National College of Ireland

# Machine Learning Powered Real Time Web Vulnerability Detection: Browser Extension for Website Security

MSc Research Project
MSc Cybersecurity

Sourabhkumar Mishra
Student ID: X23300949

School of Computing
National College of Ireland

Supervisor: Mark Monaghan

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Sourabhkumar Mishra |
| **Student ID:** | x23300949 |
| **Programme:** | Msc Cybersecurity **Year:** 2024-25 |
| **Module:** | Practicum |
| **Supervisor:** | Mark Monaghan |
| **Submission Due Date:** | 11/08/2025 |
| **Project Title:** | Machine Learning Powered Real Time Web Vulnerability Detection: Browser Extension for Website Security |
| **Word Count:** | 7175 **Page Count** 20 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Sourabhkumar Mishra |
| **Date:** | 11/08/2025 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Machine Learning Powered Real Time Web Vulnerability Detection: Browser Extension for Website Security

Sourabhkumar Mishra
x23300949

**Abstract**

The proliferation of web applications has led to a parallel increase in sophisticated cyber threats, with Cross-Site Scripting (XSS) and SQL Injection (SQLi) remaining among the most prevalent and damaging vulnerabilities. Traditional security measures often struggle to provide real-time, client-side protection. This research project addresses this gap by developing a dual-component solution: a high-accuracy machine learning model and a corresponding real-time browser extension. The core of the project is a Logistic Regression model trained on a diverse and augmented dataset of over 111,000 web payloads, achieving a remarkable 99.88% accuracy in distinguishing between malicious and benign inputs. This model uses a hybrid feature engineering approach of 30 hand crafted features and 1000 TF-IDF n-gram features. This model is practically realized through an Edge browser extension that passes user input and page content to our model and provides immediate visual feedback to users on potential threats. It then serves as a first line of defense against accidental submissions of malicious payloads. The evaluation shows the model's promising performance, and the extension's functionality to detect an abundance of XSS and SQLi attacks, thus certainly representing a development in proactive, user-focused web security.

# 1 Introduction

Digital space has become a synonym for society; web applications are the dominant medium for communication, commerce and information sharing. This level of ubiquitous access has opened an extensive attack surface for attackers. Multiple cybersecurity reports list injection attacks as commonly the most common and severe web application vulnerability with where attacks are primarily using Cross Site Scripting (XSS) or SQL Injection (SQLi) (Kaur, Garg and Bathla, 2023). Injection attacks involve engineering trust during data processing (SQLi) or delivering content (XSS) through the web application by the end-users browser session. This can lead to the theft of sensitive data, account takeovers, reputational damage, and so on. Traditional web security paradigms have often focused on server-side defenses, such as Web Application Firewalls (WAFs) and static code analysis. While essential, these methods can be bypassed by sophisticated, obfuscated payloads and do not typically offer protection at the point of user interaction. This leaves a critical window of vulnerability where a user might unknowingly submit a malicious payload propagated through social engineering or a compromised source. There is a clear and present need for a proactive, client-side security

mechanism that can identify threats in real-time, before they are transmitted to the server or executed within the browser.

Machine learning has emerged as a powerful paradigm for cybersecurity, capable of identifying complex patterns in data that elude signature-based systems (Alaoui and Nfaoui, 2022; Salem et al., 2024). By training models on vast datasets of known malicious and benign inputs, it is possible to create classifiers that can predict the nature of new, unseen payloads with a high degree of accuracy. This research leverages this potential to address the challenge of real-time vulnerability detection.

The central research question guiding this project is: Can a machine learning model, integrated into a browser extension, provide effective, real-time detection of XSS and SQL Injection vulnerabilities at the client-side without significantly impacting user experience?

To answer this question, the following research objectives were established:

1. To curate and preprocess a comprehensive dataset of web attack payloads, including XSS and SQLi, and augment it with obfuscated variants to simulate real-world attack techniques.
2. To engineer a robust set of features capable of distinguishing between malicious and benign payloads, combining lexical, structural, and statistical properties.
3. To train, evaluate, and select a high-performance machine learning model that achieves near-perfect accuracy in classifying web payloads.
4. To design and implement a browser extension for Microsoft Edge that integrates the detection logic from the trained model to perform real-time analysis of user inputs and web page content.
5. To evaluate the complete system, assessing both the statistical performance of the machine learning model and the functional effectiveness of the browser extension against a range of test payloads.

This project's contribution is a novel, dual-component system that bridges the gap between sophisticated, offline model training and practical, real-time, client-side application. The result is a tool that not only identifies threats but also educates users by providing immediate visual feedback, as illustrated in the conceptual diagram in Figure 1.

This report consists of the following divisions: Section 2 provides a critical review of the literature regarding machine learning for web security; Section 3 outlines the research methodology; Section 4 contains the design specification for the machine learning pipeline and browser extension; Section 5 details the implementation of the machine learning pipeline and browser extension; Section 6 includes a comprehensive evaluation of the performance of the machine learning pipeline and browser extension; and Section 7 concludes the report, providing a summary of the work and directions for future research.

