Anti-Phishing Browser Extension Using Machine Learning – Comparative Analysis of URL and Content Features

Aminul Aiman Bin Abdullah1, b) and Siti Husna Abdul Rahman1, 2, a)

*1Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia*

*2Centre for Cybersecurity and Quantum Computing, CoE for Advanced Cloud, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia*

*a) corresponding author: siti.husna@mmu.edu.my*

*b) aminulaiman@gmail.com*

**Abstract.** Phishing remains one of the most pervasive cyber-threats, exploiting user trust to harvest credentials by imitating legitimate websites. Traditional defenses such as blacklists and static rules struggle to keep pace with emergent phishing tactics, yielding high false-positive and false-negative rates. This paper presents a hybrid, machine-learning–driven browser extension that balances low-latency Uniform Resource Locator (URL) analysis with deeper, Docker-isolated content inspection. URL-derived features are extracted locally within a lightweight Docker container, in a parallel with website’s content parsing for content-derived features extraction. Both pipelines employ Random Forest classifiers trained on a balanced dataset of 25,000 legitimate and 25,000 phishing URLs. Machine learning models are evaluated using insights from confusion matrix which comprises accuracy, precision, recall, Type I error, and Type II error, while real-time evaluation metrics include latency, Central Processing Unit (CPU) time usage, and memory usage. URL-only detection achieves 52.4 % accuracy, generating predictions in 20ms with minimal resource overhead. Content-based detection raises accuracy to 78.3 % but incurs 1.5 s latency and higher CPU utilization. These results illuminate a clear trade-off with URL analysis offers ultra-fast, predictable performance at the cost of lower detection quality, whereas full-page parsing improves accuracy with significantly greater delay and variability. The proposed extension demonstrates a scalable framework for adaptive, real-time phishing defense, informing deployment choices across diverse threat-model scenarios.

# Introduction

Phishing attacks are among the most pervasive and rapidly evolving threats to online security: over 1675 websites are compromised daily, and an average of 2 million phishing attacks were reported in the second quarter of 2023 [1]. Traditional detection methods such as blacklists and rule-based systems fail to detect newly instantiated phishing sites that bypass static rules, resulting in high false-positive and false-negative rates [2,3] and prove increasingly unreliable [4,5]. Machine learning (ML) offers decision-support by learning subtle patterns in URL structures and webpage content from large, labeled corpora [6] but must balance speed and depth: URL-derived features enable low-latency assessments yet remain susceptible to obfuscation, while content-based analyses yield richer insights at the expense of processing time [7]. To address this trade-off, we develop a hybrid, Dockerized ML architecture packaged as a browser extension: local URL feature extraction provides rapid initial screening, while a lightweight container performs intensive content analysis within an isolated environment. We evaluate URL-only versus content-only pipelines using confusion matrices, latency, and resource-utilization metrics under realistic browsing scenarios; this proof-of-concept delivers actionable guidance for deploying scalable, real-time ML-driven phishing defenses adaptable to evolving threat landscapes.

# Literature survey

## Emergence of Machine Learning in Phishing Detection

The limitations of traditional techniques have spurred the adoption of ML approaches. ML techniques can learn complex patterns from large datasets and adapt to new phishing tactics by updating the models with fresh data [8]. Early studies focused on extracting lexical features from URLs such as URL length, the number of subdomains, and the presence of Hypertext Transfer Protocol Secure (HTTPS) to distinguish between legitimate and phishing sites [8]. For instance, classifiers like Random Forest and Support Vector Machine demonstrated high accuracy when trained on such features [1]. These approaches marked a significant improvement over static methods by offering a dynamic response to evolving phishing strategies.

## Feature Extraction

Feature extraction is a foundational step in automated phishing detection, deriving discriminative characteristics from either the URL’s lexical structure or the target page’s content via static or dynamic HTML analysis [9].

URL-based extraction treats the address as an opaque character sequence, computing metrics such as length, tokenization patterns, and host-related attributes using only string operations. This lightweight approach imposes minimal latency, preserves user privacy, and supports real-time execution on resource-constrained clients; however, it lacks semantic context, is susceptible to obfuscations like homoglyphs and URL shortening, and yields higher false-positive rates when legitimate URLs exhibit anomalous patterns [9].

