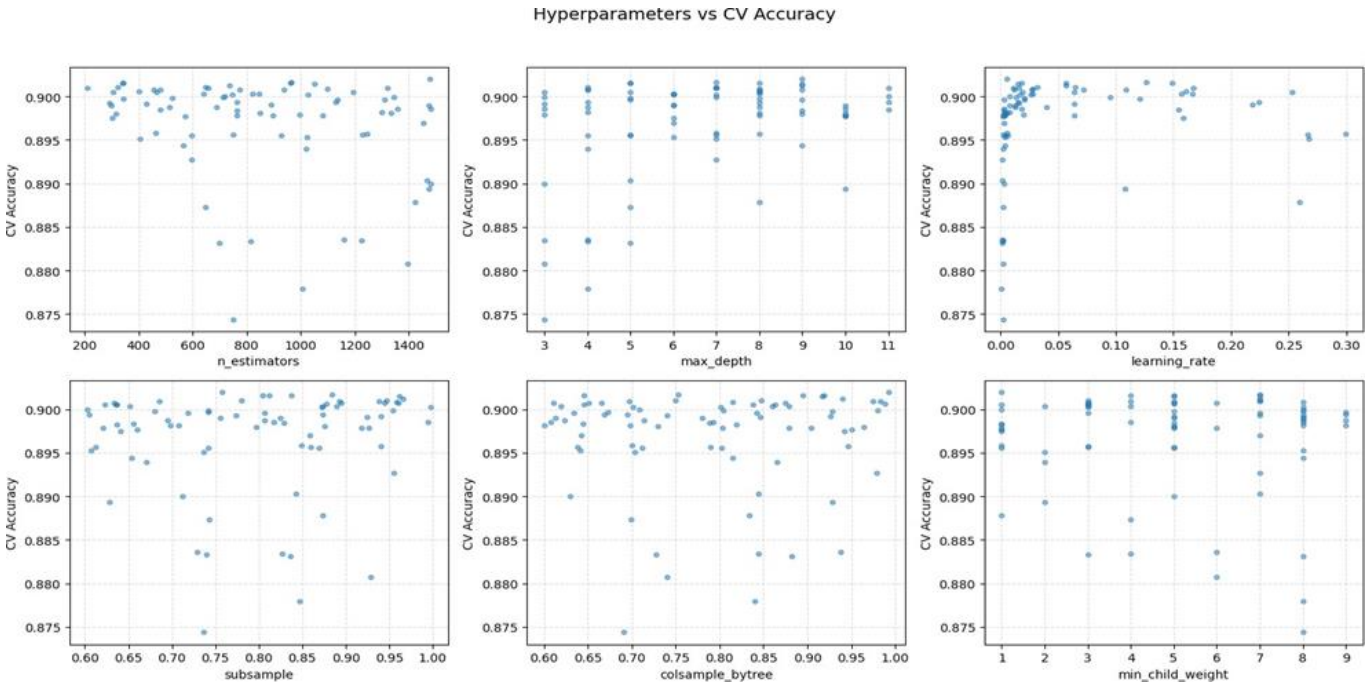

Fig. 3: Hyperparameters vs mean_test_score (accuracy)

3) Optimal Configuration

The optimization process converged to the following optimal hyperparameter configuration: n_estimators = 1479, max_depth = 9, learning_rate = 0.0057, colsample_bytree = 0.99, min_child_weight = 1, and gamma = 0.0002. This configuration achieved a mean 5-fold cross-validation accuracy of 0.90199, demonstrating consistent performance across all folds. The relationship between hyperparameter combinations and test accuracy is visualized in Figure 3.

4) Threshold Optimization

Classification threshold analysis was conducted across the range [0.05, 0.95] to identify the optimal decision boundary. Various performance metrics were evaluated at each threshold point, as illustrated in Figure 4, enabling selection of a threshold that balances precision and recall according to application requirements.

