WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

AI-BASED PHISHING WEBSITE DETECTION USING MACHINE LEARNING

SHAHEEN BUHARI [1], M.S. EDWIN ARSHINA P.X[2]

[1] MASTER OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING,VELS INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)

[2]Senior Professor, Department of CSE School of Engineering,VISTAS, Chennai

ABSTRACT-

Phishing websites imitate legitimate online services to obtain credentials, financial details, and other sensitive information. This paper presents a machine-learning-based phishing detection prototype with a web application for submitting supported inputs and displaying predictions. The architecture separates offline model development from online inference through a saved Scikit-learn pipeline, a FastAPI service, and a React interface. The project design considers Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest as candidate classifiers. The training materials include a feature-based dataset containing 11,054 records and a larger URL and webpage dataset containing 235,795 records, together with processed representations containing 13 and 10 URL features. Dataset inspection establishes the available schemas, class coding, missing-value status, and repeated URL records. Functional assessment focuses on the normal application workflow: input submission, backend prediction, and result display. A manual demonstration confirmed the operation of this workflow. The contribution is the integration of labelled data, model-ready features, a prediction service, and a browser interface into a modular prototype. Further validation is required to establish its reliability against unseen phishing campaigns.

Key Terms—Phishing detection, supervised machine learning, URL features, classification, cybersecurity, web application.

INTRODUCTION

Phishing websites impersonate trusted online services to persuade users to disclose login credentials, banking details, and personal information. Attackers imitate page layouts and brand identities and manipulate website addresses to make fraudulent pages appear credible. The resulting risk affects both users and the organizations being impersonated. Detection therefore requires examination of measurable website characteristics beyond visual appearance [1].

Existing approaches include list-based filtering, heuristic analysis, page-similarity assessment, and machine learning. Blacklists identify previously reported malicious resources, but newly created or modified websites may remain unrecognized until their addresses are added. Heuristics examine predefined indicators, while similarity-based methods compare suspicious pages with legitimate counterparts. These approaches involve trade-offs in coverage, processing requirements, and maintenance [1], [2].

Supervised learning provides a complementary method by associating website features with known class labels. A classifier receives a numerical representation of a sample and produces a class prediction. Its practical use also depends on the surrounding application: the input must be converted into the expected features, the saved preprocessing must be applied, and the result must be returned clearly to the user.

This paper describes a prototype that connects machine-learning inference with a FastAPI backend and a React interface. The project design includes Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest. The emphasis is on dataset preparation, matching training and prediction schemas, and the functional operation of the application.

The objectives are to document the available training data, explain the modular architecture, and describe the working prediction path. The prototype provides a phishing risk indication. Its functional demonstration establishes application behaviour, while broader validation against independent website samples remains future work.

RELATED WORK

Research on phishing website detection examines both detection strategies and the factors affecting their performance. The studies reviewed here provide the basis for selecting candidate classifiers, defining the evaluation procedure, and identifying requirements for application integration. Table I summarizes their relevance to the proposed framework.

A. Detection Approaches and Feature Representation

The survey "Phishing or Not Phishing? A Survey on the Detection of Phishing Websites" [1] organizes detection methods into list-based, similarity-based, and machine-learning-based approaches. It identifies the reactive nature of blacklists as a limitation when addressing previously unseen phishing websites. The survey also discusses the trade-off between feature availability and collection cost: URL features can be obtained quickly, while page-content features require additional retrieval and processing. These findings support selecting features that can be reproduced reliably during prediction.

Safi and Singh [2] review list-based, visual-similarity, heuristic, machine-learning, and deep-learning techniques. Their analysis covers the algorithms, datasets, and feature representations used in phishing detection research. The breadth of these approaches highlights the importance of describing the complete detection procedure, including data preparation and feature construction, when interpreting model performance. For the present framework, this motivates explicit documentation of the input schema and evaluation conditions.