In contrast, content-based extraction either parses the HTML statically for forms, scripts, and textual elements or renders the page to observe runtime behaviors (script execution, network requests, visual transitions), thereby detecting embedded malicious code and deceptive interface elements. Empirical studies indicate that dynamic analysis reduces classification uncertainty by approximately 16% compared to static methods, yielding higher overall detection accuracy [10]. Nevertheless, content-oriented approaches incur greater computational overhead, introduce privacy considerations through exposure of page data, and require more complex infrastructure to maintain parsing or headless-browser environments.

## Isolated Machine Learning‑Based Detection

Isolated methods use Docker containers to sandbox feature extraction and model inference, preventing malicious page scripts from affecting the host system. A workflow in which the browser extension forwards each intercepted URL to a Docker container was proposed [11]. Inside that container, a headless browser renders the page, allowing extraction of HTML elements, JavaScript behaviours, and link structures. These features are then fed into a deep learning classifier that distinguishes phishing, TinyURL abuses, and browser‑in‑the‑browser attacks all within the isolated environment. The sandboxing approach is extended by focusing on dynamic behavioural cues such as script execution patterns, redirect chains, and form submissions within Docker [9]. Their deep learning model effectively detects evasive phishing tactics like fake login forms on compromised sites, though the added isolation layer increases processing time compared to purely local methods.

## Real-Time Implementation Evaluations

Phishing detection systems are evaluated by metrics that assess both effectiveness and efficiency, notably end-to-end latency and resource usage. End-to-end latency from URL interception through feature extraction, prediction generation, and result display is benchmarked at less than 300 ms to ensure real-time usability, with timestamps logged at each stage [12]. Resource usage, quantified via CPU and memory monitoring, must be carefully managed to avoid degrading the user experience; studies recommend maintaining CPU utilization below 40 % even under heavy traffic to ensure scalability and real-world suitability [13].

# Methodology

This section details our systematic approach for developing the real-time phishing detection system, including data collection, preprocessing, feature extraction, ML model development, and system integration.

## Data Collection

A total of 50 000 URL entries were extracted from the dataset employed by Prasad and Chandra [10], with an equal split between 25 000 legitimate links and 25 000 confirmed phishing links to ensure class balance. Feature values for each URL were subsequently obtained by web-scraping the corresponding webpages using a purpose-built Python scraper. The specific feature set replicated the attributes identified in the original PhiUSIIL framework [8].

## Feature Extraction

Feature extraction was performed on lexical (URL-based) features and content-based features. Table 1 shows the features that were derived directly from the URL structure.

**TABLE 1.** URL-based (Lexical) features

|  |  |
| --- | --- |
| **Features** | **Description** |
| TLD | Identifies the domain type. |
| URLLength | Measures the total length of the URL. |
| NoOfSubDomain | Counts the number of subdomains in a URL. |
| NoOfObfuscatedChar | Detects encoded or manipulated characters in URLs. |
| IsHTTPS | Check whether the URL uses HTTP or HTTPS. |
| noOfDigits | Counts numeric characters in the URL. |
| noOfEqual | Counts occurrences of = in the URL. |
| noOfQmark | Counts occurrences of ? in the URL. |
| noOfAmp | Counts occurrences of & in the URL. |
| CharContinuationRate | Measures frequency of alphabetic, numeric, or special character. |

Table 2 shows the features that were derived directly from the webpage’s content using a headless browser (Puppeteer).

**TABLE 2.** Content-based features

|  |  |
| --- | --- |
| **Features** | **Description** |
| LargestLineLength | Identify excessively long HTML lines. |
| HasFavicon | Verifies if the site has a favicon. |
| HasDescription | Checks if the meta-description is present. |
| NoOfiFrame | Counts iFrames embedded in the page. |
| HasSocialNet | Checks if the page includes links to social media. |
| NoOfImage | Counts the number of images on the page. |
| NoOfJS | Counts the number of JavaScript files loaded on the page. |
| NoOfSelfRef | Counts hyperlinks that point to the same domain. |
| NoOfEmptyRef | Counts empty anchor tags (<a href="#">). |
| NoOfExternalRef | Counts hyperlinks pointing to external domains. |

These features provide additional context that helps identify sophisticated phishing attacks where URL analysis alone may not suffice [6].

