# ML Phishing Detection Extension

Hussain Zainal
Cyber Security Department
George Mason University
Fairfax, VA
hzainal@gmu.edu

Nate Le
Cyber Security Department
George Mason University
Fairfax, VA
nle51@gmu.edu

Supreeth Gowda A
Cyber Security Department
George Mason University
Fairfax, VA
sashwath@gmu.edu

*Abstract*—**Phishing remains one of the most effective yet common cyber threats, exploiting human behavior by engineering deceptive websites to steal vital information. The evolving phishing tactics are outpacing native safe browser support. This project proposes a Chrome extension that integrates a multi-layered phishing detection approach by using ML, URL similarity detection, network behavior analysis, and page design analysis. This extension provides to not only protect users, but to also educate them on why a website has been flagged as phishing. Evaluation will be performed on phishing and legitimate website datasets with performance measured by accuracy and precision.**

***Keywords—Phishing Detection, Machine Learning, Network Behavior, Feature Design***

## I. Introduction

Our project proposes developing a Google Chrome Extension that combines multiple phishing detection techniques such as URL similarity analysis, network behavior monitoring, and machine learning. The extension will analyze URLs via logistic regression, inspect page content, and observe network request behavior to provide real-time warnings to users. By combining multiple phishing detection strategies, the extension's goal is to catch attacks that evade traditional blacklist systems and provide users explainable insights on why such websites might be dangerous.

## II. Motivation for project

Phishing continues to be one of the most widespread and effective cyber threats, as more breakthroughs are made in the field of cybersecurity, human behavior remains to still be the weakest link in a system. Not only would this extension assist in preventing successful phishing attacks, but it would also assist users to be able to identify broader patterns of online deception. By presenting clear explanations on why a website has been flagged as suspicious, the users can be educated and build long-term awareness as phishing gets more advanced.

As graduate students currently taking network security, we are motivated to pursue this project due to the opportunity to apply network security concepts in a practical manner. Building a chrome extension allows us to design, implement, and evaluate practical countermeasures and helps bridge the gap between security research and software engineering. This endeavor would help strengthen our knowledge of protocols, traffic patterns and adversarial behavior. By doing so, we are able to incorporate network security in a practical sense alongside with our other skills we are implementing into this project.

## III. Background

Phishing has evolved into one of the most common attack vectors and damaging forms of cybercrime, targeting the most vulnerable within a system, the human. By impersonating trusted entities, attackers deceive users into willingly give up their sensitive information instead of relying on technical attacks. As a result, phishing continues to dominate cyber incident reports, forcing security researchers to develop advanced detection mechanisms that have to stay up to date with the rapid evolution of phishing techniques.

Phishing website detection has shifted from heuristic and blacklist-based approaches and towards machine learning classification due to its rising sophistication. Traditional systems struggle against zero-day threats, whereas machine learning models can learn based on statistical irregularities rather than predetermined cases. URL-based ML models transform the URL string into lexical and structural attributes, such as: length, entropy, homograph similarity, etc, all as numerical values. Classifiers such as a Random Forest algorithm identify patterns present within the relationship between attributes and can correlate it with phishing behaviour. The random forest algorithm consists of multiple decision trees that come together and have a majority vote of their final classification results. Each tree provides its own answer which gives developers and researchers a deeper insight of what features contribute to a decision the most. This makes it well-suited as the first line of defense in a phishing detection system.

However, modern phishing pages have developed counterstrategies that can get lower confidence scores when it comes to URL-only detection by using “safe” domains while embedding deceptive behaviour within the page content itself. This has driven the need for a hybrid detection model that combines URL-level classifiers with page content based analysis. Page level features include internal vs external link ratios, resource complexity, credential entry fields, metadata consistency, and more.