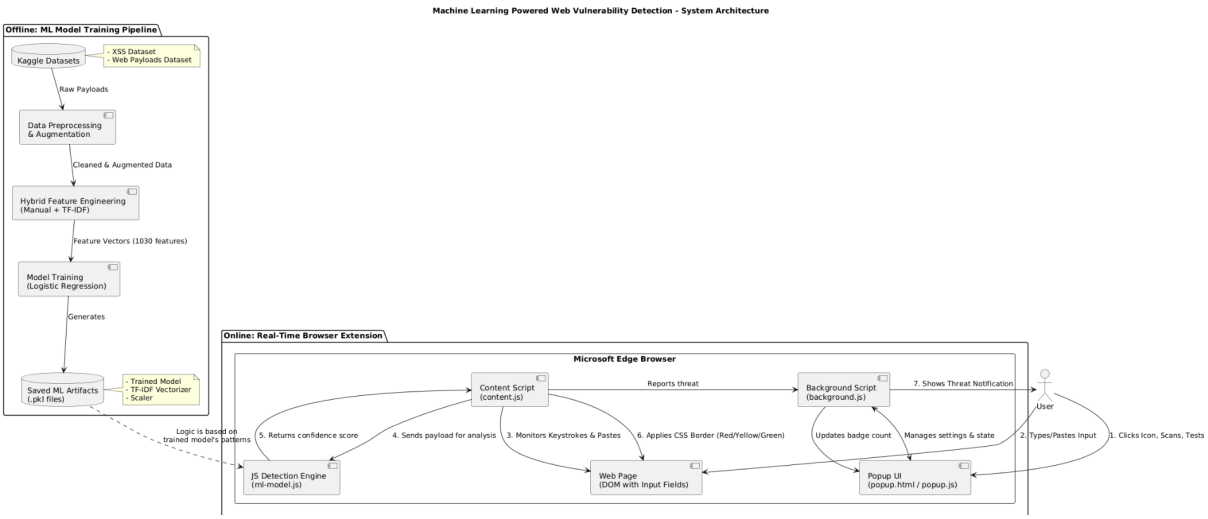


**Figure 1:** Conceptual architecture of the proposed system showing the offline ML model training pipeline and the online real-time detection done by the browser extension.

# 2 Related Work

Machine learning (ML) applications to cybersecurity is a quickly growing area of interest with substantial research stemming from detection and mitigation of web-based threats. This section reviews the literature relevant to this project critically, with specific attention to: ML for vulnerability detection; XSS and SQLi - specific approaches; and security considerations of browser extensions.

## 2.1 Machine Learning for Web Vulnerability Detection

Machine learning to detect web vulnerabilities has gained popularity as a more dynamic means of detecting these threats than static, signature-based methods. A systematic literature review by Shiri Harzevili et al. (2024) focuses on automated software vulnerability detection with the help of ML. The review notes a trend toward using deep learning models as well as the critical demand for high-quality labeled datasets. Among the authors' observations are the well-known difficulties presented by the "concept drift" wherein new attack vectors are created not appearing in the training data. This emphasizes the importance of having a robust and diverse training set, a principle directly addressed by this project through data combination and augmentation.

ALSO, Alaoui and Nfaoui (2022) delimitak in using deep learning for web applications hit detection and conclude that although models such as LSTMs and CNNs are promising, their complexity can also be a barrier toward real-world deployment, particularly within resource-constrained environments such as a browser extension. Thus justifying this project's approach to choosing a highly efficient classical model such as Logistic Regression, which offers better performance and computation cost balance for client-side implementations, with some Favor:. Chughtai et al. (2024) investigate deep learning trends in web security much however mention a lot on the fact that solutions have to be proposed to handle obfuscated as well as zero-day attacks all challenges that this research was geared towards countering via means such as comprehensive engineering and data augmentation.

Mazhar et al. (2023), and Alwhbi, Zou, and Alharbi (2024) research on the applications of ML in the field of cybersecurity, which is largely related to smart grids as well as encrypted network traffic analysis. This study may not directly delve into web payloads; nevertheless, both underscore the evidence of practical applications of ML in pattern recognition in front of convoluted adversarial data streams. In this comprehensive survey on ML-enabled encrypted network traffic analysis, whether or not such fields will face or have ever been confronted with issues pertaining to extracting meaningful features from this as if opaque data, similar to delineating malicious intent from a string of text. The techniques discussed such as statistical and payload-based feature extraction, are analogous to the methods used herein.

## 2.2 Detection of Specific Injection Attacks

A significant portion of the literature focuses on specific attack vectors like XSS and phishing, which often involve similar payload characteristics. Kaur, Garg, and Bathla (2023) present an extensive review of ML techniques for XSS detection. They categorize features into static, dynamic, and hybrid, and note that while deep learning models are popular, traditional models like SVM and Naive Bayes still perform competitively, especially when paired with strong feature engineering. This project's use of a hybrid feature set (manual and TF-IDF) aligns with their findings for achieving high accuracy. Kissoon and Bekaroo (2023) analyze tools for detecting XSS attacks, finding that many existing solutions are either server-side or operate as offline scanners, which highlights the gap for a real-time, client-side preventative tool that this research aims to fill.

Phishing detection is another closely related area. Tang and Mahmoud (2021) survey ML-based solutions for phishing website detection, where URL and page content features are paramount. Many of the features used to detect phishing, such as the presence of suspicious keywords, URL length, and special characters, are directly applicable to detecting injection payloads. Similarly, works by Blancaflor et al. (2024) and Jadhav and Chandre (2025) review neural network and other ML approaches for phishing detection, reinforcing the validity of feature-based classification for malicious web content. Thakur et al. (2023) provide a systematic review on deep learning for phishing email detection, another domain where textual analysis is key to identifying malicious intent.

## 2.3 Browser-Based Security and AI Integration

The browser is the primary execution environment for web content, making it a logical place for client-side security enforcement. Zonta and Sathiyanarayanan (2024) conduct a holistic review on detecting malicious browser extensions and links using deep learning, acknowledging the browser as a critical control point. Their work points out the risks posed by extensions themselves, a factor that necessitates careful design of any security-oriented extension to ensure it does not introduce new vulnerabilities. Allauddin and Lokhande (2024) further analyze security risks in browser extensions, emphasizing the need for privacy-preserving designs. This project addresses this by performing all analysis locally within the browser, sending no user data to external servers, a key security feature.

The integration of AI directly into front-end applications is a growing trend. Goh, Ho, and Abas (2023) review the development and deployment of front-end deep learning web apps, discussing frameworks like TensorFlow.js that enable running models directly in the browser.