B. Comparative Classification Studies

The comparative study "A Comparative Analysis of Machine Learning-Based Website Phishing Detection Using URL Information" [3] evaluates Decision Tree, Random Forest, K-Nearest Neighbors, Gaussian Naïve Bayes, and XGBoost. Random Forest achieved the highest reported accuracy of 97.0% within that experiment. This result supports including Random Forest as a candidate classifier, but does not establish that it will perform best on a different dataset or feature representation. Model selection in the present work therefore depends on its own evaluation results.

Almujahid et al. [4] compare machine-learning and deep-learning methods using datasets from Mendeley and the UCI repository. Their evaluation includes Logistic Regression, Decision Tree, Random Forest, SVM, K-Nearest Neighbors, XGBoost, and neural-network models. The study examines feature engineering, class imbalance, and hyperparameter tuning, and reports CNN as its strongest-performing approach. Its use of accuracy, precision, recall, F1-score, and false-positive rate illustrates the value of evaluating several aspects of classification behaviour.

TABLE I

SUMMARY OF SELECTED RELATED WORK

Study	Focus	Main finding or contribution	Relevance to this framework
IEEE Access survey [1]	List-based, similarity-base d, and machine-learni ng detection	Reviews detection strategies and feature-collectio n trade-offs	Guides feature selection and system scope
Safi and Singh [2]	Systematic review of phishing detection techniques	Compares approaches, algorithms, and datasets	Supports methodologi cal documentati on
Compar ative URL-inf ormation study [3]	Five machine-learni ng classifiers	Reports Random Forest as the strongest classifier in its experiment	Supports its inclusion as a candidate model
Almujah id et al. [4]	Machine-learni ng and deep-learning comparison	Examines feature engineering, tuning, and multiple evaluation metrics	Guides comparative evaluation

C. Positioning of the Present Work

The reviewed studies motivate a framework that combines consistent model comparison with a defined prediction workflow. The present work focuses on Logistic Regression, Decision Tree, SVM, and Random Forest, evaluated using a common data partition and class-specific metrics. Its application design connects the selected pipeline to FastAPI and React, with particular attention to matching training and prediction features. Experimental findings will determine the selected classifier and the limits of its suitability for phishing risk detection.

PROPOSED SYSTEM

The proposed system combines supervised machine-learning classification with a web application for submitting supported inputs and viewing phishing risk predictions. Its architecture separates offline model development from online inference. This separation allows candidate classifiers to be evaluated before deployment and enables the application to use a saved model without retraining for each request.

A. System Architecture

The framework comprises four components: a data preparation and training module, a saved model pipeline, a FastAPI service, and a React interface. The training module prepares labelled samples and compares the candidate classifiers. The selected pipeline and its feature schema form the connection between model development and application inference. Fig. 1 illustrates these relationships. The system comprises a dataset-processing component, a

saved Scikit-learn pipeline, a FastAPI service, and a React interface. The saved pipeline and feature schema connect model development with runtime inference. Fig. 1 shows the processing stages and the loading of saved artifacts into the prediction service.