## Data Preprocessing

Data preprocessing comprised two primary activities: data cleaning and feature engineering. Upon completion of feature extraction, the dataset was subjected to a systematic cleaning protocol in which tokens such as “#REF!”, “VALUE!”, “N/A” and any blank or missing entries were uniformly converted to NaN and subsequently removed. For feature engineering, the high‐cardinality Top Level Domain (TLD) attribute was encoded via a supervised “phishing rate” metric: each categorical TLD label was replaced by a continuous value reflecting its empirical association with phishing risk, and Bayesian smoothing was applied to mitigate variance arising from infrequently observed domains.

## Machine Learning Model Evaluation

To evaluate the efficacy of URL‑based, and content‑based features sets for phishing detection, we implemented three supervised classifiers like Random Forest, Decision Tree, and Extreme Gradient Boosting (XGBoost). Each model was trained on the designated training split and assessed via confusion matrix.

## Real-Time System Architecture

Figure 1 illustrates the end-to-end workflow of our phishing-detection extension, in which the browser extension runs inside the user’s browser, monitors each page load, captures the current URL, and immediately forwards it to the Feature Extraction Container, implemented as a Node.js service that accepts incoming URLs. This container performs two parallel operations: URL-based feature extraction, which computes a predefined set of attributes derived solely from the URL string, and content-based feature extraction, which launches a headless browser via Puppeteer to load the page and gather a predefined set of attributes from the rendered content. Once both feature sets are computed, they are merged into a single payload and transmitted to the Model Prediction Container. This container runs a Python Flask service exposing an endpoint that receives the merged feature payload; upon request arrival, the service loads the appropriate trained model, invokes it on the supplied features, and returns the raw model output and timing/resource-usage data to the browser extension. Finally, the extension displays to the user a clear indication of site legitimacy along with performance metrics.

A diagram of a software development

AI-generated content may be incorrect.

**FIGURE 1.** Real-time system architecture

## Real-Time Implementation Evaluation

To evaluate the performance efficiency of each detection phase, we measured three key system metrics: latency, CPU time, and memory usage. These metrics were captured for both the feature extraction and prediction phases but combined under unified expressions since the underlying calculation formulas are structurally identical for URL-based and content-based inputs.

Latency refers to the elapsed time between the start and end of the extraction and prediction task. It is calculated using high-resolution timestamps. Let and represent the millisecond timestamps taken at the start and end of the phase. The latency, denoted as *Latency*, is computed by using Equation (1):

CPU time reflects the amount of processing time consumed by the CPU in milliseconds. It includes time spent in user mode and system mode . Let and be the user CPU time before extraction and after prediction, and and be the corresponding system CPU times. The CPU time, is calculated as in Equation (2):

Memory usage measures the change in a process’s Random Access Memory (RAM) consumption, which more precisely, the change in its Resident Set Size (RSS) during a task, expressed in Megabytes (MB). Let and denote the memory usage at the start of extraction and end of prediction, respectively (see Equation (3)).

# Results

This section presents the results of the experiments conducted using the three ML models: Random Forest, XGBoost, and Decision Tree. The performance of these models was evaluated on three self-extracted datasets: We got two datasets namely URL-Based Features Dataset, and Content-Based Features Dataset.

## Machine Learning Model Evaluation

This section details the empirical evaluation of three widely used supervised algorithms: Random Forest, XGBoost, and Decision Tree applied to three separate feature configurations: URL‑based, and content‑based. Each model was assessed on an unseen test partition to derive the classical confusion‑matrix statistics.

Table 3 shows that URL‑based configuration delivers uniformly high predictive quality, with all three classifiers converging on 0.99 across every confusion‑matrix metric. Both error types are constrained to a ceiling of 0.01, indicating negligible misclassification risk at test time. The close correspondence between precision and recall values demonstrates that the models maintain an equitable balance between false‑positive and false‑negative control. From a deployment perspective, such symmetry is advantageous in real‑time phishing mitigation, as it avoids disproportionate penalties on either user trust (excessive blocking) or security (missed threats).