While this project uses a simplified JavaScript implementation of the model's logic for performance reasons, their work points to a future where more complex models could be run client-side. Ok (2025) discusses the future role of AI and predictive analytics in browser and email security, envisioning tools that can preemptively block threats, which aligns perfectly with the goals of this research. The works on real-time traffic analysis (Dhakad et al., 2023) and IoT security (Ejeofobiri, Victor-Igun, and Okoye, 2024; Gharbi et al., 2025) also speak to the broader trend of deploying ML-driven detection at the network edge for immediate response, a principle this project applies at the user's endpoint.

In summary, the literature confirms the viability of using machine learning for web vulnerability detection. However, a gap exists for a solution that combines the high accuracy of a robustly trained model with the practical, real-time, and privacy-preserving implementation of a browser extension. Many existing solutions are either server-side, offline, or use computationally heavy models unsuitable for client-side deployment. This research directly addresses this gap by creating a lightweight, performant browser extension powered by a highly accurate and efficiently trained classical machine learning model, offering a novel contribution to user-centric web security.

# 3 Research Methodology

This project adopted a quantitative, experimental research methodology to design, build, and evaluate a machine learning-powered system for real-time web vulnerability detection. The methodology is structured into a sequential pipeline, ensuring a rigorous and reproducible process from data acquisition to final system validation. The core phases of the methodology include data collection and preparation, feature engineering, model development and selection, and system implementation and evaluation.

The overall approach is rooted in the principles of supervised machine learning, where a predictive model is trained on a labeled dataset to learn the underlying patterns that differentiate between classes, in this case, malicious and benign web payloads.

The research procedure was executed in the following distinct stages:

1. **Data Collection and Consolidation:** A dataset of good quality forms the basis of any supervised learning task. Two datasets were acquired from the Kaggle setting to create a rich payload-collection to have variety and completeness.
    - **Web Application Pay Load Dataset:** The various malicious payloads available are classified according to their attack types(SQLi, XSS, and so on).
    - **Cross Site Scripting(XSS) Data set:** This dataset contains vast numbers of sentences and code snippets. It includes data that has been labeled pertain either having an XSS payload or being benign.

These datasets were loaded programmatically and combined in a single, homogeneous structure for later processing.

1. **Data Processing and Improving:** Row data is hardly ever directly available for a machine learning. It comprised several steps that were crucial. First, the consolidated data was cleaned by removing missing values and standardizing data types. Labels were unified into a binary classification scheme: 'malicious' and 'benign'. Duplicate payloads were removed to prevent model bias. Data augmentation was a very

important task in this phase. To enlarge the model detection ability against how evaded, real-world attacks could be safely generated obfuscations of some features. This included HTML entity encoding, URL encoding, and case variations as techniques, with significant improvement in the diversification and size of the training corpus that is very essential for developing a strong classifier.

2. **Feature Engineering and Representation**: The complete detection engine primarily continues to translate original text payloads into a numerical format that machines can understand and later analyze by a machine learning algorithm. Hybrid feature engineering has been adopted.
    - **Manual Feature Extraction:** More than 30 separate features were hand-crafted with domain knowledge associated with web attacks: lexical features (e.g., payload length, character diversity, number of digits/letters), structural features (e.g., presence and count of HTML tags, JavaScript keywords like 'script' or 'onclick'), and pattern-based features (e.g., presence of SQL keywords, comments, or encoding patterns).
    - **Automated Feature Extraction (TF-IDF):** For capturing a more substantial number of textual patterns without being enumerated, the Term Frequency-Inverse Document Frequency (TF-IDF) technique has been employed. A TF-IDF vectorizer was configured to analyze character n-grams (sequences of 1, 2, and 3 characters), such that a sparse matrix of 1000 features representing most significant character patterns across the dataset would be generated.

These two treat were combined together into a final feature vector of 1030 dimensions for each payload.

1. **Model Training, Selection, and Evaluation:** This phase involved training several candidate machine learning models on the engineered features and selecting the best performer.
    - **Data Splitting:** The dataset was split into training (60%), validation (20%), and testing (20%) sets to ensure unbiased evaluation. The training set was balanced using the Synthetic Minority Over-sampling Technique (SMOTE) to prevent the model from being biased towards the majority class.
    - **Model Selection:** Three powerful and widely-used classification algorithms were chosen for experimentation: RandomForest, XGBoost, and Logistic Regression.
    - **Training and Hyperparameter Tuning:** The models were trained on the balanced training set. Hyperparameters were optimized for a balance of high accuracy and training speed.
    - **Evaluation:** The performance of each model was rigorously evaluated on the held-out test set using a suite of standard metrics: Accuracy, Precision, Recall, and F1-Score. The confusion matrix was also analyzed to understand the model's performance on false positives and false negatives. The Logistic Regression model was ultimately selected due to its superior performance.
2. **System Implementation:** The final phase involved translating the research into a functional artifact. This was a two-part process.

- **Saving the ML Pipeline:** The trained Logistic Regression model and all associated preprocessing components (StandardScaler, PCA transformer, TF-IDF vectorizer) were serialized and saved into pickle files for future use.
- **Browser Extension Development:** A Microsoft Edge browser extension was developed using standard web technologies (JavaScript, HTML, CSS). The logic of the trained model was implemented in a simplified JavaScript class (VulnerabilityDetector) to run client-side. This involved creating content scripts to monitor web pages, a background script to manage extension state, and a popup interface for user control.

3. **Final System Validation:** The complete system was evaluated to confirm its functional correctness. This involved testing the browser extension against a curated list of both malicious (XSS and SQLi) and benign payloads, as specified in the project's test guide, to verify that it produced the expected real-time visual feedback and notifications.

# 4 Design Specification

The design of the system is architecturally divided into two distinct but interconnected components: the Machine Learning Detection Core and the Real-Time Browser Extension. This section details the specifications for each.

## 4.1 Machine Learning Detection Core

The Machine Learning Detection Core is the intelligence engine of the system. Its design is focused on achieving the highest possible accuracy in classifying web payloads while maintaining a structure that can be understood and partially replicated in a different programming environment (JavaScript).

