Engineering and Technology Journal e-ISSN: 2456-3358

Volume 11 Issue 05 May-2026, Page No.- 10242-10255

DOI: 10.47191/etj/v11i05.63, I.F. – 8.595

Implementation of a Browser Extension with Multi-Layer Security Filtering Architecture for Real-Time Phishing Website Detection

Muhammad Rafly Wirayudha[1], Rahmad Abdillah[2*], Novriyanto[3], Pizaini[4]

[1,2,3,4]Department of Informatics Engineering, Faculty of Science and Technology, Universitas Islam Negeri Sultan Syarif Kasim Riau, Indonesia

ABSTRACT: The increasingly sophisticated evolution of phishing attacks poses a significant challenge for cybersecurity systems, particularly regarding zero-day threats that frequently evade traditional blacklist detection. This study proposes the development of a browser extension with a Multi-Layer Security Filtering architecture to provide real-time protection. The system is designed across three defensive layers: Layer 1 employs the PhishTank API as a blacklist filter, Layer 2 uses the Tranco List API for whitelist verification, and Layer 3 applies an Extreme Gradient Boosting (XGBoost) model to detect emerging threats based on 22 lexical URL features. The system architecture is built on a Thin-Client model with a FastAPI backend to execute all security logic centrally on the server-side. The XGBoost model achieves an accuracy rate of 97.73% at an optimal threshold of 0.7 and an AUC score of 0.9927. Functionality testing demonstrates the system's success in performing automatic blocking and visual intervention through a blocking page. Furthermore, memory consumption testing validates the efficiency of the Thin-Client architecture, as the extension is proven to avoid persistent memory footprint allocation during URL capture, thereby ensuring lightweight browser performance. This implementation demonstrates that integrating list-based methods with artificial intelligence provides a proactive, accurate, and efficient detection response.

KEYWORDS: Phishing Detection, XGBoost, Browser Extension, Multi-Layer Architecture, URL Lexical Feature, Real-Time

I. INTRODUCTION

The global cybersecurity landscape currently faces unprecedented challenges due to the increasingly sophisticated and widespread evolution of phishing attacks. Phishing is defined as a cybercriminal act that exploits a combination of social engineering and technical deception to steal users personal credentials (APWG, 2025). Based on the latest data from the Phishing Activity Trends Report for the Second Quarter of 2025 released by the Anti-Phishing Working Group (APWG, 2025), a significant surge in criminal activity was recorded, with a total of 1,130,393 unique attacks reported, marking the highest record in history. These attacks not only target the financial sector but also manipulate users' trust in social media platforms, affirming that reliance on user vigilance alone is no longer sufficient. This phenomenon demands the existence of technical defense mechanisms capable of operating proactively and responsively directly within the user's browser environment (Shyni et al., 2025).

The most widely used anti-phishing solution today is the blacklist-based approach (Rashid et al., 2020). The blacklist method is a traditional security approach that works by maintaining a static repository of signatures or fingerprints representing previously known attacks (Guo, 2022). In the context of phishing detection, the system checks whether a URL visited by the user is registered in a collected phishing database. If a match is found, access to that website is immediately blocked to protect the user (Shah et al., 2020). This technique is employed by leading browsers through services such as Google Safe Browsing (Shyni et al., 2025). Although this method has the advantage of minimizing false positive rates for known threats, it carries a fundamental weakness in its reactive nature (Shyni et al., 2025). Blacklist-based solutions frequently fail to capture zero-day attacks or newly created phishing websites due to the time gap between the launch of a malicious website and the update of the security database, meaning such sites often claim victims before they can be registered (Shyni et al., 2025). This limitation represents a critical vulnerability, especially as attackers now commonly use short-lived domains to evade static detection.

To close this detection gap, the integration of artificial intelligence, particularly Machine Learning (ML), has been widely proposed as a predictive solution (Tang & Mahmoud, 2022). However, deploying complex detection models directly within browser extensions often encounters efficiency constraints (Tang & Mahmoud, 2022). Several approaches that rely on page content or visual website analysis have been shown to burden the user's device performance by introducing high latency when processing all Document Object Model (DOM) elements prior to classification (Shajahan et al., 2025; Tang & Mahmoud,

2022). Therefore, a detection strategy that is more lightweight yet maintains high accuracy is needed.

As a solution to these issues, this study proposes the development of a phishing detection system with a Multi-Layer Security Filtering architecture implemented entirely on the server side using the FastAPI framework to ensure high performance and minimal latency. This architecture is designed across three defensive layers: Layer 1 uses a Blacklist from the PhishTank API as the primary filter for known threats, Layer 2 uses a Whitelist from the Tranco List API to verify legitimate popular websites, and Layer 3 uses an XGBoost (Extreme Gradient Boosting) model to detect zero-day phishing threats based on lexical URL features.

The primary challenge remains in building a browser extension that integrates blacklist, whitelist, and lexical feature-based XGBoost mechanisms capable of detecting new phishing websites (zero-day) in real time without burdening device performance. Therefore, this study aims to implement the XGBoost model in classifying phishing URLs based on lexical features as a proactive solution for zero-day threat detection. The research scope is limited to the use of 22 lexical features from the URL-Phish dataset without conducting content or visual analysis, with the extension implemented specifically for Chromium-based browsers. All security logic processing is performed centrally on the server side using FastAPI. This research is expected to provide a theoretical contribution to the cybersecurity literature regarding the effectiveness of hybrid architectures, as well as offering practical benefits in the form of a protective tool for internet users against unrecorded phishing attacks.

II. LITERATURE REVIEW

A. Machine Learning Classification Algorithms for Phishing Detection

The use of machine learning algorithms has become a primary focus in addressing the increasingly complex evolution of phishing attacks. Supervised learning-based algorithms such as XGBoost, Random Forest, and Support Vector Machine (SVM) are frequently employed due to their ability to automatically recognize anomaly patterns in URL data. In recent years, the Extreme Gradient Boosting (XGBoost) algorithm has begun to dominate the literature owing to its efficiency in handling large-scale data with accuracy levels that surpass other conventional algorithms (Afandi et al., 2025; Das Guptta et al., 2024; Lukito & Handaya, 2025; Patil et al., 2023).

Extreme Gradient Boosting (XGBoost) is a gradient boosting-based machine learning algorithm first introduced by Friedman to improve performance in complex data analysis (Afandi et al., 2025). As a scalable and efficient tree boosting system, XGBoost was developed by the Distributed Machine Learning Community (DMLC) with the goal of optimizing computational resources through parallel processing (Das Guptta et al., 2024). This algorithm works by combining base functions and weights optimally, enabling it to produce models with a very high degree of fit to training data (Afandi et al., 2025).

Research conducted by Lukito (2025) demonstrates that XGBoost with hyperparameter tuning is capable of achieving accuracy up to 96% in distinguishing genuine from fraudulent websites (Lukito & Handaya, 2025). This is consistent with the findings of Das Guptta et al. (2024), who evaluated hybrid features (URL and hyperlink) using XGBoost and achieved a maximum accuracy of 99.17% (Das Guptta et al., 2024). The advantage of XGBoost lies in its ability to deliver an optimal balance between sensitivity (recall) and prediction stability, which is critical to ensure that the system neither misses malicious websites nor incorrectly blocks legitimate ones.

B. Implementation of Phishing Detection in Browser Extensions

A browser extension is a lightweight software tool designed to enhance functionality and customize the user experience within a web browser (Afandi et al., 2025). In the context of cybersecurity, extensions function as middleware or an intermediary layer between the user and the internet, monitoring browsing activity in real time (Thaqi et al., 2024). This client-side detection implementation minimizes latency, as the feature extraction and classification processes can be carried out through seamless integration between the browser and backend services.