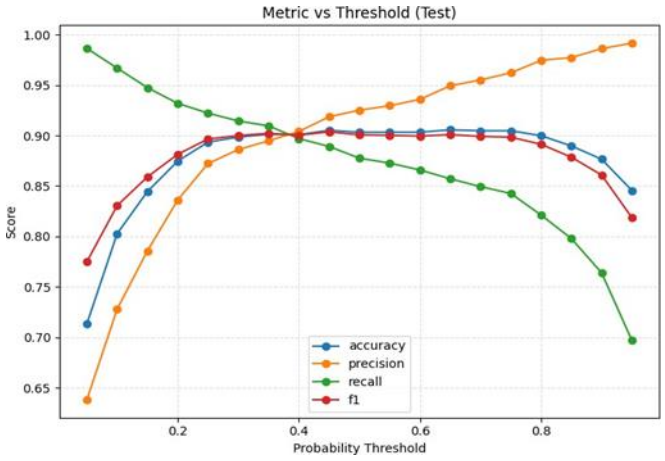

Fig. 4: Metric vs Threshold chart

E. Feature Importance Analysis

XGBoost's intrinsic feature importance metrics provided insights into feature contributions to the model's decision-making process. Three important measures were analyzed:

- Weight-based Importance: Features such as entropy_domain, brand_similarity, vowel_ratio, and consonant_ratio exhibited the highest weights, indicating frequent usage in tree construction (Figure 6).
- Cover-based Importance: The analysis revealed that has_hyphen,num_hyphens, and contains_phishing_words demonstrated maximum coverage values, suggesting their significant role in sample classification.
- Gain-based Importance: Features has_hyphen and num_hyphens achieved the highest gain scores, indicating their substantial contribution to prediction accuracy improvements.

The convergence of these importance metrics confirms that structural anomalies (hyphens), domain randomness (entropy), and brand imitation attempts constitute the most discriminative features for phishing detection.

F. Model Evaluation and Comparison

Comprehensive performance evaluation was conducted on all twelve machine learning algorithms using identical feature sets and data splits. Each model generated both categorical predictions and probability estimates for the positive class where applicable. The evaluation metrics encompassed Accuracy, Precision, Recall, F1-score, ROC-AUC, and Log Loss.

G.System Deployment and Integration

The deployment architecture comprises a Flask-based REST API backend and a Google Chrome browser extension frontend, enabling real-time phishing detection during web browsing.

ISSN: 2961-5992

1) Backend API Development

The Flask framework was selected for API development due to its lightweight architecture and rapid deployment capabilities. The API loads the trained model upon initialization and exposes an endpoint for URL feature evaluation. Upon receiving a feature vector from the browser extension, the API performs real-time inference and returns a classification result with associated confidence scores.

2) Browser Extension Integration

The Chrome extension operates as a background service that monitors navigation events. When a user initiates navigation to a new URL, the extension extracts the predefined feature set from the URL string and transmits it to the Flask API via an asynchronous HTTP POST request. The extension processes the API response and, in cases where the URL is classified as phishing with high confidence, displays an immediate warning alert to prevent page loading. This architecture ensures continuous, transparent protection without requiring explicit user interaction, effectively functioning as an autonomous security layer during web browsing sessions.

3) Real-time Processing Pipeline

The complete processing pipeline operates within milliseconds, ensuring minimal latency impact on browsing experience. Feature extraction occurs client-side within the extension, while model inference is performed server-side on the Flask API. This distributed architecture balances computational efficiency with model sophistication, enabling deployment of complex machine learning models without compromising browser performance.

	Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	Log Loss	MCC
0	Random Forest	0.905612	0.922786	0.885901	0.903967	0.957064	0.417572	0.811897
1	Logistic Regression	0.887755	0.921136	0.848837	0.883510	0.944585	0.291750	0.777973
2	Decision Tree	0.892128	0.919907	0.859738	0.888805	0.898900	2.893162	0.785988
3	KNN	0.896137	0.922541	0.865552	0.893138	0.942237	1.200202	0.793834
4	XGBoost	0.908528	0.933693	0.880087	0.906098	0.961717	0.239605	0.818442
5	Naive Bayes	0.845117	0.940686	0.737645	0.826884	0.928356	2.394767	0.707173
6	SVM	0.857872	0.918506	0.786337	0.847298	0.924796	0.347074	0.723427
7	Gradient Boosting	0.901239	0.925983	0.872820	0.898616	0.958239	0.258026	0.803843
8	AdaBoost	0.873178	0.900312	0.840116	0.869173	0.938894	0.550520	0.748086
9	Bagging	0.892857	0.907380	0.875727	0.891272	0.948711	0.802794	0.786216
10	Ridge Classifier	0.885933	0.930364	0.835029	0.880123	0.500000	18.074368	0.776039
11	SGD Classifier	0.839650	0.798469	0.909884	0.850543	0.500000	18.074368	0.685922

Fig. 5: Model testing result table

IV. Results and Discussion

This section presents the experimental results obtained from evaluating twelve machine learning algorithms for phishing URL detection, followed by an analysis of the deployed browser extension's real-time performance. The comprehensive evaluation encompasses classification accuracy, computational efficiency, and practical deployment validation.

