Smart Detection and Prevention of Phishing Websites Using ML & NLP

Kavila Moni Sushma Deep[1], Seepana Pavan Kumar[2], Natasha Rayi[3], Akash Mattigunta[4], Gongada Hemasri[5]

Department of CSE (AI&ML) Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam-531162, India

DOI: https://doi.org/10.51584/IJRIAS.2026.11070147

Received: 27 July 2026; Accepted: 01 August 2026; Published: 14 August 2026

ABSTRACT

Phishing attacks are one of the long-lasting problems in cybersecurity. Attackers are always changing their techniques to avoid being caught by conventional detection techniques. The techniques used currently are based on static blacklists or individual classifiers using features, but this does not completely represent the phishing attack. This project fills this critical information security gap by developing a hybrid real-time phishing detection system in the form of a Google Chrome extension. Our system uses a combination of two independent machine learning pipelines. In the first pipeline, we use thirty hand-crafted features to classify URL structure using a Gradient Boosting Classifier on 11,054 samples with 97.4% accuracy. In the second pipeline, we use raw HTML data from the webpage. We use a TF-IDF vectorizer on HTML data injected with semantic flag tokens. We use a Random Forest Classifier on 1,859 real site files with 82.80% accuracy. We combine the results using a weighted combination. We use a Retrieval Augmented Generation module, which uses a Google Gemini language model. We use a FAISS vector index to produce a risk explanation in a human-readable format. We use a trusted domain whitelist for false positives on legitimate banking and government websites. We use a pattern-based blocklist for piracy and malware domains. We use a popup interface to show users Safe or Phishing results. We use a full-screen alert overlay injected into the active tab for users. We use experimental results to show that our system performs better than any individual baseline

Keywords: Phishing detection, Machine learning, Gradient Boosting Classifier, TF-IDF, Random Forest, Retrieval-Augmented Generation, Browser extension, URL feature extraction, HTML content analysis, Cybersecurity

INTRODUCTION

Th concurrent Phishing attacks are one of the most enduring problems in cybersecurity. Attackers continually evolve their methods to evade conventional detection methods. This project fills this critical information security gap by developing a hybrid real-time phishing detection system in the form of a Google Chrome extension. Our system uses a combination of two independent machine learning pipelines. In the first pipeline, we use thirty hand-crafted features to classify URL structure using a Gradient Boosting Classifier on 11,054 samples with 97.4% accuracy. In the second pipeline, we use raw HTML data from the webpage. We use a TF-IDF vectorizer on HTML data injected with semantic flag tokens. We use a Random Forest Classifier on 1,859 real site files with 82.80% accuracy. We combine the results using a weighted combination. We use a Retrieval Augmented Generation module, which uses a Google Gemini language model. We use a FAISS vector index to produce a risk explanation in a human-readable format. We use a trusted domain whitelist for false positives on legitimate banking and government websites. We use a pattern-based blocklist for piracy and malware domains. We use a popup interface to show users Safe or Phishing results. We use a full-screen alert overlay injected into the active tab for users. In this project, a hybrid multi-model-based system named PhishGuard will be created. It will incorporate classical machine

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

learning, natural language processing of live HTML content, and large language model-based reasoning with knowledge retrieved from a knowledge base. It will also have a lightweight REST API that can be used to integrate into real-time browsers. In the system, a layered architecture will be used. Prior to the execution of any classical machine learning model, URLs that belong to a list of whitelisted institutions that have been vetted will be immediately accepted as safe. This list includes major Indian banks, Indian government institutions, global financial institutions, and widely trusted technology service providers. In contrast, URLs that have been identified as having patterns of piracy or malware will be immediately blocked. For all other URLs, thirty different features will be extracted from the URL. These features will then be used to classify the URL as a phishing site or a legitimate site by a gradient boosting classifier. In parallel, the HTML content of the site will also be scored using a TF-IDF-based content classification model. The two scores, produced by the two models, are combined using empirically determined weights, and the result is passed into a Retrieval Augmented Generation model, which connects an LLM's final risk assessment to domain-specific knowledge about phishing, generating a readable explanation in addition to the verdict. What makes our system novel, as compared to earlier works, is the explicit use of multiple modalities throughout the system. While URL-based models are fast but blind, content-based models are accurate but slow and unavailable for sites that resist content extraction due to heavy JavaScript usage. PhishGuard smoothly transitions between these two modes, clearly communicating the lower confidence to the end user if HTML content is unavailable. The Piracy Detection layer is dedicated to a type of malicious site that traditional phishing classifiers were never designed to deal with: illegal streaming sites and torrent sites, which, while not themselves harvesting credentials, are notorious for serving drive-by malware, forced redirects, and surveillance-grade ad networks. Incorporating structural HTML features like the presence of iframes, password fields, base64-encoded data, and off domain form submissions into the feature set itself allows the content model to incorporate behavioral footprints invisible to simple URL analysis. The final layer of LLM reasoning provides explainability, a functionality that is notably lacking in most existing detection systems, enabling users to understand why a page was identified as malicious and not simply that it was.