Research by Shyni et al. (2025) through the "Phish Defender" system underscores the importance of intervention mechanisms in extension-based detection systems. The extension was designed to identify threats through structural URL inspection and domain reputation analysis, subsequently responding with warning banners and automatic redirection of users to safe pages. The operational approach of running silently in the browser background while remaining responsive upon threat detection has become the standard for user comfort, ensuring that browsing sessions are not disrupted by the detection process.

Furthermore, the effectiveness of browser extensions is highly dependent on their capacity to perform real-time analysis. Research by Afandi et al. (2025) through the "GuardSurfing" tool demonstrated that using the Flask framework to serve the XGBoost model enables the extension to respond to web traffic extremely quickly. The success of this implementation shows that even complex machine learning models can be executed efficiently within a browser environment when supported by an optimized data communication pathway between the extension's content scripts and the detection API.

C. Implementation of Multi-Layer Security Architecture

Multi-layer security filtering is a strategic approach designed to enhance the effectiveness and quality of phishing detection by systematically combining several system intervention methods (Adu-Manu & Ahiable, 2023). In this architecture, the system does not rely solely on one algorithm; instead, it combines several defense mechanisms such as whitelists,

blacklists, and machine learning to create more robust protection.

A whitelist functions to identify and grant exceptions to domains or websites that are already known to be legitimate and trustworthy (Adu-Manu & Ahiable, 2023; Le Pochat et al., 2019). In a phishing detection system, the primary purpose of using a whitelist is to minimize the occurrence of false positives, where safe and popular websites are incorrectly classified as threats. By performing an initial validation against a list of popular websites, the workload on the detection engine in subsequent layers can be significantly reduced, thereby improving the overall efficiency of the system.

Research by Adu-Manu and Ahiable (2023) designed a multi-layer phishing detection algorithm that specifically integrates the PhishTank repository, a blacklist, and a whitelist into a single unified framework. Their findings indicate that combining these strategies provides a more accurate means of recognizing and restricting access to malicious websites compared to using any single method alone. With a stable whitelist layer such as one utilizing Tranco ranking data the system can prevent false positive blocking of popular websites that have already been verified as safe.

The selection of reliable blacklist and whitelist databases is a crucial component in a multi-layer security architecture. The blacklist layer utilizes PhishTank, a non-profit collaborative repository that enables the community to openly report and verify phishing URL data (Adu-Manu & Ahiable, 2023). Through the PhishTank API, the system executes automated queries to validate threats based on the consensus of this global security community (Adu-Manu & Ahiable, 2023). Meanwhile, the whitelist layer implements the Tranco List, a research-oriented ranking list that aggregates data from various providers using a robust methodology to filter out irrelevant daily fluctuations (Le Pochat et al., 2019). This approach yields high stability, with a composition variation of merely 0.6% per day (Le Pochat et al., 2019).

The integration of blacklists, whitelists, and machine learning models creates a comprehensive defense system, where each layer complements one another to cover the security gaps of the other layers.

D. Lexical URL Feature Analysis and Zero-Day Detection Strategy

The research focus on lexical URL features is highly relevant due to their ability to detect attacks without requiring the full download of website content. Lexical analysis examines the structure of URL strings such as domain length, number of subdomains, and the presence of special characters that attackers frequently use to disguise malicious links. This URL-based detection approach is critical for handling zero-day phishing attacks, where the attack URL signatures have not yet been registered in any blacklist database.

Research by Fernando et al. (2025) revealed a critical 16-hour window between the first victim falling prey and a new phishing website being successfully mitigated by traditional blacklists (Fernando et al., 2025). During this waiting period, static reputation-based security systems frequently fail to provide adequate protection. Therefore, the use of lexical features combined with machine learning represents a proactive solution for instantly recognizing structural patterns in malicious URLs, even for newly launched attacks.

Based on a systematic review of prior research, several key research gaps have been identified that provide a strong foundation for the present study. These gaps include: (1) the implementation of XGBoost algorithms that has predominantly been conducted on static datasets within experimental environments, meaning that single-layer systems in existing browser extensions carry a high risk of false positives on popular sites due to the absence of an initial reputation filter; and (2) previously developed multi-layer architectures that have not yet leveraged advanced classification algorithms such as XGBoost as their primary detection engine to address zero-day attack challenges. Therefore, this study aims to fill these gaps by integrating XGBoost into a Multi-Layer Security Filtering architecture that combines a Blacklist layer (PhishTank API), a Whitelist layer (Tranco List API), and a Machine Learning layer to provide more responsive and accurate real-time protection on the user side.

III.METHODOLOGY

A. Research Design

This study employs the Research and Development (R&D) method, which is a research method used to produce a specific product and test its effectiveness (Okpatrioka, 2023). In this context, the product developed is a phishing detection system in the form of a browser extension that integrates a multi-layer security architecture with the XGBoost algorithm. The system development process follows the Software Development Life Cycle (SDLC) framework, encompassing the stages of literature study, analysis, design, implementation, and testing.

B. Requirements Analysis

Data Requirements: This study utilizes the secondary dataset URL-Phish obtained from the Mendeley Data repository (Linh & Hung, 2025). The selection of this dataset was based on the availability of lexical features that have been numerically engineered.

Functional Requirements: The server-side system must be capable of executing a detection mechanism for every incoming URL, consisting of: (1) Blacklist Validation (Layer 1); (2) Whitelist Validation (Layer 2); and (3) XGBoost Prediction (Layer 3). On the client side, the browser extension must be able to run in the background, monitor every navigation event, transmit the active URL to the server, receive and interpret JSON responses, and automatically redirect the user to a blocking page when the server returns a "phishing" status.

Non-Functional Requirements: The browser extension must be fully compatible with Chromium-based web browsers, such as Google Chrome, Microsoft Edge, Brave, and Opera.

C. System Architecture Design

The system architecture consists of two main interconnected components, as illustrated in Figure 1.

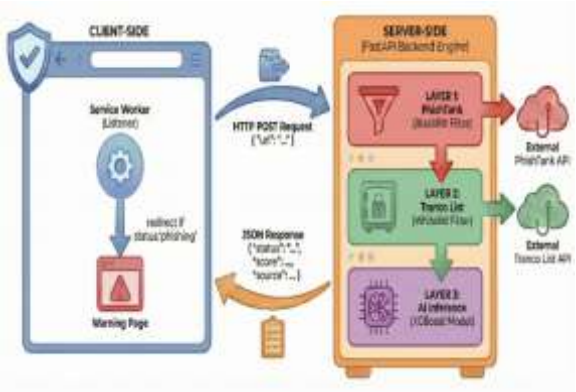

Figure 1. System Architecture Design

Client Side (Browser Extension): Its internal components consist of: (1) Service worker, which is a background script that listens for browser navigation events, captures the active URL, and executes redirection to the blocking page upon receiving a "phishing" response; (2) Blocking page, which is a full-screen blocking page that appears automatically when a threat is detected; and (3) Popup interface, which is an interface for displaying server connection status and blocking statistics.

Server Side (Backend): Its internal components consist of: (1) a Blacklist Handler (Layer 1), which manages the connection to the PhishTank API as an initial filter for globally confirmed threats; (2) a Whitelist Handler (Layer 2), which verifies domain reputation through the Tranco List API in real time to minimize false positives; and (3) an ML Inference Engine (Layer 3), which loads the pre-trained XGBoost model along with a Feature Extractor module that converts raw URLs into a vector of 22 lexical features.