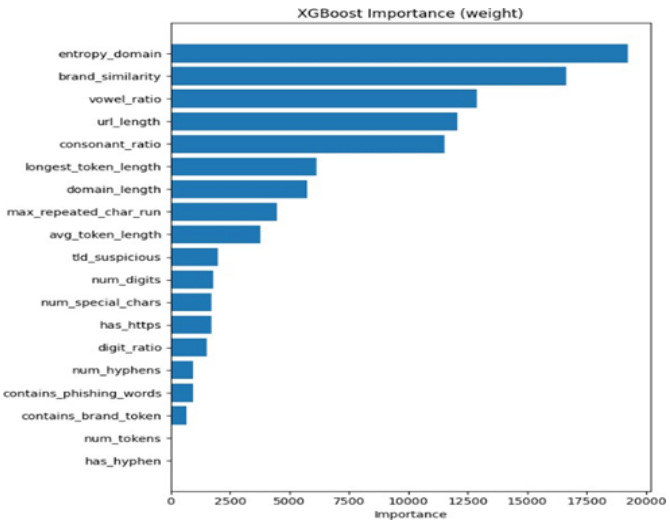

Fig. 6: XGBoost Importance (weight)

A. Model Performance Evaluation

Twelve classification algorithms were evaluated under identical experimental conditions to ensure fair comparison. The evaluated models include Random Forest, Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), XGBoost, Naive Bayes, Support Vector Machine (SVM), Gradient Boosting, AdaBoost, Bagging Classifier, Ridge Classifier, and Stochastic Gradient Descent (SGD) Classifier. Each model was trained on the engineered feature set and evaluated using standard classification metrics.

1) XGBoost: Selected Model

Based on comprehensive performance analysis, XGBoost was selected as the optimal classifier for deployment. The model achieved an accuracy of 90.85%, demonstrating robust classification capability across both phishing and legitimate URL classes. The precision score of 93.36% indicates high reliability in positive predictions, minimizing false alarms that could negatively impact user experience. The F1-score of 0.90 reflects balanced performance between precision and recall, which is critical for phishing detection systems where both false positives and false negatives carry significant consequences

While Random Forest exhibited competitive performance with metrics closely approaching XGBoost, the latter demonstrated superior balance between precision and recall among the evaluated classifiers, making it more suitable for real-time deployment scenarios. Although higher accuracy values have been reported in the literature, many such approaches employ computationally intensive models or offline evaluation settings; in contrast, this work prioritizes low-latency, lightweight deployment suitable for browser extensions.

2) Comprehensive Performance Metrics

Table II presents the detailed performance metrics for the selected XGBoost model across seven key evaluation criteria. The model achieved an accuracy of approximately 90%, indicating correct classification in nine out of ten URLs. The precision of 93% demonstrates that when the system flags a URL as phishing, it is correct 93% of the time, thereby

ISSN: 2961-5992

maintaining user trust through minimal false warnings. The recall value of 88% indicates that the system successfully identifies 88% of actual phishing URLs, providing substantial protection coverage.

The F1-score of 0.90 represents the harmonic mean of precision and recall, confirming balanced performance across both metrics. The ROC-AUC score of 0.96 demonstrates excellent discrimination capability between classes, with the model effectively separating phishing and legitimate URLs across various threshold settings. The Log Loss value of 0.23 indicates well-calibrated probability estimates, essential for threshold-based decision making. Finally, the Matthews Correlation Coefficient (MCC) of 0.81 signifies strong correlation between predictions and actual labels, accounting for class imbalance and providing a robust performance measure.

TABLE II
PERFORMANCE METRICS OF THE SELECTED XGBOOST MODEL

Metric	Value (≈)
Accuracy	90%
Precision	93%
Recall	88%
F1-Score	0.90
ROC-AUC	0.96
Log Loss	0.23
MCC	0.81

3) Real-time Performance Analysis

Real-time testing demonstrated that the integrated system achieves URL classification within approximately 0.3 to 0.5 seconds from the moment of page load initiation. This latency encompasses feature extraction within the browser extension, network transmission to the Flask API backend, model inference, and result transmission back to the extension. Although domain registration and security-related attributes are considered during feature design, the proposed system does not perform live WHOIS or SSL/TLS queries during real-time detection. All real-time decisions are made using URL-derived features that can be extracted locally and processed immediately. Network-bound operations such as WHOIS or certificate retrieval are excluded from the inference path and therefore do not contribute to runtime latency. The total response time remains consistently under one second from initial page load to on-screen warning display, meeting the stringent performance requirements for practical deployment without significantly degrading user browsing experience.