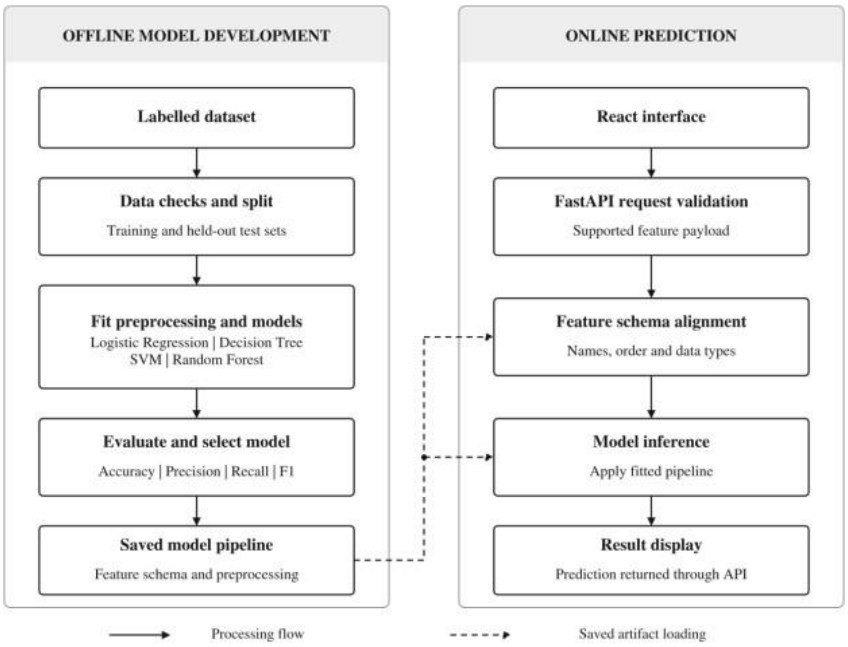


Fig. 1. Proposed architecture connecting offline model development with online prediction.

B. Offline Model Development

The offline stage begins with a labelled dataset containing legitimate and phishing website samples. Initial checks examine missing values, duplicate records, feature types, and class labels. The data are partitioned into training and held-out test sets, with preprocessing parameters learned exclusively from the training partition.

Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest are evaluated using a common experimental procedure. Each classifier is assessed through accuracy, precision, recall, F1-score, and a confusion matrix. Phishing is designated as the positive class when interpreting class-specific metrics.

Model selection considers the balance between detecting phishing samples and avoiding false alarms. The selected fitted pipeline is saved together with the expected feature names, ordering, and data types. These artifacts define the input requirements for online prediction.

C. Online Prediction and Application Integration

The React interface is designed to collect supported inputs, submit requests, indicate processing status, and display predictions or error messages. FastAPI validates each request and checks its compatibility with the saved feature schema. Validated feature values are arranged in the required order and passed to the fitted pipeline.

Two input modes are distinguished in the design. A feature-based request supplies the numerical attributes expected by the model. A URL-based request requires an implemented extractor that reproduces those same attributes from the submitted address. The reference endpoint in the source document supports precomputed features; URL extraction remains an integration requirement to be confirmed.

The backend returns a structured classification response to the interface. Missing features, invalid values, and unavailable model artifacts produce explicit errors. Prediction labels are presented as model assessments, with any optional score identified separately from a guarantee of website safety. This workflow establishes a consistent connection between the evaluated model and the user-facing application.

DATASET AND METHODOLOGY

A. Training Materials

The training materials supplied for the project comprise two source datasets and two processed versions of the larger dataset. Table II reports the row counts and available input attributes. Counts refer to the supplied files before any additional filtering.

TABLE II

SUPPLIED TRAINING DATA FILES

File	Records	Inputs
phishing.csv	11,054	30
url_phishing_dataset.csv	235,795	54
processed_features.csv	235,795	13
processed_url_features.csv	235,795	10

Input counts exclude the target label and an identifier column where present.

The larger source file matches the published structure and class distribution of the PhiUSIIL Phishing URL (Website) dataset [7]. It contains 235,795 URL and webpage records. Its 56 columns include the FILENAME identifier, 54 input attributes, and the label column. The input attributes cover URL structure, domain information, and webpage properties.

The additional phishing.csv file contains 11,054 records with an Index column, 30 feature columns, and a class column. Its feature values use coded indicators, including −1, 0, and 1. Its target
contains 4,897 records coded −1 and 6,157 records coded 1. This file is kept distinct from the larger URL dataset because its feature definitions and target coding differ.

B. Labels and Processed Representations

For the larger dataset, label 0 denotes phishing and label 1 denotes legitimate URLs [7]. The supplied file contains 100,945 phishing records and 134,850 legitimate records. Both processed versions retain these labels and the source row order. The application must use the same class interpretation when displaying predictions.