The communication flow between the client and server operates as follows: (1) the extension transmits a JSON data packet containing the URL to the /scan endpoint on the server; (2) The server executes all three layers in parallel, then determines the final result based on the priority hierarchy of Layer 1 → Layer 2 → Layer 3; (3) the server returns a JSON response; and (4) the extension executes a blocking action or grants access according to the response received.

D. Technology Stack

Table I. Technology Stack

Category	Technology	Justification
Programming Language	Python 3.13, JavaScript	Python is used for the backend and machine learning, while JavaScript handles the interactive logic of the browser extension.
Framework & Library	FastAPI, Uvicorn, XGBoost, Scikit-learn, Pandas, NumPy, httpx, tldextract	FastAPI and Uvicorn as the API server. XGBoost and the data science stack for ML modeling. httpx handles asynchronous HTTP requests, and tldextract for domain analysis.
API	PhishTank API, Tranco List API	Third-party services used for blacklist and whitelist validation.
Development Tools	Visual Studio Code, Google Colab	Supports the development process, experimentation, and data visualization.

E. XGBoost Model Implementation

Data Acquisition and Exploration: The URL-Phish dataset is loaded in its original structure of 26 columns, consisting of 22 numerical features, 3 string-type reference columns (URL, domain, TLD), and 1 label column. Data exploration is conducted to validate the dataset dimensions and identify class distribution in order to understand the degree of data imbalance between the benign and phishing categories.

Preprocessing: Cleaning is performed, encompassing the identification and removal of missing values and duplicate entries. Feature selection is subsequently carried out by extracting the 22 numerical features and eliminating non-numerical text columns to ensure data compatibility with the XGBoost algorithm. The selected features span the categories of length-based, structural, count-based, and ratio-based characteristics, which collectively represent the textual properties of URLs. The complete list of these 22 features is presented in detail in the following Table II.

Table II. Lexical Feature List

No	Category	Name	Description
1	Length	url_len	Total number of characters in the URL string.
2		dom_len	Number of characters in the main domain part.
3		tld_len	Number of characters in the Top-Level Domain (TLD).
4		path_len	Number of characters in the URL path section.
5		query_len	Number of characters in the query string section.
6	Structure	is_ip	Returns 1 if the domain is an IP address, 0 if it is a regular domain name.
7		is_https	Returns 1 if the HTTPS protocol is used, 0 if HTTP.
8		entropy	Shannon Entropy value measuring the degree of randomness in the URL string.
9	Character Count	subdom_cnt	Number of detected subdomains (separated by dots).
10		dot_cnt	Number of dot (.) characters in the URL.
11		slash_cnt	Number of slash (/) characters in the URL.
12		dash_cnt	Number of hyphen (-) characters in the URL.
13		under_cnt	Number of underscore (_) characters in the URL.
14		eq_cnt	Number of equals sign (=) symbols in the URL.
15		qm_cnt	Number of question mark (?) symbols in the URL.
16		amp_cnt	Number of ampersand (&) symbols in the URL.
17		letter_cnt	Total number of alphabetic (letter) characters in the URL.
18		digit_cnt	Total number of numeric (digit) characters in the URL.
19		special_cnt	Total number of special (other symbol) characters in the URL.
20	Ratio	letter_ratio	Ratio of the number of letters to the total URL length.
21		digit_ratio	Ratio of the number of digits to the total URL length.
22		spec_ratio	Ratio of special characters to the total URL length.

Data Splitting: The cleaned dataset is divided into two subsets: a training set and a test set, with an 80:20 ratio. The split is performed using a Stratified Sampling technique to ensure that the class proportions in both the training and test sets remain identical to those of the original dataset. The parameter random_state=42 is applied to guarantee consistency of results across every experimental iteration.

Handling Data Imbalance: Due to the significant difference in sample count between the majority and minority classes, this study applies a cost-sensitive learning strategy.

This is achieved by configuring the scale_pos_weight parameter in XGBoost, which is dynamically calculated using the ratio of negative to positive samples according to the following formula:

$$\text{scale\_pos\_weight} = \frac{\sum \text{Negative Samples}}{\sum \text{Positive Samples}}$$

Model Configuration and Training: The model is initialized using the XGBoost algorithm with fixed hyperparameter settings to balance inference speed and accuracy, namely: n_estimators=100, max_depth=6, and learning_rate=0.1. The training process is focused on minimizing the logloss loss function to achieve the most stable model convergence.

Thresholding Experiment: In this stage, the decision threshold is tested across a range of 0.1 to 0.9. This experiment aims to observe the changes in accuracy, precision, recall, and F1-score metrics in order to identify the optimal point that minimizes the False Positive rate without sacrificing detection sensitivity. The evaluation procedure is elaborated further in sub-section H.

Model Export and Persistence: The model that has undergone validation and achieved the best performance is then saved in JSON format. This format is chosen based on its portability and memory efficiency, allowing the model to be loaded instantaneously by the FastAPI server backend to serve real-time detection requests.

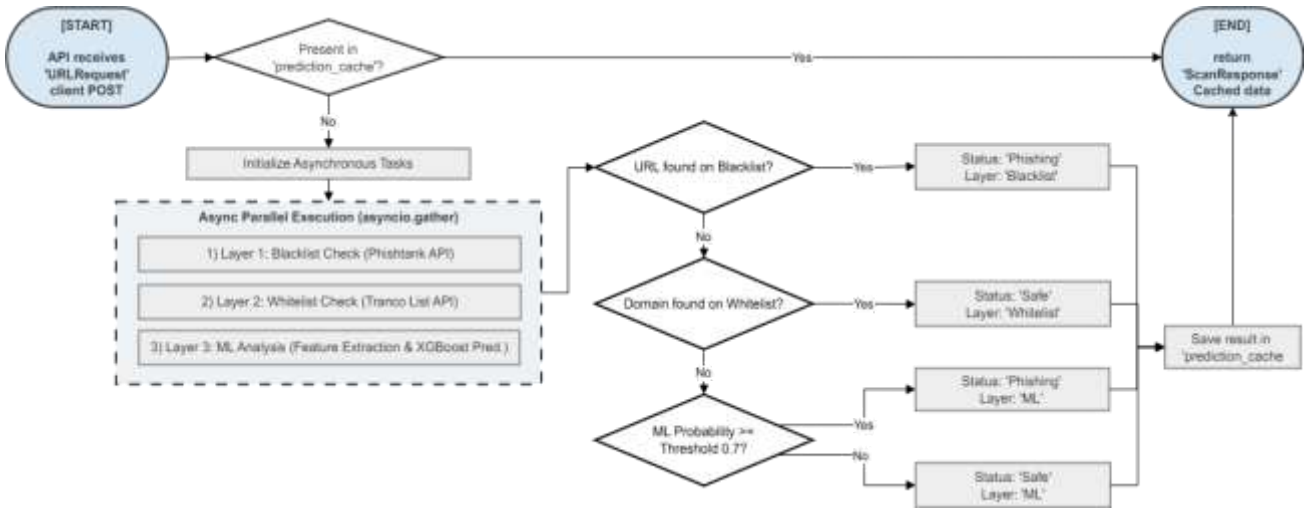

Figure 2. Multi-Laver Detection Workflow

F. Backend API Implementation

The backend API implementation in this system is designed as a control center connecting the user interface with the artificial intelligence model.

Multi-Layer Detection Workflow: As visualized in Figure 2, the detection process at the primary /scan endpoint begins by checking the prediction_cache. If the URL data is already stored, the system immediately returns the result to conserve resources. If the data is not found, the system executes three security layers in parallel using the asyncio.gather function to minimize response time.