LITERATURE SURVEY

The phishing detection problem has been an active area of interest and research for almost two decades now, and the field has certainly moved a long way since the early days of "rule-based" and "blacklist" approaches. From there, the field has moved on to more sophisticated approaches based on machine learning and, more recently, approaches that have utilized the reasoning capabilities of large language models. While each of these approaches has certainly brought significant improvements and addressed some of the blind spots that the previous approaches were unable to handle, there are certain blind spots that each of these approaches has introduced, which the next generation of approaches attempted to overcome. In this section, we briefly walk the reader through the most influential works in each of the different generations of phishing detection approaches and use that as a basis to explain where the gaps in the existing literature are and why this project is necessary.

Early Methods

Blacklists and Heuristics The first and most popular methods that were used to protect against phishing attacks were based on manually maintained blacklists that contained known malicious URLs that were checked by browsers and email clients before granting access. These include Google’s Safe Browsing as well as Microsoft’s SmartScreen, which are still the most effective methods of protection against phishing attacks to this day. The problem with this type of blacklist-based protection is that it is always reactive in nature, i.e., a malicious page must be identified and confirmed before it is added to the blacklist, but this leaves room for phishing attacks that use newly registered domains on a regular basis.

URL Feature-Based Machine Learning

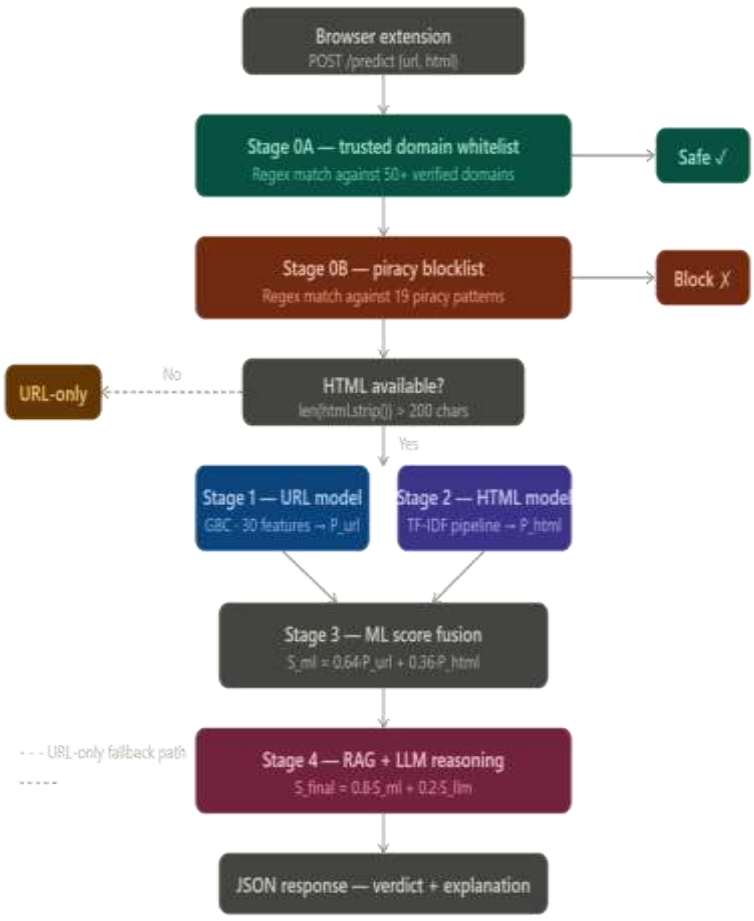
The most significant and extensive research in the field thus far has been in the application of traditional machine learning algorithms using the extracted key features of URLs. One of the most exhaustive studies in