Deploying a machine learning extension inside a browser extension would be able to combine both approaches into one. There are additional constraints that come with Chrome’s Manifest V3 architecture which limits access to network requests, background execution and deeper level inspection of APIs. These restrictions require inference logic to run locally using only client-visible features such as URLs and DOM content. The random forest models trained have to be stored in a browser compatible JSON format and have to be executed via custom JavaScript tree traversal. Ensuring that feature extraction remains efficient for real-time browsing is a key challenge. The nature of Random Forest provides rankings of feature importance which will be key for providing user explanations on why a website has triggered the phishing warning.

## IV. URL Analysis

### A. Objective

This section presents the development and evaluation of a machine learning subsystem designed for detecting if a given URL is either a phishing website or legitimate. The approach focuses on analyzing characteristics that were extracted from the URL itself. Unlike content-based or visual similarity models, this relies solely on URL-level features, enabling faster-real time evaluation. The objective of this component within the entire application is to be the first line of defense, flagging potential phishing attempts before content rendering or user interaction occurs.

## *B. Dataset and Preprocessing*

The dataset has over 235,000 URLs from multiple distinct sources to ensure a diverse and balanced dataset source of both legitimate and phishing websites. To ensure integrity, all URLs were validated and normalized through the following standardization process:

- Lowercasing all characters
- Removing trailing slashes
- Normalizing URL encodings
- Removing redundant delimiters
- Deduplicating URLs to prevent Bias

## *C. Url Feature Engineering*

Each URL is transformed into a multi-dimensional numerical representation composed of 50 features across structural, domain-level, semantic, and statistical indicators. This ensures the model can learn from both explicit patterns (e.g., suspicious tokens) and latent irregularities (e.g., entropy deviations).

| **Category** | **Features** |
|---|---|
| Basic URL Structure (12) | Url_length, domain_length, domain_name_length, tld_length, path_length, query_length, path_depth, character counts, has_file_extension, suspicious_file_ext, double_slash, trialing_slash, uses_http, uses_https |
| Domain Analysis (8) | Subdomain_count, has_subdomain, has_port, has_ip, has_suspicious_tld, has_numbers_in_domain, domain_entropy |
| Path Metrics(3) | Num_params, has_suspicious_params, query_length, path_length, path_depth, file extensions |
| Suspicious Patterns & Obfuscation (10) | Has_at_symbol, has_shortner, has_suspicious_keywords, has_obfuscation, num_obfuscated_chars, obfuscation_ratio, digit_ratio, letter_ratio, special_char_ratio, url_entropy |
| Brand Detection & Homograph (9) | Suspicious_brand_usage, brand_in_registered_domain, brand_in_subdomain, brand_in_path_or_query, brand_mismatch, brand_similairity_registered, brand_similarirty_subdomain, brand_similairty_path, brand_homograph |

Fig. 1. URL Feature categories

## *D. Hybrid Domain-Path Ensemble*

A key innovation in this implementation was a dual-model weighted ensemble approach that separated domain analysis from path analysis. The path model did reuse non-domain specific features present in the domain analysis. The total weighted model had a 80% domain 20% pathing split.

| **Parameter** | **Value** |
|---|---|
| Number of Trees | 100 |
| Max Depth | 10 |
| Min samples per Leaf | 5 |
| Min Samples Split | 5 |
| Sampling | Bootstrap |
| Criterion | Gini Impurity |
| Training-Testing split | 85-15 |

Figure 2, URL Model Param Values

## *E. Testing Results and Evaluation*

Below are the testing results and evaluation of the ML algorithm using the gathered datasets.

| **Metric** | **Score** |
|---|---|
| Accuracy | 99.33% |
| Precision | 99.45% |
| Recall | 99.38% |
| F1-Score | 99.42% |

| | |
|---|---|
| False negatives | 126 |
| False positives | 113 |
| True Negatives | 14,973 |
| True Positives | 20,358 |