Although the scanning processes run simultaneously, the final decision-making is based on a hierarchical approach with the following order of priority:

1. Layer 1 (Blacklist): Checks the URL against the PhishTank database via its API service. This layer holds the highest priority, if the URL is confirmed, the status is immediately designated as phishing with an absolute probability of 1.0.
2. Layer 2 (Whitelist): Verifies the domain reputation using the Tranco ranking list to determine whether the site is a trusted popular domain. If verified as safe, the status is designated as safe with a probability of 0.0. However, an exception applies to Private Suffix or Platform as a Service (PaaS) domains (such as vercel.app or github.io). These domains are forced to bypass this layer and must be analyzed by Layer 3 to mitigate the abuse of cloud services by phishing attackers.
3. Layer 3 (Machine Learning): Acts as the final decisive layer if the preceding two layers do not yield a conclusive result. In this stage, the extraction module first automatically converts the raw URL into 22 numerical features. Subsequently, the XGBoost model executes the analysis of these lexical features using a decision threshold.

All final results from this decision-making process are then saved back into the cache before being transmitted to the client to enhance processing efficiency for future requests.

Real-Time Feature Extraction: Prior to entering the ML prediction stage, the system runs the feature extractor module, which automatically converts a raw URL into 22 numerical features.

Response: The final detection result is transmitted back to the client-side in JavaScript Object Notation (JSON) format, structured according to the ScanResponse schema. The data includes the classification status, the identification of the active detection layer, the probability score, and the latency to measure the server-side processing duration. Furthermore, if the analysis is conducted through the Machine Learning layer, the response also includes a features attribute detailing the values of the 22 lexical features.

G. Browser Extension Implementation

The client-side implementation is realized in the form of a browser extension. This extension acts as a sensor module that actively monitors the user's navigation activity and provides a visual warning upon detection of a phishing threat.

Architecture: This browser extension is developed following modern browser architecture standards to ensure optimal security, privacy, and performance. Several key configuration components include:

1. Manifest File (manifest.json): Serves as the central configuration that declares the identity, permissions, and structure of the extension. This configuration requests the webNavigation and tabs permissions to monitor user URL transition activities, along with storage for local cache management. Furthermore, the host_permissions parameter is configured to include <all_urls> to enable scanning across all visited websites, while simultaneously granting data communication access to the backend API address.
2. Service Worker (background.js): Acts as a module-based background script that continuously runs asynchronously. This component is responsible for monitoring every URL change across browser tabs and transmitting it to the server.
3. User Interface (popup.html): Functions as the primary visual interface for users to view the information and security status of the currently visited website.
4. Blocking Page (warning.html): Serves as a security intervention interface that automatically overtakes

tactive browser tab when a URL is identified as phishing. This page displays a visual warning to prevent users from submitting credentials or sensitive information.

Automatic URL Monitoring Mechanism: The detection process begins through URL monitoring on every active tab. The technical implementation in the background.js script uses the chrome.tabs.onUpdated event listener to detect every instance of the user loading a new page or navigating to a different URL. Every captured URL is immediately forwarded to the API communication module for further analysis.

Client-Server Communication Integration: The api.js module is responsible for handling data transmission from the extension to the FastAPI backend via asynchronous methods. This integration encompasses: (1) Request Transmission: the URL is sent in JSON format via a POST request to the /scan endpoint; and (2) Response Processing: the system receives classification result data comprising the security status, probability value, and detected ML feature information.

H. Evaluation and Testing

The testing phase is divided into two primary focuses:

Classification Model Performance Testing: Evaluation of the XGBoost model is conducted entirely using a testing set that was not involved in the training process. The evaluation procedure is systematically designed with the following sequence of steps: (1) Multi-Threshold Analysis, (2) ROC Curve and AUC Analysis, (3) Metric Trend Visualization, and (4) Final Evaluation and Confusion Matrix.

Functional Testing: This method is used to verify the system's functionality from the user's perspective to ensure that every application component operates in accordance with its design. The functional testing scenarios in this study encompass: (1) validation of the system activation control via the "on" and "off" buttons on the extension popup; (2) testing of detection capability and response accuracy across each security layer (Blacklist, Whitelist, and Machine Learning); (3) the success of visual intervention through the full-screen blocking page interface in the browser; and (4) the functionality of the "Go Back" and "Proceed" buttons on the blocking page.

Memory Consumption Testing: This method aims to validate the memory efficiency of the Thin-Client architecture. Measurements were conducted using the Chrome Task Manager utility to monitor the Memory Footprint (RAM) allocation in real-time. The testing scenarios were designed to observe memory usage across various operational conditions, specifically when the system runs in the background (idle and scanning), when the user opens the Popup Interface, and when the system renders the Blocking Page. Furthermore, this testing evaluates the browser's Process-per-Extension mechanism to analyze memory accumulation characteristics when multiple interface components are active simultaneously.

IV. RESULT AND DISCUSSION

A. Data Preprocessing Results

The data preprocessing stage aims to ensure that the dataset used in training the XGBoost model is of high quality and free from anomalies that could degrade detection performance. The results of the data cleaning procedures are presented as follows:

Data Cleaning: Based on an initial inspection of the URL-Phish dataset comprising 116,600 rows, a number of invalid data entries were identified. The cleaning process was carried out in two main stages:

1. Missing Value Identification: A total of 14 rows with empty values across several attributes were found. All such rows were eliminated to prevent computational errors during model training.
2. Duplicate Data Identification: A total of 1,369 duplicate data entries were found. The removal of duplicate data was performed to prevent overfitting, whereby the model tends to memorize repeatedly occurring data rather than learning generalized patterns.

Feature Selection: Following the cleaning process, feature selection was conducted to discard non-numerical attributes. Of the 26 columns available in the original dataset, 3 text reference columns (URL, domain, TLD) were eliminated and 1 target column (label) was separated. The final result is a feature matrix consisting of 115,217 rows and 22 numerical feature columns.

Class Distribution: The data cleaning process caused a minor change in the overall class distribution; however, the proportion between the majority and minority classes remained preserved. The breakdown of label distribution before and after cleaning is presented in Table III:

Table III. Label Distribution Comparison

Label	Before Cleaning	After Cleaning
0	100,000	98,634
1	16,600	16,583
Total	**116,600**	**115,217**

The label distribution visualization following the cleaning stage is presented in Figure 3.

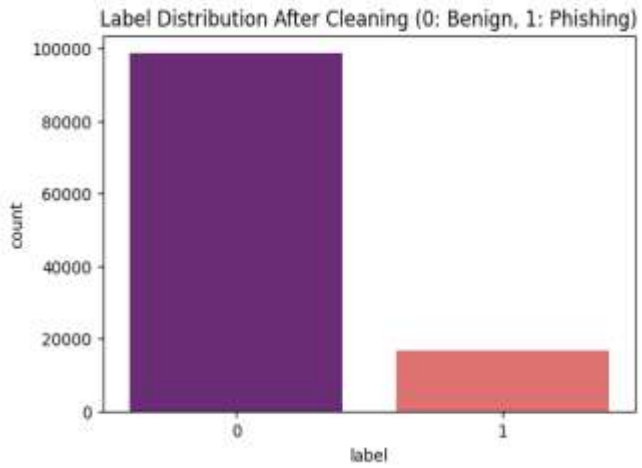

Figure 3. Label Distribution

The data imbalance visible in this figure provides the technical justification for the use of the scale_pos_weight parameter in the subsequent model implementation stage, ensuring that the model remains sensitive to the detection of the phishing class.