**TABLE 3.** URL-based features dataset evaluation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **Specificity** | **Type 1 Error** | **Type 2 Error** |
| Random Forest | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.01 | 0.01 |
| XGBoost | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.01 | 0.01 |
| Decision Tree | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.01 | 0.01 |

Switching to content‑centric attributes lowers headline performance across all metrics as shown in Table 4. For the ensemble models, accuracy falls to approximately 0.91, accompanied by a precision–recall pair that hovers near 0.90‑0.93. Although these values remain acceptable by many operational standards, the associated Type I errors of 0.11 reveal a larger proportion of legitimate sites erroneously flagged as malicious. Decision Tree records the steepest decline, with accuracy dropping to 0.87 and Type II errors doubling relative to the ensemble alternatives.

**TABLE 4.** Content-based features dataset evaluation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **Specificity** | **Type 1 Error** | **Type 2 Error** |
| Random Forest | 0.91 | 0.90 | 0.93 | 0.91 | 0.89 | 0.11 | 0.07 |
| XGBoost | 0.91 | 0.90 | 0.92 | 0.91 | 0.89 | 0.11 | 0.08 |
| Decision Tree | 0.87 | 0.89 | 0.84 | 0.87 | 0.90 | 0.10 | 0.16 |

In summary, the empirical results indicate that URL‑based feature engineering produces consistently superior and more stable predictive performance than the content‑based alternative, while simultaneously imposing a lighter computational load. Among the evaluated models, Random Forest attains the highest reliability across both evaluation strategies and maintains negligible latency under browser‑extension constraints. Therefore, Random Forest trained exclusively on URL‑derived features is recommended as the primary classifier for real‑time deployment, with XGBoost serving as a secondary option should additional ensemble diversity be required.

## Real-Time Performance Evaluation

Table 5 shows the results of testing models trained with both URL-based features and content-based features on a balanced dataset of 1,000 phishing and 1,000 legitimate URLs. The URL-based model achieved 52.4 % accuracy and 51.4 % precision, detecting 91.2 % of phishing URLs while falsely flagging 86.4 % of legitimate sites as phishing. In comparison, the content-based model reached 78.3 % accuracy and 83.7 % precision by correctly identifying 70.3 % of phishing URLs and reducing false positives to 13.7 %. These results indicate that URL-based detection maximizes recall but generates excessive false alarms, whereas content-based detection delivers a 25.9 percentage-point increase in accuracy with a manageable false-positive rate, offering a more balanced approach to phishing prevention.

**TABLE 5.** Real-time performance prediction evaluation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **Type I Error (%)** | **Type II Error (%)** |
| URL-Based Features | 52.4 | 51.4 | 91.2 | 86.4 | 8.8 |
| Content-Based Features | 78.3 | 83.7 | 70.3 | 13.7 | 29.7 |

Table 6 shows that from feature extraction through prediction, the URL-based model requires just 1.9 ± 0.8 ms of CPU time and 80.8 ± 3.9 MB of memory to complete each URL evaluation in 20.3 ± 7.0 ms. Conversely, the content-based pipeline consumes 170.5 ± 147.6 ms of CPU time and 81.3 ± 3.5 MB of memory, pushing total latency to 1535.8 ± 932.5 ms per URL. In summary, URL-based detection delivers millisecond-scale consistency with minimal resource overhead, whereas content-based detection incurs second-scale delays and greater CPU variability for deeper analysis

**TABLE 6.** Real-time performance metrics evaluation

|  |  |  |
| --- | --- | --- |
| **Metrics** | **URL-based Mean ± Standard Deviation** | **Content-based Mean ± Standard Deviation** |
| CPU time (ms) | 1.89 ± 0.78 | 170.50 ± 147.57 |
| Memory use (MB) | 80.8 ± 3.9 | 81.3 ± 3.5 |
| Latency (ms) | 20.27 ± 7.01 | 1535.82 ± 932.45 |

# Conclusion

This work has introduced and evaluated a proof-of-concept, machine-learning–based browser extension that integrates both URL and content-based phishing detection within a modular, Dockerized architecture. Through comprehensive experiments on a balanced URL dataset, we quantified the trade-offs between speed and accuracy: while URL-only analysis delivers sub-20ms predictions with 52.4 % accuracy, full-page content parsing boosts detection to 78.3 % at the expense of more than 1 s latency and increased both CPU and memory demands. Random Forest emerged as the most reliable classifier across both pipelines. These findings guide practitioners in selecting the appropriate detection strategy based on application constraints favoring URL-based methods when latency and resource efficiency are paramount, and content-based methods when higher detection fidelity is required.