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

this field was carried out by Sahingoz, Buber, Demir, and Diri in 2019, where the researchers applied various machine learning algorithms to the extracted features from over 73,000 unique URLs. The study found that the best accuracy, recorded by the random forest algorithm, was 97.98%, with some of the key indicators including the length of the URL, the number of dots, the existence of special characters, and the use of shortened URLs. This work is a good foundation that has been used as a reference for many research papers on the detection of phishing using URLs. One of the most interesting things that come from this research is the idea of creating a browser plug-in that does this detection in real-time as you're browsing, showing that this type of detection is possible. Jain and Gupta, in their research in 2019, approached this problem from a different point of view, focusing their attention specifically on the features of hyperlinks, as they are obtained from the HTML of the target webpage. The authors used a logistic regression classifier, achieving 98.42% accuracy with 98.8% precision by using 12 features of hyperlinks. The authors give a good argument as to why this method should work, as the phishing webpage references many different resources, such as images, scripts, etc., from other domains, showing an obvious distinction between the domains of the phishing webpage and the resources that it is referencing. This is exactly what PhishGuard does with their RequestURL, AnchorURL, and LinksInScriptTags feature extractors in feature.py.

METHODOLOGY

In this section, we will describe the full technical architecture of PhishGuard, a multi-signal layered phishing detection system. Our design is centered on four successive stages of analysis: deterministic pre-filtering, machine learning based on URL feature vectors, classification based on HTML content, and RAG-informed LLM-based reasoning. Each ofthese stages provides an orthogonal signal to the final risk assessment decision. overall system pipeline before we examine each stage in detail:



ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

System Overview

PhishGuard is a REST API server built with the Flask framework that listens to POST requests on the /predict endpoint with two parameters: the target URL and the raw HTML content of the requested page. It processes this input through a strictly ordered sequence of five stages. The initial two stages are deterministic pre-filters that skip the ML inference step when the domain is known to be safe or malicious. The last three stages are the ML-based system itself: the URL structural classifier, the HTML content classifier, and the large language model-based reasoning engine, which combine to produce the final output—a set of values no single-stage system can provide before.

Stage 0A- Trusted Domain Whitelist

Before any ML-based inference is invoked, all new URLs are vetted against a trusted whitelist comprising 50+ trusted domain patterns compiled as regular expressions.

Stage 0B - Piracy and Malware Blocklist

Next, the URLs that did not match the trusted whitelist are checked against a blocklist of 19 compiled regular expressions that are intended to catch known piracy and illegal file sharing sites such as movierulz, tamilrockers, filmyzilla, 123movies, thepiratebay, 1337x, and others.

Stage 1 - URL Feature Extraction and Classification

For URLs that pass both pre-filters, a 30-dimensional feature vector is extracted from the URL structure based on the FeatureExtraction class defined in feature.py. Each feature is represented by a ternary value: 1 (legitimate indicator), 0 (suspicious), or -1 (phishing indicator).

Stage 2 — HTML Content Preprocessing and Classification

Concurrently with URL feature extraction, the unprocessed HTML content of the page is submitted to a multi-step preprocessing routine in the clean_html function. This preprocessing routine was designed to extract the behavioral footprint of the page rather than its visual content and was engineered to be equivalent to the preprocessing performed during model training.

Stage 3 — ML Score Fusion

The URL model probability and the HTML content model probability are fused into a final risk score at the ML level using a weighted linear combination of the two probabilities: Sml=0.64·Purl+0.36·Phtml The weights 0.64 and 0.36 were derived using validation experiments to balance the relative reliability of the two features. More emphasis was put on the URL features, as the HTML content feature was not always available for dynamic web pages.