B. XGBoost Model Experiment Results

This section presents the results of testing the XGBoost model trained using optimal parameters. The experiment is focused on the model's ability to handle data imbalance and on determining the most accurate decision point through multi-threshold analysis.

Data Imbalance Handling Analysis: Based on the cleaned training data, a significant class imbalance exists, with a ratio of 5.9:1 between the benign class (78,907 samples) and the phishing class (13,266 samples). To mitigate model bias toward the majority class, this study configures the scale_pos_weight parameter in the XGBoost algorithm. Substituting the values into the scale_pos_weight formula yields the following calculation:

$$\text{scale\_pos\_weight} = \frac{78907}{13266} = 5.9481$$

With the application of scale_pos_weight, the model assigns a higher penalty weight to misclassifications of the phishing class, thereby significantly increasing the system's detection sensitivity toward threats without sacrificing overall classification performance.

Multi-Threshold Analysis: An experiment was conducted by testing the model's performance across a threshold range of 0.1 to 0.9 to find the optimal balance between Precision and Recall. The comparison of performance metrics is presented in Table IV.

Table IV. Performance Metrics Comparison by Threshold

Threshold	Accuracy	Precision	Recall	F1-Score
0.1	0.8929	0.5744	0.9876	0.7264
0.2	0.9346	0.6941	0.9762	0.8113
0.3	0.9533	0.7686	0.9665	0.8563
0.4	0.9647	0.8254	0.9575	0.8865
0.5	0.9706	0.8608	0.9491	0.9028
0.6	0.9755	0.8965	0.9379	0.9168
0.7	**0.9773**	**0.9185**	**0.9243**	**0.9214**
0.8	0.9776	0.9375	0.9044	0.9207
0.9	0.9766	0.9677	0.8664	0.9143

To further clarify the model's behavior in response to threshold changes, the metric trend visualization is presented in Figure 4.

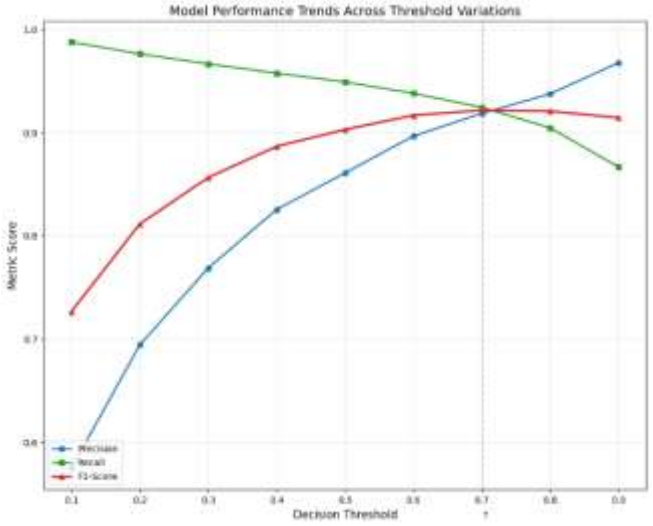

Figure 4. Model Performance Trends Across Threshold Variations

Based on the data in Table IV and the trends in Figure 4, a Precision-Recall Trade-off phenomenon can be observed. Increasing the threshold consistently raises the Precision value while lowering the Recall value. This indicates that the higher the threshold, the more selective the model becomes in classifying URLs as phishing, thereby reducing the rate of false accusations (False Positives); however, this simultaneously causes some threats to evade detection (False Negatives).

A threshold of 0.7 was selected as the optimal operational value as it yields the highest F1-Score (0.9214). At this point, the model is able to maintain a high level of confidence in detecting phishing (91.85%) while preserving excellent threat detection sensitivity (92.43%). This balance is critical for the system to provide maximum protection for users without disrupting the browsing experience due to the blocking of legitimate sites.

C. Final Performance Evaluation

This evaluation was conducted on the test set subset using the optimal threshold of 0.7 determined in the preceding experimental stage.

Confusion Matrix Analysis: The Confusion Matrix is used to thoroughly examine model performance by comparing actual values against predicted values. The Confusion Matrix visualization at a threshold of 0.7 is presented in Figure 5.

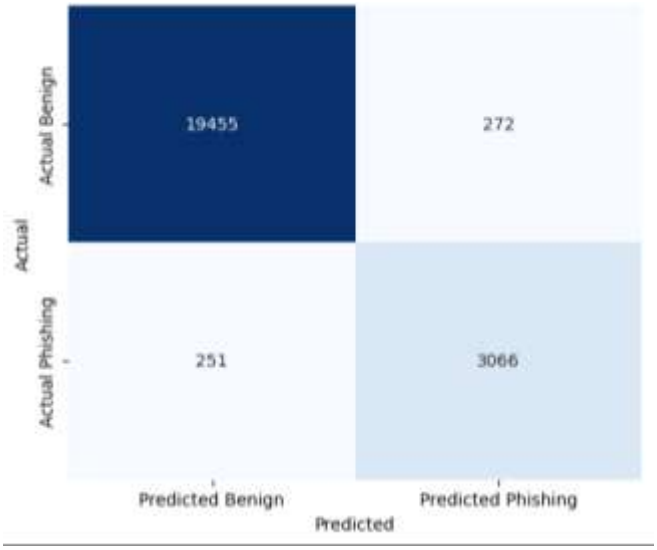

Figure 5. Confusion Matrix at Threshold 0.7

Based on Figure 5, the model demonstrates highly impressive performance in classifying both classes:

(1) True Negative (TN): The model successfully identifies the vast majority of benign sites (19,455 sites) with great accuracy. This is critical to ensure that users are not disrupted by false alarms when accessing legitimate sites.

(2) True Positive (TP): The model successfully captures the majority of phishing sites (3,066 sites) present in the test data.

(3) False Positive (FP): The model misclassified only a very low number of benign sites (272 sites) as phishing. Keeping this false alarm rate low is essential to prevent unnecessary blocking and maintain a seamless browsing experience.

(4) False Negative (FN): Only a minimal fraction of actual phishing sites (251 sites) evaded detection and were incorrectly classified as benign. This low number indicates that the extracted lexical features possess strong discriminative power, effectively minimizing the chance of threats slipping through.

Model Performance: To provide a more detailed quantitative assessment, the model's performance is summarized in the Classification Report as presented in Table V.

Table V. XGBoost Model Performance at Threshold 0.7

	Precision	Recall	F1-Score	Support
Benign	0.99	0.99	0.99	19727
Phishing	0.92	0.92	0.92	3317
Accuracy			0.98	23044
Macro Avg	0.95	0.96	0.96	23044
Weighted Avg	0.98	0.98	0.98	23044

The analysis in Table V shows that despite the imbalance in sample count (Support), the model is still able to produce an F1-Score of 0.92 for the phishing class. An overall accuracy of 98% demonstrates that the integration of the XGBoost algorithm with data imbalance handling through scale_pos_weight has achieved highly optimal results.

ROC Curve and AUC Score Analysis: The model's overall discriminative capability is measured using the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) value. The ROC curve visualization is presented in Figure 6.