### 4.1.1 Data Ingestion and Preprocessing Architecture

The system is designed to ingest multiple raw data sources. The architecture specifies a data pipeline that first consolidates these sources, then applies a series of cleaning and standardization steps. This includes handling missing data, unifying labels into a binary (malicious, benign) format, and deduplication. A key design element is the data augmentation module, which specifically targets malicious payloads and applies multiple obfuscation techniques (URL encoding, HTML entity encoding, case mixing) to generate new training samples, thus hardening the model against common evasion tactics.

### 4.1.2 Hybrid Feature Engineering Module

The design specifies a sophisticated, two-pronged feature extraction module to create a rich numerical representation of each payload.

- **Manual Feature Set:** A set of 30 meticulously designed features are to be extracted. These are categorized as follows:
  - **Basic Statistical Features:** payload_length, character_diversity, entropy. These provide a high-level view of the payload's complexity and randomness.

- **Character-Level**
  **Features:** num_digits, num_letters, num_special_characters. These capture the composition of the payload.
- **XSS-Specific Features:** Presence and count of tags like <script>, <img>, <iframe>; JavaScript protocol handlers like javascript:; event handlers like onclick, onload; and dangerous functions like alert(), eval().
- **SQLi-Specific Features:** Presence of SQL keywords like SELECT, UNION; operators like OR, =; comment characters like --, /*; and tautological patterns like '1'='1'.
- **Encoding-Specific Features:** Detection of URL encoding (%XX), HTML entities (&...;), and Unicode escaping (\uXXXX).

- **TF-IDF Feature Set:** To complement the manual features, the design specifies the use of a TfidfVectorizer. This component is configured to analyze character-level n-grams (from 1 to 3 characters). This allows the model to learn subtle, recurring character sequences that are indicative of malicious payloads, even if they do not match a predefined keyword. The design caps the number of TF-IDF features at 1000 to balance comprehensiveness with computational efficiency.

The final feature vector for each payload is a concatenation of these 30 manual features and 1000 TF-IDF features, resulting in a 1030-dimensional vector.

### 4.1.3 Modeling and Training Specification

The system is designed to be model-agnostic, allowing for the training and comparison of multiple classifiers. The specification includes three candidate models: Logistic Regression, RandomForest, and XGBoost. The training process is designed as follows:

1. **Scaling:** All features are scaled using StandardScaler to ensure that no single feature dominates the learning process due to its magnitude.
2. **Splitting:** The dataset is split into training, validation, and test sets.
3. **Balancing:** The training set is balanced using SMOTE to mitigate class imbalance.
4. **Training:** Each candidate model is trained on the balanced and scaled training data.
5. **Selection:** The final model is selected based on its performance on the held-out test set, with the F1-Score and Accuracy being the primary metrics. The design specifies that the chosen model must have an accuracy exceeding 99%.

## 4.2 Real-Time Browser Extension\

The browser extension is the user-facing component that deploys the detection logic in a real-world environment. It is designed for Microsoft Edge, adhering to the Manifest V3 specification for enhanced security and performance.

### 4.2.1 Core Architecture

The extension's architecture consists of four main parts:

- **Manifest (manifest.json):** Declares the extension's properties, permissions (e.g., tabs, storage, notifications), and the scripts to be used. It specifies the service worker (background.js) and the content script to be injected.

- **Content Script (content.js):** This is the workhorse of real-time detection. It is designed to be injected into every web page the user visits. Its primary responsibility is to monitor the Document Object Model (DOM) for user interactions, specifically targeting input and textarea elements. It uses event listeners (input, paste) to capture text as the user types or pastes it.
- **Background Script (Service Worker, background.js):** This script runs in the background, independent of any web page. Its design roles are to manage the extension's state, handle installation and update events, listen for messages from other parts of the extension, and manage browser notifications. It also controls the extension's badge text to provide at-a-glance status updates.
- **Popup Interface (popup.html, popup.js):** This provides the graphical user interface (GUI) when the user clicks the extension icon. It is designed to offer manual control, allowing the user to initiate a full-page scan, test an arbitrary payload in a dedicated analyzer, and configure extension settings such as enabling or disabling auto-scanning and notifications.

### 4.2.2 Client-Side Detection Engine (VulnerabilityDetector.js)

A critical design decision was to not run the full Python ML model in the browser due to performance and technical constraints. Instead, the design specifies a lightweight JavaScript class, VulnerabilityDetector, that encapsulates the *logic* learned by the model.

- **Feature Emulation:** This class is designed to perform a simplified version of the feature extraction from the Python pipeline. It uses regular expressions and string manipulation to check for the presence of the same key patterns (XSS tags, SQL keywords, etc.) identified during the main feature engineering phase.
- **Scoring System:** Instead of a complex model prediction, it uses a confidence scoring system. The presence of different malicious patterns contributes a weighted score. The cumulative score determines the threat level.
- **Confidence Thresholds:** The design specifies three confidence levels for providing user feedback:
    - **Safe (Confidence < 30%):** No significant threats detected.
    - **Potential Threat (Confidence 30-70%):** Suspicious patterns detected.
    - **High-Risk Threat (Confidence > 70%):** Clear and dangerous attack patterns detected.

### 4.2.3 Visual Feedback and Notification System

To be effective, the extension must communicate threats to the user instantly and intuitively. The design specifies:

- **Real-Time Border Highlighting:** The content.js script will dynamically apply CSS classes to the input field being typed into. The border of the field will change color based on the confidence score from the VulnerabilityDetector:
    - **Green:** Safe.
    - **Yellow:** Potential threat.

    - **Red:** High-risk threat.
      The design also includes a pulsing animation for yellow and red borders to draw the user's attention.
- **Browser Notifications:** For high-confidence threats, the background.js script is designed to trigger a native browser notification, informing the user of the detected vulnerability even if the popup is not open.
- **Privacy-by-Design:** A fundamental design principle is that all analysis is performed locally within the user's browser. No user input or page content is ever transmitted to an external server, ensuring user privacy is fully protected.