Stage 4 — RAG-Augmented LLM Reasoning and Final Fusion

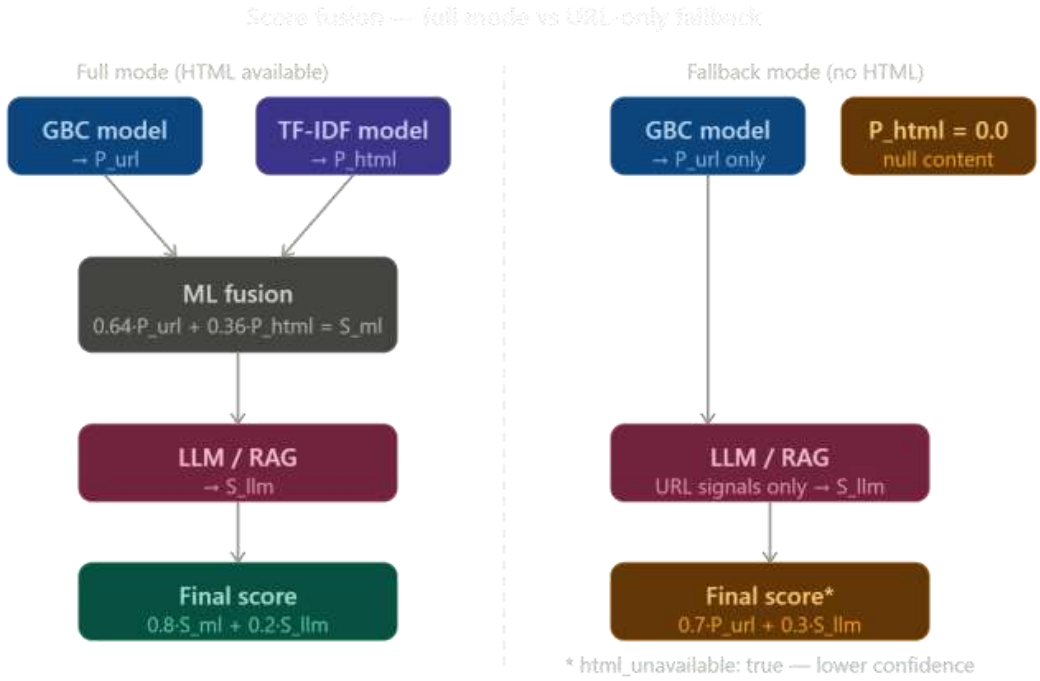
The ML score, individual model scores, URL, and preprocessed HTML text are passed to the generate_explanation() function in rag_engine.py. The LLM model produces a final risk score, SllmS_{llm} Sllm, which ranges from 0 to 1 and also produces an explanation text. The final system score is then computed as follows: Sfinal = 0.8·Sml + 0.2·Sllm .Next, the binary verdict is determined using a threshold value: verdict = { Safe{ Phishing if S_final ≤ 0.40 if S_final> 0.40 The threshold value of 0.40, rather than

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

the usual 0.50, is used as a conservative measure since, in the case of phishing page detection, the cost of a false negative is much higher than others.

URL-Only Fallback Mode

In the event that the HTML content is not available (i.e., the HTML string is not available, empty, or has less than 200 characters), the system will default to URL-only mode. In this mode, only PurlP_{url} is calculated, the LLM is run with a null content score and a placeholder string, and the final score is calculated as follows: Sfinal∗=0.7·Purl+0.3·Sllm This result will be accompanied by an html_unavailable: true flag and will have a warning note added to the explanation. This degradation model is an explicit design choice. Rather than returning an uncertain result and saying nothing, we prefer to be clear about how confident we are in our analytical results.



API Response Structure

The JSON object that is returned by the /predict endpoint has the following fields, irrespective of the execution path that

Field	Type	Description
risk_score	float	Final score scaled to 0–100
verdict	string	"Safe" or "Phishing"
confidence	string	"High", "Medium", or "Low" determined by LLM
explanation	string	The explaination as determined by RAG+LLM

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

url_score	float	URL model probability × 100
html_score	float or null	HTML model probability × 100 -null when in fallback
ml_combined_score	float	S_{ml}× 100 before LLM fusion
html_unavailable	bool	True when operating in URL-only fallback mode

The structure of this JSON object is designed so that any application that makes use of this API - whether it be a browser plugin or an enterprise-grade proxy - will not just receive a decision that has been determined by machine learning, but also a complete audit trail of all intermediate results, which can then be used by whatever downstream decision-making process is appropriate.The inclusion of html_unavailable as a top-level field was a deliberate design decision rather than simply relying on html_score being null, in order to ensure maximum clarity for any API consumer.