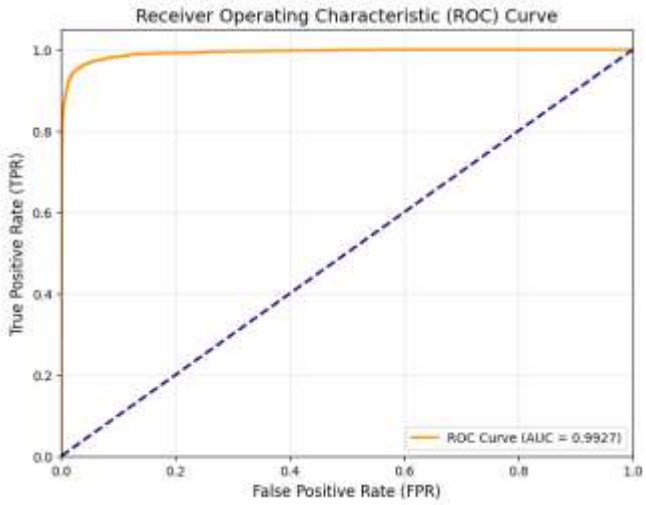

Figure 6. XGBoost Model ROC Curve

The obtained AUC value of 0.9927 (or approaching 1.0) indicates that the model has a very high probability of correctly distinguishing between benign and phishing sites across various threshold levels. This near-perfect AUC score provides strong validation that the model is ready to be deployed in a production environment (browser extension) to deliver real-time protection for users.

D. Feature Importance Analysis

To understand the basis of the model's decisions in classifying URLs, a feature contribution analysis was conducted using the Gini Importance method. The results for the 10 most influential lexical features are presented in Figure 7.

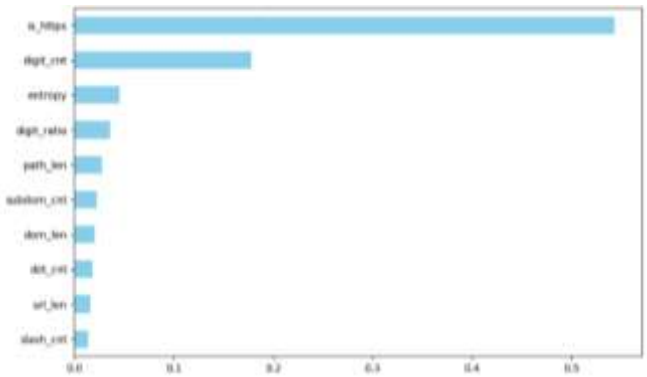

Figure 7. Top 10 Lexical Feature Contributions

Based on Figure 7, the hierarchical analysis of feature contributions reveals a highly sharp discrimination pattern. The is_https feature (use of the HTTPS protocol) emerges as

an overwhelmingly dominant contributor, far exceeding the contributions of all other lexical features. This finding strongly confirms the cybersecurity theory that phishing sites tend not to use a valid HTTPS security protocol due to the characters) and entropy (degree of character randomness) as the next most significant contributors. The dominance of these two features indicates that the XGBoost model has successfully learned the unique textual characteristics of phishing URLs, which frequently employ specific digit patterns or randomly generated domains, as reflected in the high digit count and character randomness.

E. System Implementation Results

This section presents the final results of the browser extension system implementation, encompassing the user interface of the browser extension across various detection scenarios as well as the automatic blocking mechanism through the blocking page.

Extension Popup Interface: The popup interface is the primary component that interacts directly with the user to provide real-time site security status. Based on testing, two main conditions are displayed by this interface:

certification process requiring domain ownership validation. The absence of valid HTTPS thus serves as the strongest indicator for the model in distinguishing malicious sites.

This is followed by the digit_cnt feature (number of digit Although length-based features (such as dom_len and url_len) and special character count features (such as dot_cnt and slash_cnt) also contribute, their weights are relatively small compared to security protocol indicators and specific textual patterns. This finding affirms that for the dataset used, security protocol and URL character complexity are sharper discriminators than URL length alon

1. Safe Website Scenario: Figure 8 shows the extension display when the user accesses a verified safe domain. The system displays a "Safe Site" label in green font to convey a sense of calm and security.
2. Phishing Site Scenario: Figure 9 shows the system's response upon detecting a malicious site. The extension panel automatically changes color to red with a "Phishing" warning.

Figure 8. Safe Website Scenario Display

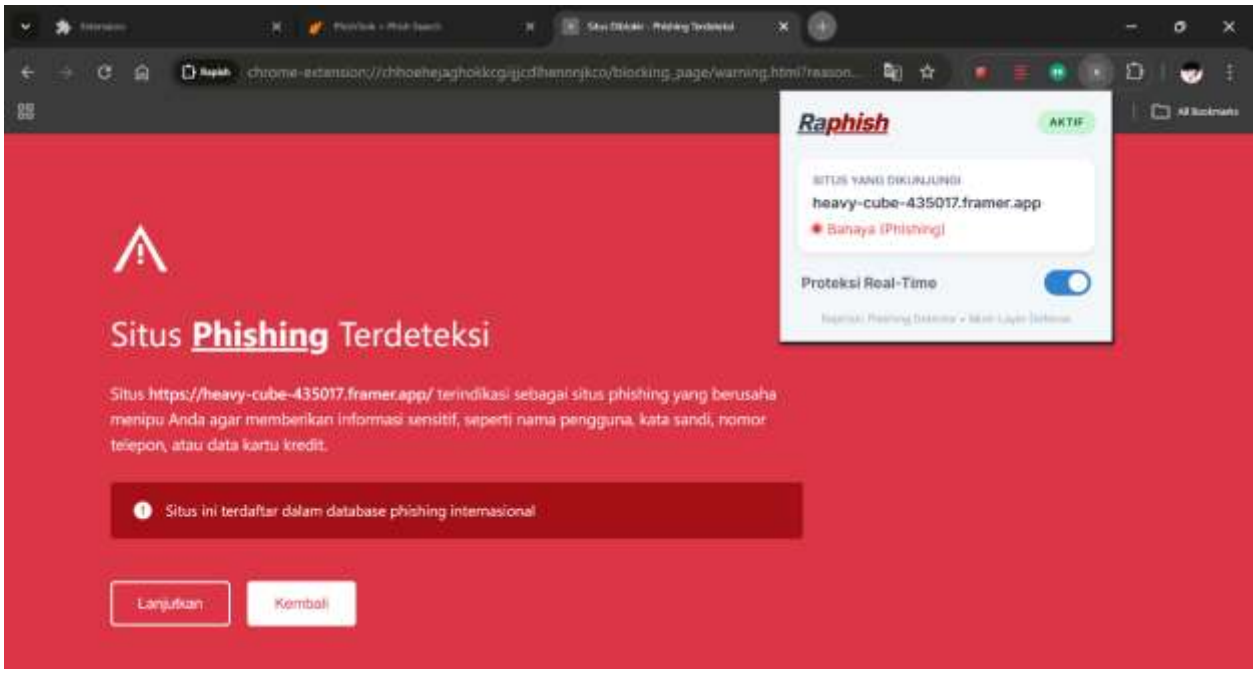

Figure 9. Phishing Website Scenario Display

Blocking Page: A critical feature of this system is its ability to forcibly halt user access upon detection of a phishing threat. This mechanism is realized through an automatic blocking page as shown in Figure 10. The blocking page is designed with visually striking elements to alert the user. The use of a red color scheme via the warning.css file is intended to produce a psychological danger signal effect.