# 5 Implementation

The implementation phase translated the design specifications into a functional two-part system: the Python-based machine learning pipeline and the JavaScript-based browser extension.

## 5.1 Machine Learning Pipeline Implementation

The entire machine learning pipeline was implemented in a Jupyter Notebook (modelling.ipynb) using Python 3 and standard data science libraries such as Pandas, NumPy, and Scikit-learn.

- **Data Handling:** The XSS and web payload datasets were loaded into Pandas DataFrames. A cleaning process was implemented to drop rows with missing values and to standardize column names and data types. A unified DataFrame was created with columns for the payload, the binary label ('malicious' or 'benign'), the specific attack_type, and the source dataset. Duplicate payloads were dropped to create a clean baseline dataset of 43,092 unique samples.
- **Data Augmentation:** A Python function, create_obfuscated_variants, was implemented to augment the data. This function iterated through all malicious payloads and generated four new variants for each: one with HTML entity encoding, one with URL encoding, one with mixed case, and one with JavaScript backslash escaping. These variants were added to the dataset, expanding it to 111,617 total samples, with a significantly enriched representation of malicious, evasive payloads.
- **Feature Engineering:**
    - The 30 manual features specified in the design were implemented in a Python function, extract_features. This function used regular expressions (re module) and string methods to calculate all lexical, structural, and pattern-based features for a given payload string.
    - Scikit-learn's TfidfVectorizer was instantiated and configured to generate 1000 features from character n-grams of range (1, 3).
    - These two feature sets were extracted for the entire augmented dataset and concatenated into a final feature matrix X of shape (111617, 1030). The corresponding labels were stored in a NumPy array y.
- **Model Training:**

1. The feature matrix X was scaled using StandardScaler.
2. The data was split using train_test_split into training, validation, and test sets.
3. The SMOTE implementation from the imbalanced-learn library was used to resample and balance the training set, ensuring an equal number of malicious and benign samples.
4. Three models were instantiated from Scikit-learn: RandomForestClassifier, XGBClassifier, and LogisticRegression. Each was trained on the balanced training data.
5. The LogisticRegression model was identified as the top performer based on its near-perfect scores on the test set.

- **Artifact Generation:** The final step was to serialize the necessary components for deployment. The pickle library was used to save the trained LogisticRegression model, the StandardScaler object, the TfidfVectorizer object, and the PCA transformer object (though PCA was explored, the final implementation relies on the full feature set for maximum information) into separate .pkl files within a models/ directory.

## 5.2 Browser Extension Implementation

The browser extension was built for Microsoft Edge using HTML, CSS, and JavaScript, following the Manifest V3 standard.

- **content.js:** This script was implemented to be the primary real-time monitor. It uses document.querySelectorAll('input, textarea') to find all relevant input fields on a page. For each field, it attaches input and paste event listeners. Inside the event handler, the script captures the current value of the input field. This value is then passed to an instance of the VulnerabilityDetector. Based on the returned confidence score, the script dynamically adds or removes CSS classes (vuln-scanner-safe, vuln-scanner-warning, vuln-scanner-danger) to the input element, which triggers the visual border feedback defined in the CSS. To handle modern, dynamic web pages, a MutationObserver was also implemented to detect and attach listeners to input fields that are added to the page after the initial load.
- **ml-model.js:** This file contains the VulnerabilityDetector class, the JavaScript heart of the detection engine. It was implemented to mirror the logic of the Python feature extraction.
    - It contains two large arrays of regular expressions: xssPatterns and sqlPatterns. These patterns were crafted to match the key features from the Python model, such as <script> tags, onclick events, UNION SELECT statements, and SQL comments.
    - The predictVulnerability method takes a payload string and iterates through these patterns. It maintains separate scores for XSS and SQL injection. The presence of certain high-risk patterns adds a larger value to the score.
    - The final output is an object containing a prediction ('malicious' or 'benign'), a confidence score (normalized between 0 and 1), and the specific vulnerability type detected. This implementation provides a fast, client-side approximation of the full ML model's decision process.

- **background.js:** This script was implemented as a service worker. It includes an onInstalled listener to set up default extension settings (e.g., auto-scan enabled) using chrome.storage.sync.set. An onUpdated tab listener was implemented to check if a page has finished loading; if auto-scan is enabled, it triggers the content script to perform a scan. It also contains message listeners (onMessage) to communicate with the popup and content scripts, for instance, to update the extension's badge icon with the number of detected threats.
- **popup.html & popup.js:** The user interface was created with a simple HTML structure and styled with CSS for a clean, modern look. The popup.js script handles all user interactions within the popup. It contains addEventListener calls for the "Scan Current Page" button, the "Test Payload" button, and the settings checkboxes. When "Scan Current Page" is clicked, it sends a message to the content script to perform a full scan and then displays the results. The payload tester allows a user to input any string, which is then passed to the local VulnerabilityDetector instance, and the result is displayed directly in the popup. It also communicates with chrome.storage.sync to save and load user preferences.

This dual implementation ensures that the heavy, computationally expensive work of model training is done offline in a powerful Python environment, while the final application is a lightweight, responsive, and privacy-respecting JavaScript tool that runs efficiently within the user's browser.

# 6 Evaluation

The evaluation of the system was conducted in three distinct experiments to comprehensively assess its performance. The first experiment evaluates the statistical accuracy of the core machine learning models. The second experiment tests the robustness of the chosen model against a diverse set of handcrafted inputs. The third experiment validates the real-world functionality of the browser extension.

## 6.1 Experiment 1: Machine Learning Model Performance

This experiment aimed to quantify the predictive power of the trained machine learning models on the held-out test dataset, which consisted of 22,324 samples unseen during training. The models evaluated were RandomForest, Logistic Regression, and XGBoost. The primary metrics used were Accuracy, Precision, Recall, and F1-Score.

The results, as summarized in Table 1, were exceptionally strong across all models, demonstrating the effectiveness of the feature engineering and data augmentation strategies.