B. Browser Extension Deployment Results

The trained XGBoost model was successfully deployed through a Flask-based REST API integrated with a Google Chrome browser extension, enabling real-time phishing detection during normal web browsing activities.

1) System Architecture and Workflow

The deployment architecture consists of two primary components operating in tandem. The Chrome extension functions as the client-side monitoring agent, intercepting navigation events and extracting URL features using the predefined feature engineering pipeline. Upon feature extraction, the extension transmits the feature vector to the Flask API backend via asynchronous HTTP POST request, ensuring non-blocking operation that maintains browser responsiveness.

The Flask API backend loads the trained XGBoost model upon initialization and maintains it in memory for rapid inference. When receiving a feature vector, the API performs classification inference and returns a structured JSON response containing the prediction label (phishing or legitimate) along with associated confidence scores. The extension processes this response and triggers appropriate user interface actions based on the classification result.

2) User Interface and Warning Mechanism

When a user navigates to a URL classified as phishing with high confidence, the extension immediately displays a warning message, as illustrated in Figure 7. The warning interface is designed to be prominent yet non-intrusive, clearly communicating the detected threat while providing options for the user to proceed at their own discretion or navigate away to safety. This approach balances security with user autonomy, acknowledging that false positives may occasionally occur while ensuring users are informed of potential risks.

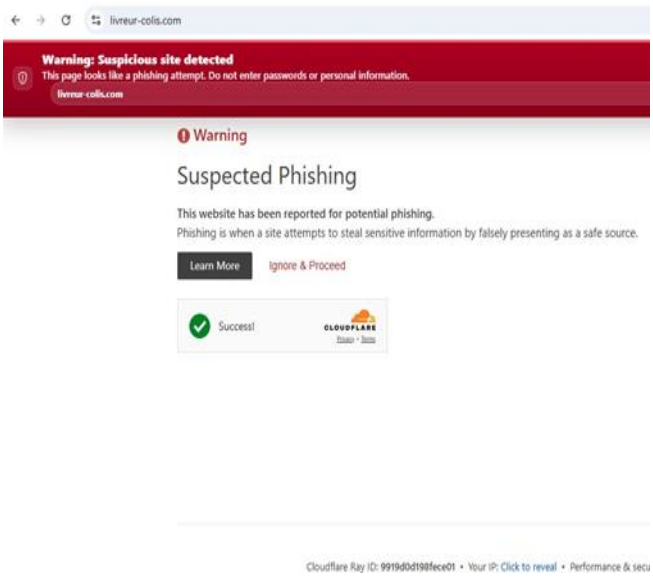

Fig. 7: Phishing site detected by browser extension

The automatic detection and warning mechanism operates transparently in the background, requiring no manual activation or configuration from users. This zero-interaction security model significantly enhances protection coverage, particularly benefiting users who may lack the technical expertise to manually identify phishing attempts.