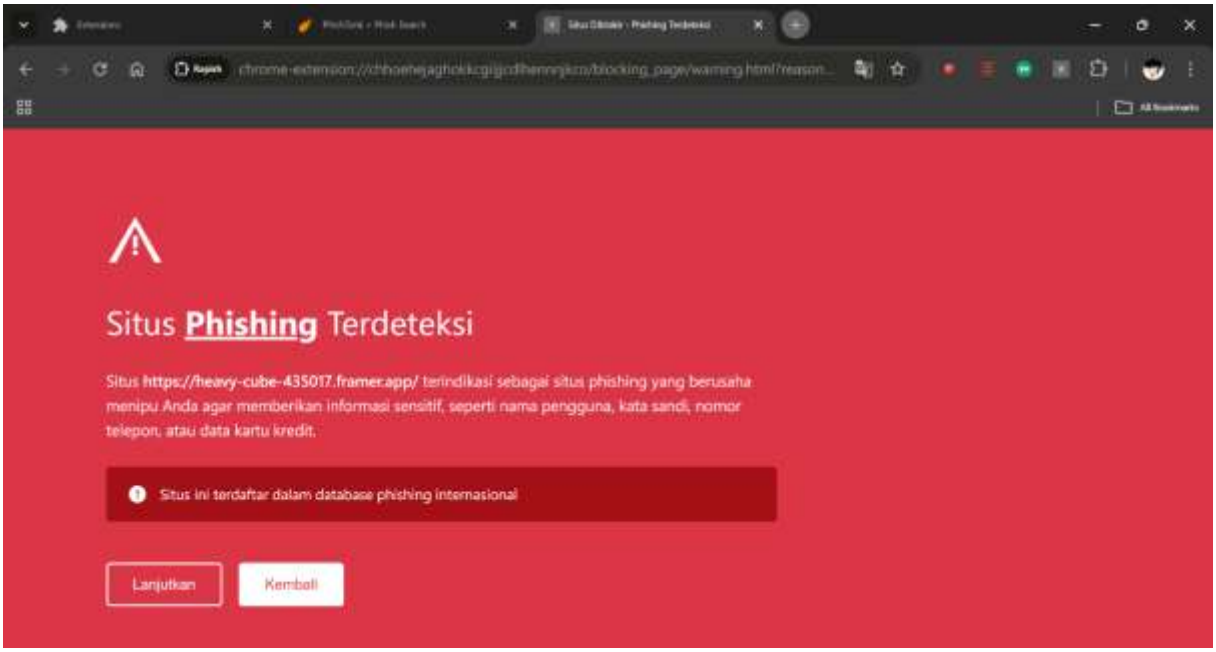

Figure 10. Blocking Page Display

When the backend API returns a phishing classification result, the background script immediately redirects the active tab's navigation to this page. The page effectively isolates the user from the malicious site and, through the implementation of warning.js, provides the user with the option to immediately return to the previous safe page.

F. Functional Testing Results

Functional testing was conducted to verify that all features operate in accordance with the requirements specification. The results of testing across five main scenarios indicate a "Pass" status for all system functionalities.

Table VI. Functional Testing Results

ID	Feature	Scenario	Expected	Status
TC-01	"on/off" Button	The user activates the "on/off" button on the application	The application checks the URL when the user activates the on button and vice versa	Pass
TC-02	Blacklist Detection	The user accesses an active phishing URL found in the PhishTank database	The extension displays blocking page	Pass
TC-03	Whitelist Detection	The user accesses a popular website found in the Tranco List	Access is permitted, no warning appears, and latency feels instant	Pass
TC-04	Phishing Detection (XGBoost)	The user accesses a simulated new phishing URL that is not present in the Blacklist or Whitelist	The system performs a prediction and the extension displays a blocking page	Pass
TC-05	Blocking Page	The user attempts to access a blocked website	The original page is completely covered by a security warning	Pass
TC-06	"Go Back" Button	The user presses the "Go Back" button on the blocking page	The system redirects the user to the previous page or closes the malicious tab	Pass
TC-07	"Proceed" Button	The user presses the "Proceed" link/button on the blocking page	The system grants access to the original URL	Pass

G. Memory Consumption Testing Results

The client-side memory consumption profile demonstrates low memory allocation across various operational conditions. Specific data regarding this memory usage (Memory Footprint) is presented comprehensively in Table VII.

Table VIII. Memory Consumption Testing Results

Phase	Scenario	Status	Consumption	Description
Idle	No active navigation in the browser.	Inactive	0 MB	Service Worker is terminated; no memory allocated.
Scanning	The URL is captured by the Service Worker and forwarded to the API.	Active (Background)	0 MB	No persistent memory allocated; all computational load is executed server-side.
Popup Interface	The extension menu panel is opened by the user	Active (UI)	25 - 33 MB	Memory is allocated to render interactive interface elements (DOM).
Blocking Page	The threat warning page takes over the screen	Active (UI)	26 - 30 MB	Memory is allocated to render the security warning visual interface.
Combined Condition	Popup and Blocking Page are simultaneously active.	Active (UI)	30 - 35 MB	Memory consumption is cumulative within a single process (Process-per-Extension).

H. Discussion

The findings of this study demonstrate that the Multi-Layer Security Filtering architecture in the browser extension successfully addresses the primary limitations of traditional phishing detection, namely the delay in blacklist updates and the high computational burden on the client side. The use of three defense layers operating in parallel has been proven to provide more comprehensive protection compared to single-layer approaches that rely solely on a database or a single algorithm (Shah et al., 2020). In comparison to systems that only employ static blacklist checking, this browser extension is capable of capturing zero-day threats through heuristic analysis of 22 URL lexical features processed by the XGBoost engine.

In terms of model performance, the XGBoost algorithm applied in this study achieves an accuracy rate of 97.73% with an AUC score of 0.9927. These results demonstrate superior performance when compared to prior research that also employed XGBoost yet only reached an accuracy of 95% (Shajahan et al., 2025). This improvement is attributed to the use of the scale_pos_weight parameter to handle imbalanced data, whereby a higher penalty weight is assigned to misclassifications of the minority class (phishing). This approach proves more effective in suppressing false negatives compared to the use of the Random Forest algorithm, which often encounters stability issues on datasets with imbalanced class ratios (Thaçi et al., 2024).

From an architectural standpoint, the FastAPI-based Thin-Client model implementation offers greater efficiency compared to the Flask framework used in similar research (Afandi et al., 2025). By leveraging the asynchronous mechanism (asyncio.gather), this browser extension is able to simultaneously execute checks against PhishTank, Tranco, and the ML engine, such that the total server latency is determined solely by the slowest process (Afandi et al., 2025). This addresses the third-party service stability challenges frequently raised in the development of deep learning-based detection frameworks (Tang & Mahmoud, 2021). By offloading the entire feature extraction workload to the server, the user's browser performance remains unaffected even as the system performs in-depth lexical feature analysis. Furthermore, the memory profiling validates the advantages of the system architecture integrated with the FastAPI backend. Empirical observations using the Chrome Task Manager indicate that during the idle and scanning phases, the extension does not trigger persistent memory allocation. The background script is only active ephemerally to capture and forward the URL to the server, after which it is automatically suspended by the browser. This demonstrates that the real-time protection system operates without imposing computational overhead on the user's device.

Finally, visual intervention through the blocking page offers an advantage in terms of User Experience. Several security systems only provide warning banners without offering clear technical details (Shyni et al., 2025). In contrast, this browser extension grants full authority to the user through "Go Back" or "Proceed" navigation options. This approach ensures that users are not only protected automatically, but also educated regarding the risk indicators present on the sites they visit (Shyni et al., 2025).

I. Limitation

Although this system demonstrates optimal performance in detecting phishing threats, several limitations need to be acknowledged in this study. Identifying these limitations aims to provide readers with context regarding the system's operational scope while also serving as a guide for future research development.

