www.ijarp.com ISSN 2456-9992 Page: 01-06

International Journal Advanced Research Publications

Research Article

Volume: 02
Issue: 03

FAKE URL DETECTION SYSTEM USING MACHINE LEARNING

Nadinharan G. T.[1]*, Ms. J. Vinitha[2]

[1]Student, [2] Assistant Professor

Department of Artificial Intelligence and Machine Learning

Dr. N.G.P Arts and Science College, Coimbatore, Tamil Nadu, India.

Article Received: 17 February 2026, Article Revised: 07 March 2026, Published on: 27 March 2026

***Corresponding Author: Nadinharan G. T.**

Student, Department of Artificial Intelligence and Machine Learning, Dr. N.G.P Arts and Science College, Coimbatore, Tamil Nadu, India.

DOI: https://doi-doi.org/101555/ijarp.3878

ABSTRACT

Phishing is one of those problems that sounds simple until you try to stop it at scale. Attackers build fake versions of websites people trust, wait for someone to type in their password or card number, and disappear before most security tools have even noticed. The traditional defence — checking URLs against a list of known bad addresses — works fine for yesterday's attacks but falls flat against anything new. We set out to build something better. Our Fake URL Detection System analyses a URL the way a trained analyst would: checking its length, its punctuation patterns, how many subdomains it has, whether it is using a raw IP address instead of a proper domain, and whether suspicious vocabulary has been worked into the path. A Random Forest classifier processes these observations and delivers a verdict — legitimate or phishing — in under two seconds. The whole system runs as a Flask web application that anyone can use without any technical background. Our evaluation showed strong accuracy and a low false-negative rate, confirming that machine learning trained on URL structure can reliably catch phishing links that no blacklist would ever see.

KEYWORDS: Phishing Detection; Machine Learning; Random Forest; URL Feature Extraction; Cybersecurity; Flask Web Application; Zero-Day Attacks

4. INTRODUCTION

Most people have received a message at some point that looked completely genuine — the right logo, the right tone, a link that appeared to go exactly where it claimed — and then

something felt off. That instinct is worth paying attention to. Phishing attacks are engineered to suppress it, and they are getting better at doing so.

The core mechanics have not changed much over the years. An attacker registers a domain that resembles something people trust, puts a convincing fake site on it, and waits for someone to enter credentials. What has changed is the speed of the operation. Modern phishing infrastructure can be stood up and taken down in a matter of hours — fast enough to stay ahead of the databases that most security tools rely on. A URL that does not yet appear on any blacklist passes through every check. That gap is the real problem.

The question we started with was straightforward: can a machine learning model examine a URL it has never seen before and make a reliable judgment about whether it looks suspicious? Not by checking a list, but by recognising patterns. The answer, as our results show, is yes. A Random Forest classifier trained on a small but carefully chosen set of URL features achieves strong accuracy and operates fast enough for real-time use. This paper describes how we built it, what we learned from testing it, and where we think it can go from here.

5. MATERIALS AND METHODS

5.1 Dataset and Preprocessing

We worked with a publicly available labelled dataset of URLs, each tagged as either legitimate or phishing based on verified real-world classifications. The first task was cleaning it. We removed columns that carried no useful signal, converted class labels to binary integers — 0 for legitimate URLs, 1 for phishing — and filtered out any rows with missing or malformed entries. A model trained on dirty data learns the wrong things, so we treated this stage seriously rather than rushing through it.

5.2 Feature Extraction

Six URL-level features were computed for every record in the dataset. These were selected because each one corresponds to a real pattern that distinguishes phishing URLs from legitimate ones in practice:

- URL length: Phishing addresses are frequently padded with extra characters to obscure the actual destination or mimic a legitimate URL structure

- Dot count: A legitimate URL typically has one or two dots separating subdomains and the top-level domain. Phishing links often have more, using nested subdomains as camouflage
- Hyphen count: Hyphens get inserted to construct convincing-looking domain names without actually owning the brand being mimicked — for example, paypal-secure-login.com
- Special character frequency: Characters such as @, %, and = appear rarely in genuine URLs but turn up frequently in phishing links, often as part of redirection tricks
- Suspicious keyword presence: Words commonly associated with account activity — login, verify, confirm, secure, update — appearing in the path or subdomain of a URL raise the likelihood of phishing intent
- IP address substitution: Replacing a readable domain name with a raw IP address is a classic indicator that the owner is avoiding the paper trail that comes with domain registration
- Each of these six measurements was extracted programmatically for every URL in the dataset, producing a structured numerical feature matrix ready for model training.

5.3 Model Training

The feature matrix was split into training and test partitions using an 80/20 stratified divide, preserving the class distribution in both halves. Stratification matters here — if the split happens to land mostly legitimate URLs in the test set, accuracy figures become meaningless.