ISSN: 2961-5992

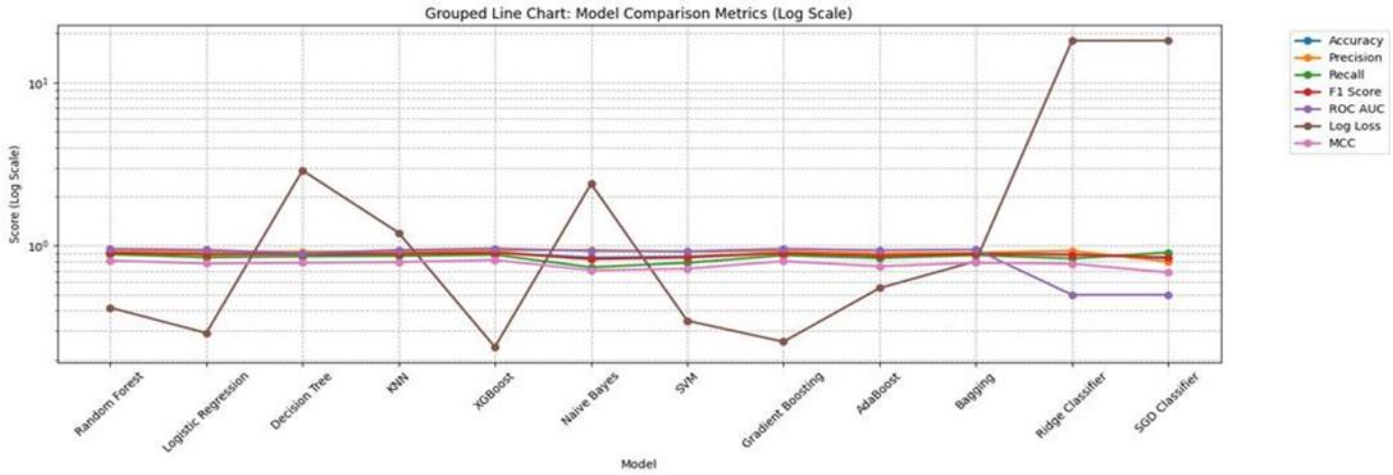

Fig. 8: Grouped line chart log scale

C. Deployment Validation

Field testing of the deployed extension across diverse browsing scenarios confirmed consistent performance across various URL types and website categories. The system successfully identified known phishing URLs from test datasets while maintaining low false positive rates on legitimate websites. The sub-second response time ensures seamless integration into normal browsing workflows, with users experiencing negligible latency impact. The lightweight architecture of both the extension and API ensures scalability, with the system capable of handling multiple concurrent requests without performance degradation.

D. Analysis and Interpretation

The experimental results demonstrate that machine learning-based approaches, particularly ensemble methods like XGBoost, provide effective solutions for real-time phishing detection. The achieved performance metrics indicate practical feasibility for real-time deployment, with accuracy and precision levels sufficient for practical deployment while maintaining acceptable recall rates. The successful integration of the trained model with a browser extension validates the feasibility of deploying sophisticated machine learning models in resource-constrained environments, with the sub-second response time confirming that complex feature extraction and model inference can be performed efficiently without compromising user experience.

The system's effectiveness can be attributed to the comprehensive feature engineering approach that captures lexical, statistical, domain-based, and semantic characteristics of URLs. The combination of multiple feature categories enables the model to identify phishing attempts through various attack patterns, from domain obfuscation techniques to brand impersonation strategies. However, the 88% recall rate indicates that approximately 12% of phishing URLs may evade detection, representing a limitation inherent to pattern-based classification approaches. Additionally, the system's performance depends on the representativeness of the training data, with novel phishing strategies not represented in the training set potentially challenging the model's generalization capability. Despite these constraints, the achieved balance between detection accuracy and computational efficiency demonstrates the viability of the proposed approach for real-world deployment as a practical phishing protection mechanism.

V. Limitations and Future Work

While the proposed system demonstrates effective real-time phishing detection with sub-second response times, several limitations should be acknowledged. First, the dataset used in this study represents a time-bounded snapshot collected from publicly available phishing and legitimate URL repositories. The evaluation employed random train–test splits, which enable fair model comparison but do not explicitly assess temporal generalization against newly emerging phishing URLs or concept drift over time. Second, although domain and security-related characteristics are considered at the feature design level, the real-time detection pipeline relies primarily on URL-derived lexical and structural features to maintain low latency. Live network-dependent enrichment sources such as WHOIS or SSL/TLS queries are excluded from the inference path, which may limit access to certain contextual signals. Finally, the current deployment architecture performs inference on a backend server, requiring transmission of URL-level data. Although data exposure is minimized and secured, this design introduces inherent privacy considerations. Future work will focus on evaluating temporal robustness using chronologically ordered datasets, exploring periodic model retraining to address evolving phishing strategies, and investigating privacy-enhanced deployment alternatives, such as lightweight edge-based inference within the browser extension.