The 13-feature representation contains url_length, domain_length, path_length, query_length, dots, hyphens, digits, special_chars, has_ip, https, subdomains, keywords, and entropy. These fields describe URL size, character structure, address indicators, and additional lexical characteristics.

The 10-feature representation contains has_ip, url_length, has_at, dots, hyphens, digits, https, slashes, question, and equal. It provides a smaller set of URL-derived indicators. Shared columns in the two processed files match exactly, and their URL-length values match the lengths of the corresponding source URL strings.

C. Data Inspection and Preparation

Inspection found no missing cells in the four supplied CSV files. The larger source contains 235,370 distinct URL strings, with 425 additional rows repeating an existing URL. No repeated URL was found with conflicting class labels. The source count and the distinct-URL count are recorded separately to make the dataset description reproducible.

The processed datasets contain many repeated numerical feature patterns. Some patterns occur under both class labels because different URLs can produce the same reduced feature vector. Such feature collisions must be distinguished from duplicated website records; removing every repeated feature row would change the class distribution and discard separate observations.

Preprocessing must follow the schema required by the model. The report's reference training code uses median imputation and includes standardization for Logistic Regression and SVM. Learned transformations should be fitted using training data and reused for prediction through the saved pipeline [5]. An identifier such as Index is not a website characteristic and should be excluded from model inputs.

D. Model and Application Integration

Logistic Regression supplies a linear classification baseline. Decision Trees express decisions through successive feature rules. SVM uses a separating decision boundary, while Random Forest combines multiple decision trees. These four methods remain the candidate algorithms identified in the project design; this paper does not assign a performance ranking to them.

For deployment, a fitted pipeline is saved with its associated schema and loaded by FastAPI. The service constructs a model-ready row from validated input and returns the predicted class as JSON. The frontend submits requests, indicates processing status, and renders successful predictions or error messages. The same pipeline Artifact and feature interpretation must be maintained across the application components.

E. Evaluation Metrics

Accuracy, precision, recall, and F1-score summarize classification performance. With phishing defined as the positive class, TP denotes correctly detected phishing samples, TN correctly classified legitimate samples, FP legitimate samples incorrectly flagged, and FN missed phishing samples.

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

$$Precision = \frac{TP}{(TP+FP)}$$

$$Recall = \frac{TP}{(TP+FN)}$$

$$F\text{-}1\ Score = \frac{(2TP)}{(2TP+FP+FN)}$$

Confusion matrices accompany these metrics to show the underlying error counts. Recall describes the proportion of phishing samples detected, while precision describes how frequently a phishing warning is correct. These measures support assessment of the trade-off between missed attacks and false alarms.

FUNCTIONAL DEMONSTRATION AND DISCUSSION

A. Demonstration Procedure

Functional assessment follows the normal prediction path. A user enters a supported input in the interface and submits the request. FastAPI validates the payload, prepares the feature vector, and invokes the saved pipeline. The response is then displayed in the React interface. Table III identifies the stages checked within this workflow.

TABLE III

FUNCTIONAL DEMONSTRATION SCOPE

Stage	Functional check
Input submission	Submit supported input from the user interface to the service.
Prediction request	Process the request using the saved pipeline and matching feature schema.
Result display	Return the class prediction and display it in the interface.

B. Functional Outcome

The normal prediction workflow was tested manually. Supported input was submitted through the interface, processed by the prediction service, and returned as a displayed classification. This demonstrated the functional connection between the user interface, backend, and model artifact.

The assessment was limited to the normal workflow. The functional-check table identifies the stages covered by the demonstration and does not constitute an exhaustive test log. A systematic set of labelled cases and formal robustness checks remain outside the scope of this assessment.

C. Interpretation and Limitations

A completed request confirms that the application components communicate and that a model response can be displayed. It does not establish how reliably the classifier distinguishes phishing from legitimate websites. Detection quality has not been quantified in this paper, so no claim is made that a particular classifier is superior or that the displayed class guarantees website safety.