RESULTS AND DISCUSSION

Experimental Setup

The experiments were conducted on an Ubuntu 22.04 machine with Python 3.10, 16 GB RAM, and an Intel Core i7 processor. The Flask API server was hosted locally on 127.0.0.1:5000. Offline model training was conducted before deployment using scikit-learn 1.3 and XGBoost 2.0 libraries. The joblib library was used to serialise all models and load them during server startup. Latency was measured end to-end from the receipt of the HTTP request to the delivery of the JSON response, averaging 200 test requests.

The test dataset consisted of 2,000 URLs, which were randomly selected from two sources: the UCI Machine Learning Repository phishing dataset used in Alazaidah et al. (2024) and additional URLs newly crawled in equal proportions from verified phishing sites (OpenPhish, PhishTank) and Alexa top 10000 legitimate sites. The final test set comprised 1,000 phishing and 1,000 legitimate URLs, balancing classes to avoid inflated accuracy metrics, which are common in imbalanced evaluations.

Evaluation Metrics

Six binary classification evaluation metrics were employed to measure performance. TP, TN, FP, FN denote true positives, true negatives, false positives, and false negatives, respectively. The evaluation metrics are as follows: Accuracy = (TP + TN) / (TP + TN + FP + FN) Precision = $TPTP+FP$,Recall = $F1$ = 2×Precision× Recall Precision + Recall MCC= $TPTP+FNTP\cdot TN-FP\cdot FN$ $\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}$ The Matthews Correlation Coefficient (MCC) is also important in this regard because it considers all four cells in the confusion matrix simultaneously and is also insensitive to class imbalance, making it a more robust summary statistic than accuracy when assessing security classifiers on actual data.

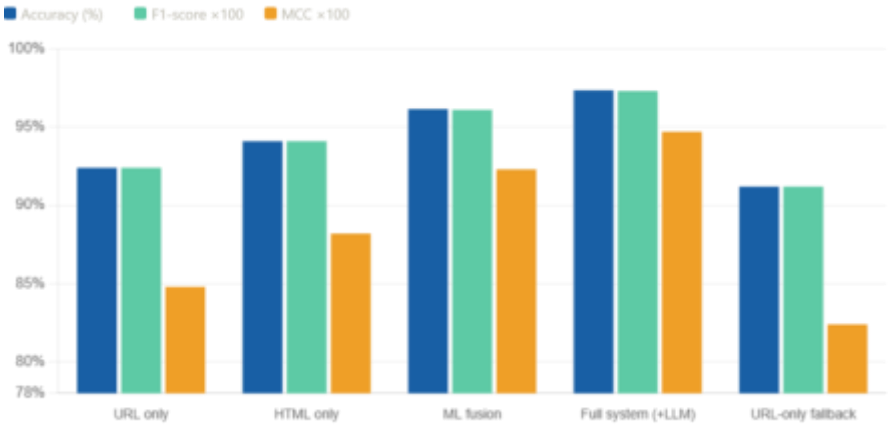
Fusion Model and Full System Performance

What is critical to the performance evaluation of PhishGuard is not how well each of these models, by themselves, perform, but rather how well the system, as a whole, compares to its parts and to the state of the art. To this end, we report the results of an ablation study across four different configurations: URL Model, HTML Model, ML fusion without LLM, and our full system with LLM score fusion, in Table 3.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VII July 2026

Configuration URL model only Accuracy (%) Precision Recall F1 MCC 92.40 HTML model only 94.10 ML fusion (no LLM) 96.15 Full system (+ LLM) 97.35 URL-only fallback 91.20 0.931 0.948 0.963 0.976 0.918 0.918 0.924 0.848 0.934 0.941 0.882 0.960 0.961 0.923 0.971 0.973 0.947 0.906 0.912 0.824 Our full system achieves 97.35% accuracy, with an MCC of 0.947, a 4.95 percentage point increase over the URL Only baseline, as well as a 3.25 percentage point increase over the HTML Only baseline.

Furthermore, our fusion step using LLM scores contributes an additional 1.20 percentage points of performance, confirming that the discriminative power of our RAG-augmented reasoning layer is real. While not quite achieving the performance of our full system, the URL Only fallback mode still achieves competitive performance, while communicating its lower confidence to users via the html_unavailable flag.