Fig. 3. Final URL Model Results

The top contributing features reveal that protocol and path structure are the strongest indicators, with protocol features resulting in 41.7% of model decisions, path structure features contributing 32.5%, and the remaining six out of the top 10 resulting in ~17%

| **Feature** | **Importance** |
|---|---|
| uses_https | 22.38% |
| uses_http | 19.32% |
| num_slashes | 12.48% |
| path_depth | 10.66% |
| digit_ratio | 3.96% |
| brand_similarity_subdomai n | 3.93% |
| trailing_slash | 3.37% |
| url_length | 2.04% |
| Brand_similiarity_path | 1.90% |

Fig. 4. Top 10 important URL features

Manual inspection of misclassified URLs revealed that 196 out of the 239 total errors (82%) with 102 false positives and 94 false negatives involved URLs with irregularly long or complex paths. This can be attributed to the limited path-specific feature set, which lacks granularity to distinguish between legitimate complex paths and malicious obfuscation. Specifically, legitimate URLs containing UUIDs, session tokens, hash identified, legitimate file names inflate path length and entropy metrics in ways that are similar to phishing characteristics. Additionally, path segments such as /login, /verify, account trigger false positives despite being commonplace in legitimate authentication flows.

### *F. Conclusion of URL analysis*

The URL-based model achieves 99.93% accuracy with minimal false negatives making it suitable to act as the first-line of defense. The hybrid domain-path ensemble architecture provides flexibility in weighting component contributions and the feature importance analysis confirms that path complexity and protocols are the most discriminating factors in identifying phishing attempts. Error analysis reveals that the majority of misclassifications stem from path complexity edge cases that context-aware keyword analysis and feature engineering around UUID recognition would be needed for real world accuracy improvements.

## V. Website Design Analysis

### *A. Objective*

The webpage design analysis module aims to examine the visual and structural components of a website to detect subtle differences between legitimate and phishing pages. While the URL-based model focuses on string-level analysis, this stage evaluates what is actually displayed once the page is loaded. By analyzing the Document Object Model (DOM), visible content, embedded resources, and basic network behavior indicators, the system provides a deeper layer of defense against deceptive phishing attacks. This layer acts as the second line of defense after URL analysis, specifically targeting phishing sites that use safe-looking URLs but expose inconsistencies in page design. According to Li et al. [1], integrating content-based and structure-based webpage features significantly improves phishing detection accuracy compared to URL-only methods.

### *B. Data Collection and Scraping*

To extract webpage data, we automated the process using Selenium WebDriver and BeautifulSoup. Selenium was employed to load each webpage, including dynamically generated content, while BeautifulSoup was used to parse the rendered HTML. This hybrid approach allowed us to analyze both static and dynamic features of the page, consistent with recommended practices in feature-based phishing detection research [1].

The scraper captured multiple elements from each webpage:

The scraper captured multiple elements from each webpage:

- Page Text: all visible text from headings, paragraphs, and hyperlinks
- Images and Media: image filenames, counts, and embedded resource paths
- Links and Forms: ratio of internal to external links, presence of credential fields, and use of suspicious keywords such as 'login', 'verify', or 'update'
- Layout Characteristics: DOM depth, number of unique HTML tags, and the ratio of scripts to content tags
- Network Activity: external resource requests, SSL certificate data, and domain origins of linked elements

All collected attributes were converted into numeric or boolean representations to ensure they could be easily incorporated into a machine learning pipeline.

### *C. Feature Engineering*

Following Li et al. [1], content structure and network interaction patterns provide valuable indicators of phishing activity. This separate page dataset contained 235,000 webpages with pre-extracted HTML features. Each page was scraped and analyzed to extract structural and content-based indicators to be trained on with the same Random Forest model architecture used for URL analysis. Each webpage has 28 features extracted from the scraped content organized into the following categories:

| **Category** | **Features** |
|---|---|
| Content Structure (3) | LineOfCode, LargestLineLength, HasTitle |
| Title Matching (2) | DomainTitleMatchScore, URLTitleMatchScore |
| Trust Signals (4) | HasFavicon, Robots, IsResponsive, HasCopyRightInfo |
| Redirect Analysis (2) | NoOfURLRedirect, NoOfSelfRedirect |
| Metadata (1) | HasDescription |
| DOM Structure (2) | NoOfPopup, NoOfiFRame |
| Form Analysis (4) | HasExternalFormSubmit, HasSubmitButton, HasHiddenFields, HasPasswordField |
| Social Presence(1) | HasSocialNet |
| Keyword Detection (3) | Bank, Pay, Crypto |
| Resource Counts (3) | NoOfImage, NoOfCSS, NoOfJS |
| Link Analysis (3) | NoOfSelfRef, NoOfEmptyRef, NoOfExternalRef |

Fig. 5. Page Model Features

### *D. Role in the System*

The page analysis module runs after the initial URL screening and provides deeper inspection of the page's visual and structural behavior. It is particularly effective at detecting cloned or template-based phishing pages that may bypass URL-level checks. The contribution to the final model follows a weighted ensemble approach:

- URL Model: 80% weight
- Page Model: 20% weight

The combined score generates a risk rating of Safe, Mixed, or Suspicious. This weighting reflects the reliability hierarchy where URL features provide faster, more definitive signals, while page features add contextual validation and reduce false negatives.

### *E. Feature importance Analysis*

The top contributing features reveal that external reference patterns and resource and resource counts provide the strongest indicators to determine a page's legitimacy.

| Feature | Importance |
|---|---|
| NoOfExternalRef | 24.67% |
| NoOfSelfReg | 14.91% |
| NoOfImage | 13.15% |
| LineOfCode | 12.22% |
| NoOfJS | 8.44% |
| NoOfCSS | 6.22% |
| HasSocialNet | 5.68% |
| HasCopyRightInfo | 4.51% |
| HasDescription | 3.14% |

Fig. 6. Top 10 Important Page Model Features

### *F. Results & Parameters*

| **Parameter** | **Value** |
|---|---|
| Number of Trees | 100 |
| Max Depth | 10 |
| Min samples per Leaf | 2 |
| Min Samples Split | 5 |
| Sampling | Bootstrap |
| Criterion | Gini Impurity |
| Training-Testing split | 85-15 |

Figure 7, Page model Param Values

| Metric | Score |
|---|---|
| Accuracy | 99.70% |
| Precision | 99.70% |
| Recall | 99.70% |
| F1-Score | 99.70% |
| True positives | 20,207 |
| True negatives | 15,056 |
| False positives | 86 |
| False negatives | 20 |

Fig. 8. Final Page Model Results

### G. *Key Findings*

1) Link Structure

Link analysis features (NoOfExternalRef, NoOfSelfRef) accounted for over 39.5% of model decisions, indicating that phishing pages exhibit distinct linking patterns, often lacking internal navigation or containing excessive external references. Phishing sites often contain a disproportionate amount of external references because common attack kits typically load scripts, images, and styling resources from third-party hosting services to reduce development effort while trying to maximize its likelihood of seeming legitimate. Conversely, legitimate websites employ their own link ecosystem that reinforces branding and site identity. This aligns with existing research showing that phishing pages generally contain fewer internal links and a shallower structure depth than legitimate domains [2].

2) Resource Complexity Reflects Development Investment

Resource complexity features contributed to 40% of model decisions, showcasing that phishing pages are typically simpler clones with fewer assets. As a result, phishing pages contain fewer images, shorter frontend code, and minimal CSS & JavaScript. Legitimate websites invest in more media assets, dynamic components, responsive frameworks, and interactive website design. Security trend reports state that due to the short-lived, low-investment nature of phishing pages, they are designed with rapid deployment in mind, which contributes to their low resource diversity [3].

3) Trust Signal Absence

Professional indicator features (HasSocialNet, HasCopyRightInfo, HasDescription) contribute to 13.51% of the model's decision-making. Legitimate organizations reliably include social media icons, copyright notices, brand metadata, and SEO-optimized metadata. These attributes do not assist in tricking users about the legitimacy of a website. Prior studies confirm that the absence of branding signals indicates a higher likelihood of phishing attempts [4].