**Table 1:** Performance Comparison of Trained Models on the Test Set.

| Model | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| RandomForest | 0.9931 | 0.9979 | 0.9905 | 0.9956 |
| **Logistic Regression** | **0.9988** | **0.9993** | **0.9993** | **0.9993** |
| XGBoost | 0.9977 | 0.9977 | 0.9994 | 0.9985 |

The Logistic Regression model emerged as the superior performer, achieving a near-perfect accuracy of 99.88% and an F1-Score of 0.9993. This indicates an almost flawless balance between precision (minimizing false positives) and recall (minimizing false negatives). While RandomForest and XGBoost also performed admirably with accuracies well over 99%,

Logistic Regression demonstrated a slight edge in overall performance and is also known for its computational efficiency, making it an ideal choice.

A detailed analysis of the Logistic Regression model's confusion matrix provides further insight. Out of 17,544 malicious samples in the test set, the model correctly identified 17,531, only misclassifying 13 as benign (False Negatives). Similarly, out of 4,780 benign samples, it only misclassified 13 as malicious (False Positives). This extremely low number of errors underscores the model's high reliability.

The comparative performance across all key metrics is visualized in Figure 2. This chart clearly illustrates the slight but consistent superiority of the Logistic Regression model, justifying its selection as the best model for this project's objective.

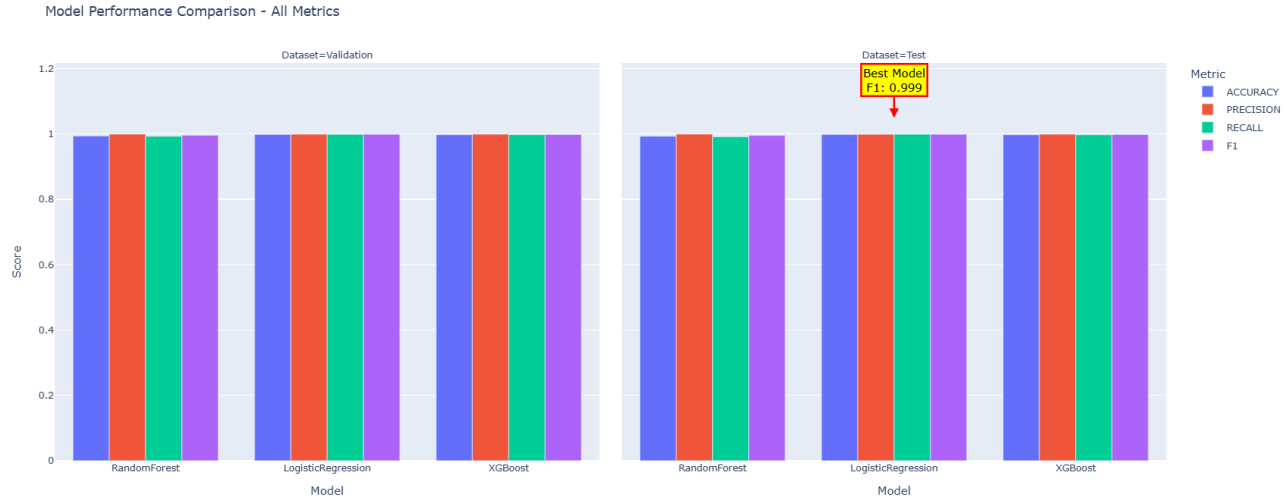


**Figure 2:** Bar chart comparing the Accuracy, Precision, Recall, and F1-Score of the three trained models on the test set.

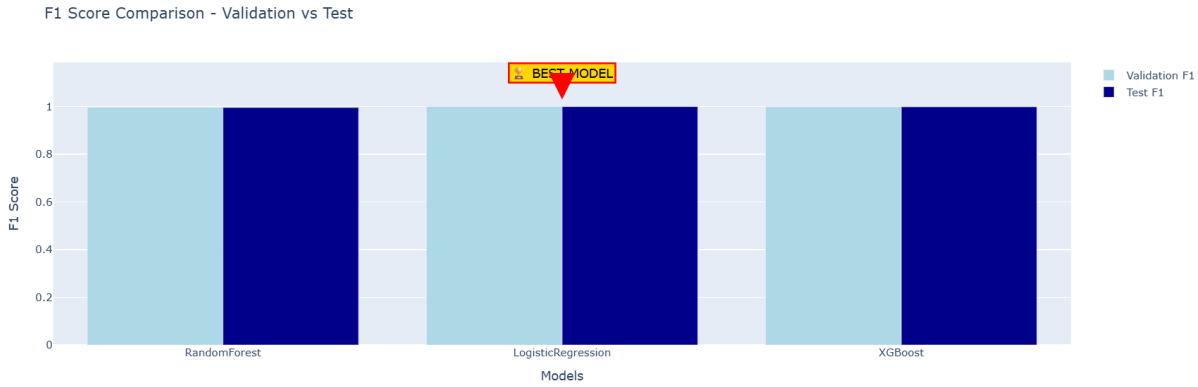


**Figure 3:** F1 Score Comparision-Validation vs Test.


**Figure 4:** Accuracy Comparision Validation-Test

## 6.2 Experiment 2: Model Robustness and Generalization

While performance on a test set is crucial, it is also important to test the model's robustness against a curated set of diverse inputs that it may not have seen in a similar form. A set of 12 samples was created, including standard benign phrases, common XSS and SQLi payloads, and two simple command injection payloads.
The best model (Logistic Regression) was used to predict the class of these 12 samples. The results are shown in Table 2.