VI. Conclusion

This research presented a real-time phishing URL detection system utilizing machine learning integrated with a browser extension. A dataset of 13,716 balanced URLs was collected from trusted sources and processed through a comprehensive feature engineering pipeline encompassing lexical, statistical, domain-based, and semantic attributes. Systematic evaluation of twelve classification algorithms identified XGBoost as the optimal model, achieving 90.85% accuracy, 93.36% precision, and an F1-score of 0.90. The system was successfully deployed through a Flask API backend connected to a Google Chrome

ISSN: 2961-5992

extension, enabling automatic phishing detection with sub-second response times during normal browsing activities. Real-time testing validated the system's capability to identify malicious URLs and display warnings without manual user intervention, demonstrating the feasibility of deploying sophisticated machine learning models in resource-constrained browser environments. While the dataset size and pattern-based approach present limitations for novel phishing techniques, the achieved balance between detection accuracy and computational efficiency confirms the viability of browser-based machine learning security solutions for practical cybersecurity applications.

REFERENCES

[1] S. Abad and H. Gholamy, "Evaluation of machine learning models for classifying malicious URLs," 2023.

[2] A. Baiomy, M. Mostafa, and A. Youssif, "Anti-Phishing Game Framework to Educate Arabic Users: Avoidance of URLs Phishing Attacks," *Indian J. Sci. Technol.*, vol. 12, no. 44, pp. 01–10, 2019, doi: 10.17485/ijst/2019/v12i44/147850.

[3] J. A. Ochu, G. I. O. Aimufua, H. Musa, and S. E. Chaku, "Detecting phishing websites using large language model," *Sci. World J.*, vol. 20, no. 2, pp. 692–697, 2025, doi: 10.4314/swj.v20i2.33.

[4] K. Subashini and V. Narmatha, "Detecting Phishing Websites using recent Techniques: A Systematic Literature Review," *ITM Web Conf.*, vol. 57, p. 01008, 2023, doi: 10.1051/itmconf/20235701008.

[5] M. Aljabri *et al.*, "Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions," *IEEE Access*, vol. 10, pp. 121395–121417, 2022, doi: 10.1109/ACCESS.2022.3222307.

[6] C. M. Igwilo and V. T. Odumuyiwa, "Comparative Analysis of Ensemble Learning and Non-Ensemble Machine Learning Algorithms for Phishing URL Detection," *FUOYE J. Eng. Technol.*, vol. 7, no. 3, pp. 305–312, 2022, doi: 10.46792/fuoyejet.v7i3.807.

[7] S. M. Wajid, T. Javed, and M. Mohd Su'ud, "Ensemble Learning-Powered URL Phishing Detection: A Performance Driven Approach," *J. Informatics Web Eng.*, vol. 3, no. 2, pp. 134–145, Jun. 2024, doi: 10.33093/jiwe.2024.3.2.10.

[8] S. M. Alshahrani, N. A. Khan, J. Almalki, and W. Al Shehri, "URL Phishing Detection Using Particle Swarm Optimization and Data Mining," *Comput. Mater. Contin.*, vol. 73, no. 3, pp. 5625–5640, 2022, doi: 10.32604/cmc.2022.030982.

[9] N. Koppula, V. P. Ganti, and P. Gandhe, "URL Based Phishing Detection," *Int. J. Recent Technol. Eng.*, vol. 9, no. 1, pp. 1872–1875, May 2020, doi: 10.35940/ijrte.A2657.059120.

[10] G. Varshney, A. Raj, D. Sangwan, S. Abuadbba, R. Mishra, and Y. Gao, "A login page transparency and visual similarity-based zero-day phishing defense protocol," *Comput. Secur.*, vol. 158, no. 2007, 2025, doi: 10.1016/j.cose.2025.104598.

[11] P. Preeti and P. Sharma, "Evolving strategies in anti-phishing: an in-depth analysis of detection techniques and future research directions," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 37, no. 3, p. 1726, 2025, doi: 10.11591/ijeecs.v37.i3.pp1726-1733.

[12] O. C. Angel, P. Melanie, P. Cristhian, R. Padilla-Vega, and C. José, "Analyzing website characteristics and their impact on web traffic and legitimacy classification for phishing detection: A structural equation modeling approach," *Issues Inf. Syst.*, vol. 26, no. 2, pp. 150–161, 2025, doi: 10.48009/2_iis_112.

[13] K. Omari, "Comparative Study of Machine Learning Algorithms for Phishing Website Detection," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 9, pp. 417–425, 2023, doi: 10.14569/IJACSA.2023.0140945.

[14] sopnil nepal, hemant gurung, and roshan nepal, "Phishing URL Detection Using CNN-LSTM and Random Forest Classifier," Nov. 01, 2022. doi: 10.21203/rs.3.rs-2043842/v2.