4) Form Features as Secondary Validators

Despite form-related features (HasPasswordField, HasHiddenFields) not showing significant individual importance, they remain essential for detecting credential harvesting behavior. Their significance only shows when combined with other suspicious features. Research supports that form characteristics when tested individually are weak indicators but they become high-level discriminators when assessed in tandem with structural and content-level anomalies [5].

### H. *Benefits of the Page Analysis Layer*

The integration of page-based analysis provides several key advantages:

- Reduces false negatives by catching sophisticated phishing attempts that pass URL screening
- Detects fake login pages and brand impersonation through content and form analysis
- Improves reliability of the extension's warnings through multi-layered validation
- Catches cloned pages that replicate legitimate designs but lack proper resource investment

### I. *Issues & Limitations*

1) Dynamic content & JavaScript Rendering

A major limitation of a page-based ML analysis approach is the inability to fully analyze dynamically rendered content. This was an issue due to our scraping implementation being unable to obtain these features, which was consistent with the referenced datasets also not having those features listed. Modern phishing kits are increasingly using script-injected content which is both difficult to scrape, and to assess its harm capabilities without fully rendering the website. Lastly, because our system captures the DOM at a single point in time, it may miss malicious elements introduced at a later point.

2) Redirect Chain Analysis

The redirect-related features exhibited insignificant importance totalling 0.0502%. This is due to Google Chrome extensions being entirely unable to observe HTTP-level request sequences. DOM inspect cannot reliably expose redirect chains, especially before the page fully loads.

3) Single-Page analysis scope

The current model analyzes pages individually without evaluating pages across the entire site. While phishing websites tend to be shallow and not have many subpages as they tend to only have isolated credential harvesting pages; Having multiple pages is a good sign of a website's legitimacy. Expanding the analysis to multi-page behaviour could improve accuracy but would run into the risk of causing server issues due to deeper page scraping.

## VI. Extension Implementation

To achieve a reliable phishing detection, we combined the URL-based approach with the page-based approach. Each technique captures different dimensions of phishing behaviour, and their integrations from a multi-layered defense. The integration pipeline follows a staged sequential workflow:

### *A. Stage 1- URL Pre-screening (Fast Filter Layer)*

When a user attempts to access a website, the extension immediately extracts and processes the raw URL.

- The engineered URL features are computed locally
- The random forest ensemble provides a probability score indicating the likelihood that the website is a phishing site.
- If the prediction is above the warning threshold (>0.5), the extension issues a warning.
- If the prediction is above the blocked threshold (>0.85), the extension blocks the website entirely while providing the user why it has been flagged along with the option to continue.

### *B. Stage 2 - Webpage Design*

Once the URL passes initial screening or receives a borderline score, the extension proceeds to analyze the rendered website:

- DOM Feature Extraction: After page load completes, the content script extracts 28 features (Section V) using JavaScript DOM APIs.
- Page Model Prediction: Features are sent to the background service worker.
- Weighted Combination: The final phishing likelihood is computed using weighted average: Combine_Score = (URL_Score × 0.8) + (Page_Score × 0.2)
- If combined_Score >= 0.7, a full-screen warning is displayed to the user

1. Chrome Extension Architecture

   The extension follows Chrome's Manifest V3 architecture [6] with three core components

   Background Service Worker (background.js) [7]:

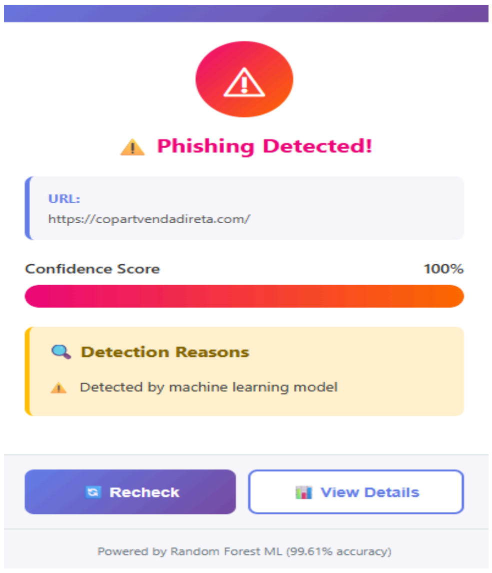
- Loads both URL and Page ML models on extension startup (JSON files)
- Monitors all tab navigation events
- Executes UrL prediction immediately on navigation
- Receives page features from content scripts via message passing
- Combines predictions using weighted averaging
- Triggers warning

Content Script (content.js) [8]:

- Injected into every webpage user visits
- Wait 500ms post-load for DOM stabilization
- Extract 28 page features
- Sends features to background worker
- Display warnings when phishing detected
- Shows discrete security badges with confidence percentages

Popup Interface (popup.html/popup.js)

- Activated when user clicks extension icon
- Displays detailed analysis breakdown



⚠ Phishing Detected!

URL:
https://copartvendadireta.com/

Confidence Score 100%

Detection Reasons
⚠ Detected by machine learning model

Powered by Random Forest ML (99.61% accuracy)

Fig. 9. Popup Interface

2. Machine Learning Integration

The Python-trained scikit-learn Random Forest models cannot run directly in browsers. We developed a conversion pipeline:

Model Export WorkFlow:

1. Export Script: export_model_to_js.py and export_page_model_to_js.py
2. Tree Extraction: Extracted decision tree structures from RandomForestClassifier
3. Scaler Parameters: Exported StandardScaler mean and scale values for feature normalization
4. Saved trees as JSON with split conditions and leaf values
5. Model Variants: Generated full (100 trees) and lite (30 trees) version.

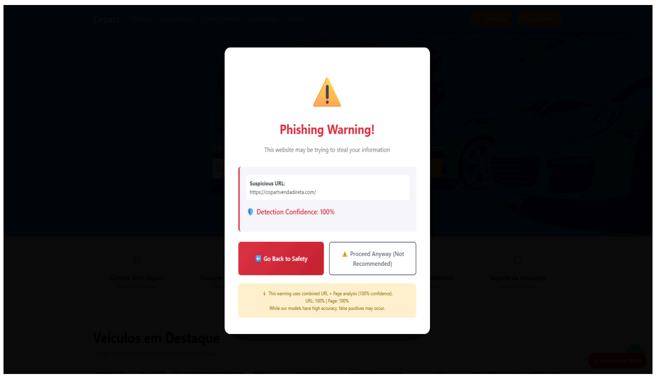
JavaScript Model Implementation

- Implemented Random Forest prediction logic in pure JavaScript (ml-model.js and page-model.js)
- Supports feature scaling, tree traversal, and voting aggregation

Feature Extraction in JavaScript

- URL feature extraction ported to url-features.js
- Webpage feature extraction uses DOM APIs (e.g., querySelectorALL)

3. User Interface and Warning System


Phishing Warning!

This website may be trying to steal your information

Suspicious URL:
https://copartvendadireta.com/

Detection Confidence: 100%

This warning uses combined URL + Page analysis (100% confidence).
URL: 100% | Page: 100%
While our models have high accuracy, false positives may occur.

Fig. 10. Full-Screen Phishing Warning

Displayed when the combined Score >= 0.7:

- Large warning icon and color-coded risk indicator.
- Show URL, prediction scores, and key reasons for detection.
- “Go Back to Safety” and “Proceed Anyway” actions.
- Includes a short disclaimer emphasizing possible positives

Security badge displayed for all analyzed pages

- Green badge for safe sites
- Red badge for suspicious pages with confidence score.
- Clicking the badge opens a detailed view in the popup.
- Automatically fades for low-risk pages to avoid alert fatigue.