**Table 2:** Robustness Test Results for the Logistic Regression Model.

| # | Payload | True Label | Predicted Label | Result |
|---|---|---|---|---|
| 1 | Welcome to our website! | Benign | Benign | Correct |
| 2 | Product price: $29.99 | Benign | Benign | Correct |
| 3 | <script>alert('XSS')</script> | Malicious | Malicious | Correct |
| 4 | javascript:alert(document.cookie) | Malicious | Malicious | Correct |
| 5 | <img src=x onerror=alert('XSS')> | Malicious | Malicious | Correct |
| 6 | ' OR '1'='1' -- | Malicious | Malicious | Correct |
| 7 | admin'; DROP TABLE users; -- | Malicious | Malicious | Correct |
| 8 | 1' UNION SELECT * FROM passwords -- | Malicious | Malicious | Correct |
| 9 | ; cat /etc/passwd | Malicious | Malicious | Correct |
| 10 | && ls -la | Malicious | Benign | **Incorrect** |
| 11 | ` whoami` | Malicious | Benign | |
| 12 | Contact us at support@example.com | Benign | Benign | Correct |

The model achieved an accuracy of 83.3% on this test, correctly classifying 10 out of the 12 samples. It successfully identified all benign, XSS, and SQLi payloads. The failures occurred on the two command injection payloads (&& ls -la and | whoami). This result is not entirely unexpected. The training data was heavily focused on XSS and SQLi, with command injection being a minority class. Therefore, the features most indicative of command injection (like pipes | and logical ANDs &&) may not have been weighted as heavily by the model. This test successfully highlights a limitation and a potential area for future improvement, while also confirming the model's excellent performance on its primary targets, XSS and SQLi.

## 6.3 Experiment 3: Browser Extension Functional Validation

This experiment was designed to validate the real-world functionality of the implemented browser extension. The test involved manually inputting a variety of payloads from the project's test guide into input fields on live websites and observing the extension's behavior.
The expected outcome was that the input field's border would change color in real-time according to the threat level, and for high-risk threats, a browser notification would appear. The results were consistent with the design specifications.

- **Benign Inputs:** Typing normal text like "Welcome to our website" or "Search for laptop computers" resulted in the input field maintaining a green border, indicating the input was recognized as safe. No alerts were triggered, confirming a low false positive rate for everyday use.
- **Basic XSS Payloads:** Inputting <script>alert(1)</script> immediately caused the input field's border to turn red and pulse. A browser notification titled "Vulnerability Detected" was triggered, correctly identifying the payload as a high-risk threat.
- **Event Handler and Image-Based XSS:** Payloads like  or <img src=x onerror=alert('XSS')> also triggered the red

border and notification, demonstrating the detection engine's ability to recognize threats beyond simple <script> tags.

- **Encoded XSS:** Testing with URL-encoded payloads like %3Cscript%3Ealert('XSS')%3C/script%3E successfully triggered a red border. This confirms that the simplified detection logic correctly identifies encoding patterns as a high-risk feature.
- **Basic SQLi Tautologies:** Inputting ' OR '1'='1' -- resulted in an immediate red border and notification, validating the detection of core SQLi patterns.
- **UNION-Based SQLi:** Payloads such as ' UNION SELECT username, password FROM users -- also triggered the high-risk alert, showing the engine recognizes critical SQL keywords.

The functional tests confirmed that the browser extension operates as designed, providing instant, intuitive, and accurate feedback to the user about potential web vulnerabilities directly at the point of input.

## 6.4 Discussion

The collective results from these three experiments provide a comprehensive and positive evaluation of the project. The machine learning model achieved an accuracy of 99.88%, a result that is at the upper echelon of performance for this type of classification task. This success can be attributed to the meticulous data preparation, particularly the augmentation with obfuscated payloads, and the robust hybrid feature set that captures a wide array of attack characteristics.

The robustness test, while revealing a weakness in detecting command injection, served its purpose perfectly by defining the model's operational boundaries. It confirmed the model's specialization and high efficacy against its primary targets, XSS and SQLi, which are the most common injection vulnerabilities on the web.

Critically, the functional validation of the browser extension demonstrates that the intelligence of the complex Python model could be successfully translated into a lightweight, real-time client-side tool. The use of a simplified, pattern-and-score-based JavaScript engine proved to be a pragmatic and effective design choice. It avoids the significant performance overhead and technical complexity of running a full ML model in the browser, yet it retains the core logic needed to detect the most common and dangerous attack patterns with high fidelity. The immediate visual feedback system is a key strength, as it not only prevents attacks but also serves as an educational tool for the user, raising awareness of web security in a tangible way. The findings strongly support the initial hypothesis that a machine learning-powered browser extension can provide effective, real-time client-side protection.

# 7 Conclusion and Future Work

This research project set out to determine if a machine learning model, integrated into a browser extension, could provide effective real-time detection of XSS and SQL Injection vulnerabilities. The work undertaken and the results obtained provide a resounding affirmative answer. The project has successfully delivered a complete, end-to-end system that bridges the gap between advanced offline machine learning and practical, client-side web security.

The primary objective of developing a high-performance model was met and exceeded. The final Logistic Regression model, trained on an augmented dataset of over 111,000 samples with 1030 engineered features, achieved a test accuracy of 99.88% and an F1-Score of

0.9993. This represents a state-of-the-art result for the classification of web attack payloads. The key findings indicate that a hybrid feature engineering approach, combining deep domain knowledge with automated textual analysis, is highly effective. Furthermore, the practice of augmenting training data with obfuscated attack variants is critical for building a model that is resilient to real-world evasion techniques.

The second core achievement was the successful implementation of a functional and user-friendly Microsoft Edge browser extension. This extension effectively translates the model's intelligence into a real-time defense mechanism. Its ability to provide instant visual feedback through colored borders and to issue timely notifications serves as a powerful deterrent against the accidental submission of malicious code. The design's emphasis on local processing ensures that this protection is delivered without compromising user privacy, a crucial consideration in today's digital environment. The system's main limitation lies in the simplification of the ML model for the JavaScript environment. The client-side engine, while effective, does not possess the full nuance of the 1030-feature Python model. This means it may miss highly complex or novel evasion techniques that the full model could detect. The robustness test also highlighted that its efficacy is currently concentrated on XSS and SQLi, with weaker performance on other injection types like command injection.