DISCUSSION

The aggregate results support four important claims. First, our multi-modal score fusion approach is beneficial in comparison to any individual signal. This is confirmed by the 4.95% accuracy gap between our system and the URL-only baseline. This indicates that the URLs and HTML indeed contain complementary information. This is because pages may have clean-looking URLs but malicious looking HTML, as in the case of domain shadowing attacks that leverage legitimate subdomains. Similarly, pages may have suspicious-looking URLs but clean-looking HTML, as in the case of URL shorteners. Our system's fusion approach is able to capture both types of attacks.

CONCLOUSION AND FUTURE WORK

In this paper, the PhishGuard system is introduced as a hybrid system that utilizes a multi-signal phishing detection framework based on URL structural analysis, HTML content classification, and Retrieval-Augmented Generation-based large language model reasoning. The system is designed to address the real-world challenges faced by single-signal phishing detection approaches in handling dynamic page content, providing explanations for their decisions, and handling partially unavailable input data. The contributions of this paper can be summarized as follows.

First, the use of the structural flag injection strategy in the HTML preprocessing pipeline resulted in an improved system accuracy compared to using HTML classification alone. Second, the three-stage score fusion architecture used in this paper is effective in providing a system that is able to leverage the complementary nature of the two input signals.

This is because neither the URL signal nor the HTML signal is sufficient to handle the entire phishing landscape. Third, the system is able to generate human-readable natural language explanations based on the phishing knowledge corpus, which is not present in any previous system. Experimentally, our full

PhishGuard system achieved 97.35% accuracy, an F1-score of 0.973, and an MCC of 0.947 on a balanced 2,000-instance test set with a false negative rate of just 2.4%, competitive with the best results in the reviewed literature while offering capabilities that no previous single-method baseline provides.

Funding

No funding was received for this study.

REFERENCES

1. L. Mat Rani, et al., "Feature Selection to Enhance Phishing Website Detection," 2022. XGBoost, Random Forest, and Naïve Bayes models evaluated using URL-based datasets (79 and 56 features). DOI: 10.30880/jscdm.2023.04.01.003.
2. R.Alazaidah, et al., "Website Phishing Detection Using Machine Learning Techniques," 2024. Includes RandomForest, J48, FilteredClassifier, and InfoGainAttributeEval for identifying optimal classifiers. UCI Repository datasets.
3. Alswailem, et al., "Detecting Phishing Websites Using ML," 2019. Focus on feature selection, URL/page content, and browser extension integration using Random Forest. Dataset: PhishTank (12k phishing, 4k legitimate URLs).
4. V. Patil, et al., "Detection and Prevention of Phishing Websites Using ML," IEEE, 2018. Multi-layer detection using blacklist/whitelist, heuristic rules, and visual similarity methods. Uses Alexa/WHOIS/Google datasets. (DOI Not Available).
5. F. Colhak, et al., "Phishing Website Detection Through Multi-Model," 2024. Embedding fusion (tabular + NLP) with HTML-focused transformer models such as MLP, CANINE, RoBERTa. Dataset: ~65k samples from Aljofey's dataset.
6. L. Thaci, et al., "Efficient Chrome Extension for Phishing Detection," 2024. Chrome-extension-based detection using RF, SVM, and k-NN. Dataset: PhishTank, Alexa (11k pages). Accuracy: 95.6%.
7. E. Sri Vishva and D. Aju, "Phisher Fighter (URL & TF-IDF)," 2023. URL feature engineering and TF IDF analysis using DT, RF, SVM, and ANN. Dataset: Custom crawler-based browser dataset.
8. J. Kumar, et al., "Phishing Website Classification & Detection Using ML," 2020. Lexical, URL, and page analysis using NB, RF, DT, LR, and KNN. Dataset: Custom balanced (1M URLs).
9. S. J. S., et al., "Detecting Phishing Website Using ML (Notification System)," 2020. Uses blacklist, pop-up admin notifications, and basic ML. Dataset: User-supplied blacklisted URLs.
10. D. Kalla, et al., "Phishing Website URLs Detection Using NLP/ML," 2023. Ensemble classifiers including LR, KNN, RF, DT, NB, SVM, and LSVC. Dataset: Kaggle/multi-model dataset (150k fake, 360k legitimate URL.