4. Performance Optimization

- Caching: 10-minutes LRU cache for URL predictions (<5ms lookup)
- Lite Models: Reduced model size by 78% with <1% accuracy loss.
- Lazy Loading: Models loaded on Startup; fallback deferred loader for reliability

5. Deployment

Current deployment uses Chrome Developer Mode:

1. Navigate to chrome://extensions/
2. Enable Developer mode.
3. Select Load unpacked and choose the extension folder.
4. Extension loads immediately for testing and evaluation.

6. Challenges

- Random Forest in Browser: Python model cannot run in JavaScript.
- Models too large & Slow performance: Original full models are too large (100 trees) which make the loading time long (~2.5 ms).
- Some features only indicate phishing when combined with other features. For example, having multiple login forms is only suspicious if the site also lacks HTTPS. Individual feature flags in isolation can mislead users into thinking legit sites are dangerous.

## VII. Conclusion

### *A. Main Limitations & Issues faced*

Although the system demonstrated capability in detecting site legitimacy with high accuracy in controlled testing, several limitations constrained its real-world performance. A major limitation of the extension was its inability to scrape and analyze dynamically injected JavaScript content, which is a critical feature for determining a website's status. Modern phishing sites often render credential fields after the initial DOM load, but our extension only captures the initial snapshot of the website at 500ms post-load and could miss key details that appear later.

Another significant limitation stemmed from Chrome's Manifest V3 architecture, which severely restricts an extension's access to network-level data. Because service workers cannot inspect full redirect chains and certificate details, redirect-related features contributed insignificantly (0.0502%). These restrictions prevented implementing Layer 3 network analysis that was present in the original project plan, which would have monitored SSL/TLS certificates, tracked redirect chains, analyzed external resource requests, and detected mixed content and suspicious connections.

Despite the large dataset of 235,000+ entries for both URL and page models, limitations persisted. The page dataset was missing multiple features such as dynamically rendered attributes and comprehensive robot.txt checks, reducing model generalizability. The URL dataset had heavy emphasis on domains, making URLs with complex paths less commonplace. The model tended to overfit on domain-based analysis rather than holistic URL evaluation. For example, legitimate URLs with long paths containing UUIDs would trigger detection trees that emphasized URL length. While this issue was addressed with a split URL/Domain model with 80-20 weighting, the reference set for path-heavy URLs remained limited.

Real-world testing revealed a higher false positive rate than the ML testing phase indicated. This discrepancy occurred because real testing happened late in development, causing unexpected edge cases to surface (mostly path-based). The HTTP/HTTPS feature dominance (40% combined importance) meant that nearly every non-HTTPS URL was immediately flagged. Several features had less than 0.001% importance, indicating poor feature engineering decisions. Additionally, localhost testing for fake websites produced flags that would not be present in production environments, skewing early validation results.

Despite these constraints, each limitation provides a clear direction for future improvements. Addressing them will be crucial to enhance the extension's reliability in real-world settings that are increasingly dependent on dynamic content and modern web technologies.

### *B. Future work*

Looking ahead, several enhancements can improve the reliability and adaptability of this extension in real-world scenarios. One major direction is the development of a vetted community-driven blacklist and whitelist system. When a report is submitted and verified, it can be added to the training data for continuous model improvement. A dynamic whitelist would not only reduce false positives in production but could enable more precise detection of brand spoofing attacks, an area where URL-only heuristics remain limited.

Currently, our extension flags individual features in isolation (e.g, “brand name in URL”), this makes users confused when they see warnings on legitimate sites, but the Random Forest Model actually makes decisions based on feature combinations. For example, multiple forms are only suspicious when combined with things like missing HTTPS. So our future work is to find a way to help users understand why specific combinations indicate phishing rather than just listing flagged features.