[15] H. R. Alavala, S. Singh, P. Joshi, and S. Basavaraju, "Enhancing Malicious URL Detection with Advanced Machine Learning Techniques," *1st Int. Conf. Adv. Comput. Sci. Electr. Electron. Commun. Technol. CE2CT 2025*, vol. 11, no. 8, pp. 151–156, 2025, doi: 10.1109/CE2CT64011.2025.10939290.

[16] A. Bharambe, "A Neural Network-Based Detection of Phishing URLs using Embeddings," *SSRN Electron. J.*, 2025, doi: 10.2139/ssrn.5380655.

[17] Y. S Tambe, "Phishing URL Detection Using Machine Learning," *J. Adv. Res. Prod. Ind. Eng.*, vol. 10, no. 01, pp. 1–5, Sep. 2023, doi: 10.24321/2456.429X.202301.

[18] S. H. Nallamala, K. Namitha, K. Raviteja, K. S. Sumanth, and J. S. Kota, "Phishing URL Detection using Machine Learning," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 12, no. 3, pp. 1984–1995, Mar. 2024, doi: 10.22214/ijraset.2024.59261.

[19] B. K. Gontla, P. Gundu, P. J. Uppalapati, K. Narasimharao, and S. M. Hussain, "A Machine Learning Approach to Identify Phishing Websites: A Comparative Study of Classification Models and Ensemble Learning Techniques," *EAI Endorsed Trans. Scalable Inf. Syst.*, vol. 10, no. 5, pp. 1–9, 2023, doi: 10.4108/eetsis.vi.3300.

[20] A. Al-qasmi, A. Al-anazi, L. Al-shehri, S. A-lshaman, and W. Al-atawi, "INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING Machine Learning-Based Phishing Detection System," vol. 12, no. 4, pp. 4367–4372, 2024.

[21] P. Janarthanan, K. Pratiba, S. SriS, and B. Yashashwini, "EFFECTIVE DETECTION OF MALICIOUS URLS USING VARIOUS MACHINE LEARNING TECHNIQUES," 2022. [Online]. Available: www.irjmets.com

[22] S. V. Oprea and A. Bâra, "Detecting Malicious Uniform Resource Locators Using an Applied Intelligence Framework," *Comput. Mater. Contin.*, vol. 79, no. 3, pp. 3827–3853, 2024, doi: 10.32604/cmc.2024.051598.

[23] K. S. Jishnu and B. Arthi, "Real-time phishing URL detection framework using knowledge distilled ELECTRA," *Automatika*, vol. 65, no. 4, pp. 1621–1639, 2024, doi: 10.1080/00051144.2024.2415797.

[24] S. Iftikhar, O. A. Abdulkader, and B. A. A. R. Al-Ghamdi, "UPADM - A Novel URL Phishing Attack Detection Model based on Machine Learning and Deep Learning Algorithms," *Int. J. Cyber Criminol.*, vol. 18, no. 1, pp. 244–260, 2024, doi: 10.5281/zenodo.4766814.

[25] A. Karim, M. Shahroz, K. Mustofa, S. B. Belhaouari, and S. R. K. Joga, "Phishing Detection System Through Hybrid Machine Learning Based on URL," *IEEE Access*, vol. 11, pp. 36805–36822, 2023, doi: 10.1109/ACCESS.2023.3252366.

[26] M. A. Adebowale, K. T. Lwin, and M. A. Hossain, "Intelligent phishing detection scheme using deep learning algorithms," *J. Enterp. Inf. Manag.*, vol. 36, no. 3, pp. 747–766, 2023, doi: 10.1108/JEIM-01-2020-0036.

[27] R. Alazaidah *et al.*, "Website Phishing Detection Using Machine Learning Techniques," *J. Stat. Appl. Probab.*, vol. 13, no. 1, pp. 119–129, 2024, doi: 10.18576/jsap/130108.

[28] K. Thakur, M. L. Ali, M. A. Obaidat, and A. Kamruzzaman, "A Systematic Review on Deep-Learning-Based Phishing Email Detection," *Electron.*, vol. 12, no. 21, pp. 1–26, 2023, doi: 10.3390/electronics12214545.

[29] O. K. Sahingoz, E. Buber, and E. Kugu, "DEPHIDES: Deep Learning Based Phishing Detection System," *IEEE Access*, vol. 12, no. January, pp. 8052–8070, 2024, doi: 10.1109/ACCESS.2024.3352629.

ISSN: 2961-5992