The supplied feature representations also constrain what the system can infer. Reduced URL features can map different websites to identical vectors. A URL-only request requires a matching extractor, and any use of webpage-content attributes requires controlled retrieval. Error handling, uncommon inputs, fresh phishing campaigns, and secure deployment require further verification beyond the normal-workflow demonstration.

CONCLUSION AND FUTURE WORK

This paper described a machine-learning-based phishing detection prototype organized around labelled training data, a saved model pipeline, a FastAPI service, and a React interface. The dataset description records the two source datasets and the 13- and 10-feature processed URL representations, including their row counts and the larger dataset's class coding.

The demonstrated workflow connects input submission, backend processing through a saved pipeline, and display of the predicted class. It establishes application integration within the scope of a manual functional test. The results support discussion of the prototype's operation, with broader detection validation left for subsequent work.

Future work includes verifying the deployed feature extractor against its training schema, recording reproducible model configurations, and evaluating the system on independent and recent website samples. Further improvements may include feature-level explanations, controlled threat-intelligence integration, and protections for any remote page retrieval. Broader functional testing should cover invalid inputs, unavailable services, model-loading failures, and responsive interface behaviour.

REFERENCES

[1] R. Zieni, L. Massari, and M. C. Calzarossa, "Phishing or not phishing? A survey on the detection of phishing websites," IEEE Access, vol. 11, pp. 18499–18519, 2023, doi: 10.1109/ACCESS.2023.3247135.

[2] A. Safi and S. Singh, "A systematic literature review on phishing website detection techniques," Journal of King Saud University—Computer and Information Sciences, vol. 35, no. 2, pp. 590–611, Feb. 2023, doi: 10.1016/j.jksuci.2023.01.004.

[3] M. M. Uddin, K. A. Islam, M. Mamun, V. K. Tiwari, and J. Park, "A comparative analysis of machine learning-based website phishing detection using URL information," in Proc. 2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI), Chengdu, China, 2022, pp. 220–224, doi: 10.1109/PRAI55851.2022.9904055.

[4] N. F. Almujahid, M. A. Haq, and M. Alshehri, "Comparative evaluation of machine learning algorithms for phishing site detection," PeerJ Computer Science, vol. 10, Art. no. e2131, Jun. 2024, doi: 10.7717/peerj-cs.2131.

[5] Scikit-learn Developers, "Common pitfalls and recommended practices," Scikit-learn Documentation. Accessed: Oct. 27, 2026. [Online]. Available: https://scikit-learn.org/stable/common_pitfalls.html.

[6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://jmlr.org/papers/v12/pedregosa11a.html.

[7] A. Prasad and S. Chandra, "PhiUSIIL Phishing URL (Website)," *UCI Machine Learning Repository*, 2024. Accessed: Oct. 27, 2026. [Online]. Available: https://archive.ics.uci.edu/dataset/967/phiusil-phishin g-url-dataset.

[8] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019, doi: 10.1016/j.eswa.2018.09.029.

[9] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Paris, France, 2009, pp. 1245–1254, doi: 10.1145/1557019.1557153.

[10] A. Prasad and S. Chandra, "PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning," *Computers & Security*, vol. 136, Art. no. 103545, 2024, doi: 10.1016/j.cose.2023.103545.

[11] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986, doi: 10.1007/BF00116251.

[12] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995, doi: 10.1007/BF00994018.

[13] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[14] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, Art. no. e0118432, 2015, doi: 10.1371/journal.pone.0118432.

[15] M. Zouina and B. Outtaj, "A novel lightweight URL phishing detection system using SVM and similarity index," *Human-centric Computing and Information Sciences*, vol. 7, Art. no. 17, 2017, doi: 10.1186/s13673-017-0098-1. Springer Nature Link

[16] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using URL and HTML features for phishing webpage detection," *Future Generation Computer Systems*, vol. 94, pp. 27–39, 2019, doi: 10.1016/j.future.2018.11.004.