Expanding the diversity and scale of all datasets is critical. The URL dataset needs more path-oriented URLs with complex structures (UUIDs, session tokens, hash identifiers) to improve the model's ability to distinguish between legitimate complexity and malicious obfuscation. The page dataset requires modernization with features that capture JavaScript-rendered content, single-page application behaviors, and contemporary web framework patterns.

Another critical improvement is advancing the page analysis model to better support dynamic and JavaScript-heavy websites. The present model struggled with SPAs utilizing modern frameworks, suggesting that feature extraction must incorporate behavior-based monitoring and continuous DOM observation techniques rather than single-point-in-time snapshots. Implementing delayed feature extraction at multiple intervals or monitoring DOM mutation events could capture phishing elements that load asynchronously.

More rigorous real-world testing is essential. The extension needs extensive testing with actual phishing websites in production environments rather than localhost simulations. This would expose edge cases earlier in development and allow for iterative refinement of the model and thresholds. Beta testing with real users could provide valuable feedback on false positive rates and usability issues.

Finally, revisiting the original vision for Layer 3 network analysis remains a priority, as it provides a whole new avenue for model features and assists with adding more features for the page model. While Manifest V3 imposes restrictions, exploring workarounds such as declarativeNetRequest API capabilities or companion native messaging hosts could enable limited network monitoring. Features like certificate validation, mixed content detection, and redirect chain analysis would significantly strengthen detection capabilities.

Improving contextual explanations is also necessary. Users need clearer, more actionable information about why a site was flagged, presented in non-technical language. This educational component enhances long-term user awareness and trust in the system. Collectively, these improvements would strengthen the extension's accuracy, transparency, and usability, making it a production-ready phishing defense system.

### *C. Final conclusion*

The development of this multi-layed phishing detection system demonstrates that combining URL-based machine learning, DOM structural analysis, and browser-integrated feature extraction can enhance real-time phishing protection. The random forest URL model achieved high precision and accuracy, proving the effectiveness of treating it as a first line of defense. Complementing the URL model, the webpage design model provided deeper insight into resource usage, trust signals, link structures, and form behaviour, all attributes that legitimate websites usually contain whereas phishing pages often lack. The systems architecture implemented within Chrome’s Manifest V3 constraints utilizes these layers into a functional browser extension capable of delivering clear and explainable warnings to users.

Beyond classification performance, this project highlights the importance of educating users about phishing websites. The extension not only alerts users to malicious activity but explains why the page is suspicious, reinforcing safe browsing habits and long-term awareness. This hybrid design also emphasizes the need of a holistic approach rather than just focusing on one method. This project went deeper when compared to the standard phishing URL ML projects. By utilizing this extension in a real world setting, project members were able to identify signs of a phishing website that a traditional training/testing verification would. Overall, this project bridges the gap between academic research and deployable security tools, demonstrating a practical and scalable framework for protecting users against the increasingly sophisticated phishing landscape and its threats.

## VIII. References

[1] W. Li, S. Manickam, Y.-W. Chong, W. Leng, and P. Nanda, “A State-of-the-Art Review on Phishing Website Detection Techniques,” IEEE Access, vol. 11, pp. 1–20, 2024, doi: 10.1109/ACCESS.2024.3514972.

[2] Mohammad, R., Thabtah, F., & McCluskey, L. (2013). Predicting phishing websites based on self-structuring neural networks.

[3] APWG. (2022). Phishing Trends Report.

[4] Liu, W., & Zhang, G. (2018). Phishing web page detection by analyzing the page layout and contents.

[5] Abdelhamid, N., et al. (2014). Phishing Detection Based on Hybrid Features.

[6] Google Chrome Developers, "Manifest V3 Overview," Chrome for Developers, 2023. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/intro/.

[7] Google Chrome Developers, "Service Workers in Chrome Extensions," Chrome for Developers, 2023. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/service_workers/.

[8] Google Chrome Developers, "Content Scripts," Chrome for Developers, 2023. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/content_scripts/.

[9] Github page https://github.com/MarioDS15/CYSE610Project