Future work will extend this hybrid framework by optimizing content-parsing speed, incorporating dynamic behavioral features by investigating script execution patterns and network requests, and evaluating performance on larger, more diverse URL corpora. Additionally, exploring ensemble or multi-stage fusion techniques may further improve accuracy without prohibitive latency. Ultimately, this research contributes actionable insights toward deploying adaptive, scalable phishing defenses in modern browsing environments.

# References

1. Anti-Phishing Working Group, *Phishing Activity Trends Report*, 2025, <http://www.apwg.org>.
2. B. Gogoi, T. Ahmed, and A. Dutta, “A Hybrid approach combining blocklists, machine learning and deep learning for detection of malicious URLs,” in *Proceedings of the 3rd IEEE India Council International Subsections Conference (INDISCON 2022)* (IEEE, 2022).
3. S. Parekh, D. Parikh, S. Kotak, and S. Sankhe, “A new method for Detection of Phishing Websites: URL Detection,” in *Proceedings of the 2018 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018)* (IEEE, 2018).
4. D. M. Linh, H. D. Hung, H. M. Chau, Q. S. Vu, and T. N. Tran, “Real-time phishing detection using deep learning methods by extensions,” International Journal of Electrical and Computer Engineering **14**(3), 3021–3035 (2024).
5. M. A. Syafiq Rohmat Rose, N. Basir, N. F. Nabila Rafie Heng, N. Juana Mohd Zaizi, and M. M. Saudi, “Phishing Detection and Prevention using Chrome Extension,” in *Proceedings of the 10th International Symposium on Digital Forensics and Security (ISDFS 2022)* (IEEE, 2022).
6. B. B. Gupta, K. Yadav, I. Razzak, K. Psannis, A. Castiglione, and X. Chang, “A novel approach for phishing URLs detection using lexical based machine learning in a real-time environment,” Computer Communications **175**, 47–57 (2021).
7. D. Prasad, S. Suhasini, B. Parthasarathy, and J. Praveen, “Enhancing Internet Security: A Machine Learning-Based Browser Extension to Prevent Phishing Attacks,” in *Proceedings of the International Conference on Communication, Computer Sciences and Engineering (IC3SE 2024)* (IEEE, 2024), pp. 896–900.
8. A. Prasad and S. Chandra, “PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning,” Computers and Security **136** (2024).
9. A. S. Rafsanjani, N. B. Kamaruddin, M. Behjati, S. Aslam, A. Sarfaraz, and A. Amphawan, “Enhancing Malicious URL Detection: A Novel Framework Leveraging Priority Coefficient and Feature Evaluation,” IEEE Access **12**, 85001–85026 (2024).
10. A. O’Mara, I. Alsmadi, A. Aleroud, and D. Alharthi, “Phishing Detection Based on Webpage Content: Static and Dynamic Analysis,” in *Proceedings of the 2023 3rd Intelligent Cybersecurity Conference (ICSC 2023)* (IEEE, 2023), pp. 39–45.
11. S. Asiri, Y. Xiao, and S. Alzahrani, “Towards Improving Phishing Detection System Using Human in the Loop Deep Learning Model,” in *Proceedings of the 2024 ACM Southeast Conference (ACMSE 2024)* (ACM, 2024), pp. 77–85.
12. A. B. Majgave and N. L. Gavankar, “Automatic phishing website detection and prevention model using transformer deep belief network,” Computers and Security **147** (2024).
13. A. Aljofey, Q. Jiang, A. Rasool, H. Chen, W. Liu, Q. Qu, and Y. Wang, “An effective detection approach for phishing websites using URL and HTML features,” Scientific Reports **12**(1) (2022).