First, the detection mechanism in this system relies entirely on lexical feature analysis derived from the URL-Phish dataset. This means the system only evaluates the structural characteristics of URL strings without performing any analysis of text content, HTML scripts, or visual elements (images) on the visited web pages. This focus on lexical features was chosen to optimize detection speed, but carries a limitation in recognizing attacks that conceal their threats within page content.

Second, the artificial intelligence model used as the decision-making engine in the third layer is limited to the Extreme Gradient Boosting (XGBoost) algorithm. Although XGBoost is known to excel in handling tabular data, this study does not conduct an in-depth comparison with other classification algorithms beyond the scope of the defined experiments.

Third, all security logic processing from feature extraction to model inference is performed centrally on the server side using the FastAPI framework. This thin-client approach causes detection performance to be highly dependent on internet connection stability and the availability of the central server service. Should any network or server issues arise, the automatic detection function of the extension cannot operate optimally.

Fourth, security validation in this system is limited to three main layers: the PhishTank API as the blacklist data provider, the Tranco List API as the whitelist database, and predictions from the Machine Learning engine. The system's effectiveness is greatly influenced by the currency of data provided by these third-party services as well as the model's accuracy in processing new data.

Finally, the implementation of this browser extension is specifically designed for Chromium-based browsers, such as Google Chrome and Microsoft Edge. Consequently, the system does not yet support use on browsers with different engines (such as Mozilla Firefox or Safari) nor on mobile devices. This limitation is important to understand as the operational scope of the system in providing protection to users.

V. CONCLUSIONS

Based on the research findings, implementation, and testing conducted on the system, the following conclusions can be drawn:

1. XGBoost Model Performance: The implementation of the Extreme Gradient Boosting (XGBoost) algorithm in this system demonstrates exceptionally high performance in classifying phishing sites. With the optimization of the scale_pos_weight parameter to handle data imbalance, the model achieves an accuracy rate of 97.73% and an AUC score of 0.9927 at an optimal decision threshold of 0.7. This proves the model's reliability in detecting threats based on 22 URL lexical features.
2. Browser Extension Functionality: The browser extension successfully executes its layered protection function through the Multi-Layer Security Filtering architecture operating in parallel. Functional testing demonstrates that the system is capable of automatically performing visual intervention through a blocking page when a threat is detected by either the blacklist layer or the Machine Learning layer. All primary features, including real-time URL monitoring and cache management to accelerate response times, have functioned in accordance with the system design.
3. Memory Consumption Efficiency: The implementation of the Thin-Client architecture is proven to be highly lightweight for client devices. The extension does not trigger a persistent memory footprint during the idle phase or the URL capture process, ensuring that cybersecurity protection operates optimally without significantly burdening the user's browser performance.

ACKNOWLEDGMENT

The author wishes to express his sincerest gratitude to the Department of Informatics Engineering, Faculty of Science and Technology, Universitas Islam Negeri Sultan Syarif Kasim Riau, for providing the essential academic environment and facilities that made this research possible. The department's commitment to technical excellence and the continuous support from the faculty members have been instrumental in the development of the system and the successful completion of this study. Furthermore, the author is deeply thankful for the laboratory resources and the constructive academic atmosphere that provided a solid foundation for both the research and development phases of this project. This work is a testament to the high standards of education and professional guidance fostered within the department.

REFERENCES

1. Adu-Manu, K. S., & Ahiable, R. K. (2023). Detecting Phishing Using a Multi-Layered Social Engineering Framework. Journal of Cyber Security, 5(0), 13–32. https://doi.org/10.32604/jcs.2023.043359
2. Afandi, H. A. K., Al-Dzaki, M. L. F., Qomariasih, N., & Wildana, R. A. (2025). GuardSurfing : Ekstensi Browser sebagai Alat Bantu Deteksi Website Phishing dengan Metode Klasifikasi XGBoost untuk Deteksi URL Phishing Berbasis Flask Framework. Info Kripto, 19(2), 73–85. https://doi.org/10.56706/ik.v19i2.124
3. APWG. (2025). PHISHING ACTIVITY TRENDS REPORT 2nd Quarter 2025 (Issue August).
4. Das Guptta, S., Shahriar, K. T., Alqahtani, H., Alsalman, D., & Sarker, I. H. (2024). Modeling Hybrid Feature-Based Phishing Websites Detection Using Machine Learning Techniques. Annals of Data Science, 11(1), 217–242. https://doi.org/10.1007/s40745-022-00379-8
5. Fernando, M., Mahmood, A., Chowdhury, M. J. M., & He, Z. (2025). PhishLex: A Real-Time Machine Learning Model for Zero-day Phishing Detection by Systematizing URL Techniques. SSRN Electronic Journal. https://ssrn.com/abstract=5205908
6. Guo, Y. (2022). A review of Machine Learning-based zero-day attack detection: Challenges and future directions. Computer Communications, 198, 175–185. https://doi.org/10.1016/j.comcom.2022.11.001
7. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., & Joosen, W. (2019). TRANCO: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. 26th Annual Network and Distributed System Security Symposium, NDSS 2019, February. https://doi.org/10.14722/ndss.2019.23386
8. Linh, D. M., & Hung, T. C. (2025). A feature-engineere d dataset of b enign and phishing URLs for machine learning and large language models evaluation. Data in Brief, 63, 112162. https://doi.org/10.1016/j.dib.2025.112162
9. Lukito, & Handaya, W. B. T. (2025). Deteksi Website Phishing Menggunakan Teknik Machine Learning. Jurnal Informatika Atma Jogja, 6(1), 69–80. https://doi.org/10.24076/infosjournal.2023v6i01.1268
10. Okpatrioka. (2023). Research And Development (R & D) Penelitian Yang Inovatif Dalam Pendidikan. Jurnal Pendidikan, Bahasa Dan Budaya, 1(1), 86–100.
11. Patil, S., Patil, M., & Chinnaiah, K. (2023). Machine Learning and Deep Learning for Phishing Page Detection. Research Reports on Computer Science, Selected Papers from MIND-2022, 45–54.
12. Rashid, J., Mahmood, T., Nisar, M. W., & Nazir, T. (2020). Phishing Detection Using Machine Learning Technique. Proceedings - 2020 1st International Conference of Smart Systems and Emerging Technologies, SMART-TECH 2020, 43–46. https://doi.org/10.1109/SMART-TECH49988.2020.00026
13. Shah, B., Dharamshi, K., Patel, M., & Gaikwad, D. V. (2020). Chrome Extension for Detecting Phishing Websites. International Research Journal of Engineering and Technology (IRJET), 07(03), 2958–2962. https://doi.org/10.2139/ssrn.4398136
14. Shajahan, S., George, T., V, J. P., M, S. K., Krishna, S., Krishna, S., & Jagadish, N. (2025). Phishing Website Detection Using Machine Learning. International Journal of Research Publication and Reviews, 6(3), 10039–10043. https://doi.org/10.55248/gengpi.6.0325.1325
15. Shyni, S. A., Harish, A., Karthikeyan, P., Zerro, S. W. A., & Yogesh, R. (2025). Phish Defender: Real-Time Detection Of Phishing Websites Using A Browser Extension. Metallurgical and Materials Engineering, 1109–1117.
16. Tang, L., & Mahmoud, Q. H. (2022). A Deep Learning-Based Framework for Phishing Website Detection. IEEE Access, 10, 1509–1521. https://doi.org/10.1109/ACCESS.2021.3137636
17. Thaqi, L., Halili, A., Vishi, K., & Rexha, B. (2024). NoPhish: Efficient Chrome Extension for Phishing Detection Using Machine Learning Techniques. 1–21. http://arxiv.org/abs/2409.10547