We trained a Random Forest classifier with one hundred decision trees. Random Forest was chosen because it handles mixed feature types well, it is resistant to overfitting through ensemble voting, and its feature importance scores give clear insight into which attributes are doing the most work. Hyperparameters were tuned through five-fold cross-validation on the training partition.

After training, the model was serialised using Python's pickle library so that the Flask application could load it once at startup and then serve predictions entirely from memory. This design choice keeps response times consistent regardless of how frequently the app is queried.

5.4 System Implementation

The full stack was written in Python 3. Pandas managed all data loading and transformation. Scikit-learn provided the Random Forest implementation along with the confusion matrix and

classification report utilities used for evaluation. Feature extraction logic was written as a standalone module shared between the training pipeline and the Flask application — ensuring that the exact same transformations are applied during both training and inference, which is a common failure point in deployed machine learning systems.

The Flask application exposes a single endpoint. A user pastes a URL into a web form and submits it. The application extracts the six features, loads them into the model, and returns a colour-coded result: green for safe, red for phishing. The interface was designed to require no technical knowledge to use.

6. RESULTS AND DISCUSSION

The model was evaluated on the held-out twenty percent of the dataset — data that had been kept entirely separate from training from the beginning. This is the honest test: does the model generalise, or has it simply memorised?

The Random Forest classified URLs with high accuracy across both classes. Feature importance analysis provided useful insight into what drove those predictions. URL length and special character density emerged as the strongest individual predictors. Suspicious keyword presence and IP address substitution followed. Subdomain count was the weakest standalone feature, but it contributed meaningfully when the ensemble considered it alongside the others — a property of ensemble models that makes them more useful than any single feature's individual performance might suggest.

Precision was strong across both classes. More important to us was recall on the phishing class — the model's ability to catch actual phishing URLs rather than letting them through. A false negative here is the mistake that matters in the real world: a phishing link called safe means someone clicks it. That rate was low, and we were satisfied with it.

False positives were also well controlled. A system that flags too many legitimate URLs as suspicious will lose user trust quickly, and users who stop trusting the tool will stop using it. Balancing these two error types is where a lot of detection systems struggle, and we paid careful attention to it throughout tuning.

End-to-end response time averaged 1.3 seconds across one hundred test submissions. For a tool meant to be used before clicking a suspicious link, that speed is practical. It does not feel like waiting.

7. CONCLUSION

We built this system to answer a specific question — can a lightweight machine learning model detect phishing URLs it has never seen before, using only the structure of the URL itself? Based on our results, the answer is yes. Strong accuracy, a low false-negative rate, and sub-two-second response times, all on standard hardware without any dependency on external databases or blacklists.

The web interface means that the intelligence inside the classifier is available to anyone. You do not need to understand machine learning, or cybersecurity, or even what a URL really is. You paste the link and you get an answer. That accessibility is intentional — the people who most need protection from phishing are often the least equipped to recognise it on their own.

This work demonstrates that proactive, pattern-based phishing detection is achievable at low cost and within practical time constraints. It does not replace enterprise-scale security infrastructure, but it extends a meaningful first layer of protection to users and organisations who do not have that infrastructure available to them.

8. ACKNOWLEDGEMENTS

The authors would like to thank the faculty and staff of the Department of Artificial Intelligence and Machine Learning at Dr. N.G.P Arts and Science College, Coimbatore, for their continued guidance and support throughout this research. The authors are also grateful to the open-access cybersecurity research community whose publicly available datasets made this work possible.

9. REFERENCES

1. S. Dua and X. Du, Data Mining and Machine Learning in Cybersecurity, CRC Press, Boca Raton, FL, USA, 2011.
2. A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed., O'Reilly Media, Sebastopol, CA, USA, 2022.
3. N. Provos and T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection, Addison-Wesley Professional, Upper Saddle River, NJ, USA, 2007.
4. S. Afroz and R. Greenstadt, "PhishZoo: Detecting Phishing Websites by Looking at Them," IEEE Transactions on Information Forensics and Security, 2023.
5. Y. Ding, Q. Luktarhan, K. Li, and W. Slamu, "A Keyword-Based Combination Approach for Detecting Phishing Websites," Computers & Security, vol. 112, Jan. 2022, Art. no. 102469.

6. A. K. Jain and B. B. Gupta, "A Machine Learning Based Approach for Phishing Detection Using Hyperlinks Information," Journal of Ambient Intelligence and Humanized Computing, vol. 13, pp. 1–13, 2022.
7. X. Hua and Y. Zhang, "Deep Learning for Phishing Webpage Detection: A Survey," Neural Computing and Applications, vol. 35, pp. 1–21, 2023.