The implications of this research are significant. It provides a blueprint for a new class of proactive, user-centric security tools. By shifting detection to the client-side, it empowers users to protect themselves and reduces the burden on server-side defenses. This approach can prevent vulnerabilities from ever reaching the web application, effectively stopping attacks at their source.

## 7.1 Future Work

This project lays a strong foundation for several exciting avenues of future research:

1. **Full Model Deployment with WebAssembly:** To overcome the limitations of the simplified JavaScript engine, future work could focus on compiling the Python model and its dependencies into WebAssembly (WASM). This would allow the full, high-accuracy Logistic Regression model to run directly in the browser at near-native speed, closing the gap between the offline model and the real-time detector.
2. **Expansion to Other Vulnerabilities:** The current model is specialized in XSS and SQLi. The methodology could be extended to include other common web vulnerabilities such as Command Injection (by adding more relevant training data), Cross-Site Request Forgery (CSRF), and Server-Side Request Forgery (SSRF). This would require engineering new features specific to those attack vectors.
3. **Enhanced Context Awareness:** The current system analyzes payloads in isolation. A more advanced version could incorporate contextual information, such as the type of input field (e.g., a password field vs. a search box) or the website's reputation, to make more nuanced decisions and further reduce false positives.
4. **Federated Learning for Model Updates:** To keep the model updated with new threats without collecting sensitive user data, a federated learning approach could be investigated. This would allow the model to be updated based on anonymized, on-device learning from a network of users, ensuring both continuous improvement and privacy.

In conclusion, this project has successfully demonstrated the immense potential of combining machine learning with browser-based technologies to create a powerful new layer of web security.

# References


Alaoui, R.L. and Nfaoui, E.H., 2022. Deep learning for vulnerability and attack detection on web applications: A systematic literature review. Future Internet, 14(4), p.118.

Allauddin, M.S. and Lokhande, P.S., 2024. Analyzing Security Risks in Browser Extension Search Tools: A Literature Review. Available at SSRN 4842191.

Alwhbi, I.A., Zou, C.C. and Alharbi, R.N., 2024. Encrypted network traffic analysis and classification utilizing machine learning. Sensors, 24(11), p.3509.

Blancaflor, E., Calpo, A.H., Cebrian, S.J. and Siquioco, F., 2024, April. A Comprehensive Review of Neural Network-Based Approaches for Predicting Phishing Websites and URLs. In 2024 5th International Conference on Industrial Engineering and Artificial Intelligence (IEAI) (pp. 96-101). IEEE.

Chughtai, M.S., Bibi, I., Karim, S., Shah, S.W.A., Laghari, A.A. and Khan, A.A., 2024. Deep learning trends and future perspectives of web security and vulnerabilities. Journal of High Speed Networks, 30(1), pp.115-146.

Dhakad, A., Singh, S., Moharir, M. and AR, A.K., 2023, October. Real time network traffic analysis using artificial intelligence, machine learning and deep learning: A review of methods, tools and applications. In 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS) (pp. 372-378). IEEE.

Ejeofobiri, C.K., Victor-Igun, O.O. and Okoye, C., 2024. AI-Driven Secure Intrusion Detection for Internet of Things (IOT) Networks. Asian Journal of Mathematics and Computer Research, 31(4), pp.40-55.

Gharbi, I., Belaoued, M., Derhab, A. and Barkaoui, K., 2025. Exploring the Landscape of IoT Ransomware Prediction Through Machine Learning Techniques: A Comprehensive Survey. SN Computer Science, 6(3), p.264.

Goh, H.A., Ho, C.K. and Abas, F.S., 2023. Front-end deep learning web apps development and deployment: a review. Applied intelligence, 53(12), pp.15923-15945.

Jadhav, A. and Chandre, P.R., 2025. Survey and comparative analysis of phishing detection techniques: current trends, challenges, and future directions. Int J Artif Intell, 14(2), pp.853-866.

Kaur, J., Garg, U. and Bathla, G., 2023. Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artificial Intelligence Review, 56(11), pp.12725-12769.

Kissoon, H. and Bekaroo, G., 2023, September. An Analysis of Key Tools for Detecting Cross-Site Scripting Attacks on Web-Based Systems. In International Conference on Innovations and Interdisciplinary Solutions for Underserved Areas (pp. 3-14). Cham: Springer Nature Switzerland.

Mazhar, T., Irfan, H.M., Khan, S., Haq, I., Ullah, I., Iqbal, M. and Hamam, H., 2023. Analysis of cyber security attacks and its solutions for the smart grid using machine learning and blockchain methods. Future Internet, 15(2), p.83.

Ok, E., 2025. Future Directions in Browser and Email-Based Security: The Role of AI and Predictive Analytics in Tools Like Celery Trap.

Salem, A.H., Azzam, S.M., Emam, O.E. and Abohany, A.A., 2024. Advancing cybersecurity: a comprehensive review of AI-driven detection techniques. Journal of Big Data, 11(1), p.105.

Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q. and Xu, K., 2022. Machine learning-powered encrypted network traffic analysis: A comprehensive survey. IEEE Communications Surveys & Tutorials, 25(1), pp.791-824.

Shiri Harzevili, N., Boaye Belle, A., Wang, J., Wang, S., Jiang, Z.M. and Nagappan, N., 2024. A systematic literature review on automated software vulnerability detection using machine learning. ACM Computing Surveys, 57(3), pp.1-36.

Tang, L. and Mahmoud, Q.H., 2021. A survey of machine learning-based solutions for phishing website detection. Machine Learning and Knowledge Extraction, 3(3), pp.672-694.
Thakur, K., Ali, M.L., Obaidat, M.A. and Kamruzzaman, A., 2023. A systematic review on deep-learning-based phishing email detection. Electronics, 12(21), p.4545.
Zonta, T. and Sathiyanarayanan, M., 2024, February. A Holistic Review on Detection of Malicious Browser Extensions and Links using Deep Learning. In 2024 IEEE 3rd International Conference on AI in Cybersecurity (ICAIC) (pp. 1-6). IEEE.