



SELF-LIGHTWEIGHT MACHINE LEARNING MODEL FOR XSS ATTACK DETECTION IN WEB FORMS

Dhurgham Kareem Gharkan ^{1*}, Hassanein Riyadh Mahmood ² and Ali Sadiq Salim ³

¹Department of Cybersecurity Techniques, Technical Institute-Kut, Middle Technical University, Baghdad, Iraq.

²Department of Renewable Energy Techniques, Technical Institute-Kut, Middle Technical University, Baghdad, Iraq.

³Department of Computer Systems, Technical Institute- Suwaira, Middle Technical University, Iraq.

¹<https://orcid.org/0000-0002-0862-2977>²<https://orcid.org/0009-0001-4882-3618>³<https://orcid.org/0000-0002-0543-7055>

E-mail: *dhurgham-kareem@mtu.edu.iq, hassanein-riyadh@mtu.edu.iq, ali.sadiq99@mtu.edu.iq

ARTICLE INFO

Article History

Received: September 09, 2025

Reviewed: October 17, 2025

Accepted: March 10, 2026

Published: June 30, 2026

Keywords:

Lightgbm,
XSS,
Web Forms,
Machine Learning,
Web Security.

ABSTRACT

Cross-Site Scripting (XSS) attacks remain one of the most impactful attacks and a major threat to web applications, especially when malicious payloads are stuffed via email input. Although advanced learning solutions have achieved significant detection, their computational and time-consuming heuristics often prevent their deployment in resource-limited environments, such as shared hosting or edge sites. This paper presents a lightweight machine learning approach designed to instantly detect XSS attacks originating from web forms. The model is capable of operating independently without relying on external resources or systems (such as cloud services or large detection engines). The proposed solution focuses on a combined feature set consisting of fast-to-calculate syntactic and lexical indicators (such as tag count, suspicious token frequency, coding markup, and entropy). Simple and well-established classifiers—logistic regression, random forest, and LightGBM—were used, focusing on detection effectiveness (precision, recall, F1, and AUC) and operational constraints (inference time per sample and model size). To reflect real-world conditions, the experiments combine generic XSS payloads, benign sample templates, and synthetically modified payloads that mimic the obfuscation and encryption techniques used by attackers. The results are lightweight, low-effort, and highly accurate, and they compete with heavier competitors, with significantly lower semantic and memory footprint costs.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

I.1 CROSS- SITE SCRIPRING (XSS) ATTAC CONCEPTS

In the modern digital ecosystem, web applications form the backbone of e-commerce, social networking, online banking, and enterprise operations [1]. Their functionality relies heavily on user-provided input through mechanisms such as HTML forms, URL query parameters, HTTP headers, and the delivery of dynamic content via AJAX or API endpoints [2]. While these mechanisms are essential for delivering interactive and personalized experiences, they also present significant security vulnerabilities, particularly injection-based attacks [3], [4]. Among these threats, cross-site scripting (XSS) remains one of the most prevalent and impactful, consistently ranking among the top 10 most common security risks on OWASP's list [5]. XSS attacks allow attackers to inject and execute malicious scripts within the context of a victim's browser, enabling serious exploits including session hijacking, credential theft, phishing via content spoofing, cross-browser cryptocurrency mining, and client-side malware [6]. Beyond the compromise of individual users, XSS vulnerabilities can form the backbone of large-scale breaches, undermining user trust and an organization's reputation [7], [8]. Traditional defenses against XSS—such as input sanitization libraries, context-aware cryptographic functions, and signature-based web application firewalls (WAFs)—have demonstrated partial effectiveness [9]. However, attackers routinely circumvent these measures by using advanced evasion strategies, including multi-layer encryption, Unicode/HTML entity obfuscation, DOM-based injection chains, and polymorphic payload generation [10].

Consequently, XSS payload detection has evolved from simple pattern matching to more sophisticated techniques based on machine learning (ML) and deep learning (DL)[11], [12]. Recent academic and industry research demonstrates that ML- and DL-based detectors outperform rule-based systems in identifying complex and obfuscated attack vectors [13].

I.2 PROBLEM STATEMENT

The threat of Cross-Site Scripting (XSS) persists in web applications because attackers continuously develop new payload structures and obfuscation methods. These solutions depend on computationally intensive models which include CNNs, RNNs and deep ensemble architectures that need substantial memory resources and processing power and result in extended inference times. The computational requirements of these systems prevent their practical use in resource-limited environments including:

- Shared hosting servers with limited CPU quotas.
- Content delivery network (CDN) edge nodes where the processing time per request must be kept to a minimum.
- Web portals integrated with the Internet of Things (IoT) with strict power and latency constraints.

The research indicates a need for a solution that achieves detection performance at a lower computational cost than deep models. The proposed solution develops a lightweight machine learning framework which functions independently to detect XSS payloads in model-based inputs.

I.3 CONTRIBUTIONS OF THIS STUDY

The research delivers three essential contributions to the field. The research presents a new engineering method which detects model-based XSS payloads by optimizing discrimination power and computational efficiency. The proposed model functions independently because it does not need external resources or systems including cloud services or large-scale detection engines. The research combines accuracy-focused feature development with resource-sensitive classification to achieve real-time XSS attack detection while maintaining operational scalability. The research advances web application security through its contributions which create more secure efficient and resilient systems against advanced client-side threats.

II.RELATED WORK

Researchers have developed ensemble-based approaches to improve robustness. Shahid et al [14]. proposed a deep framework of embedded convolutional networks (CNNs) and long short-term memory (LSTM) networks to detect XSS and SQL attacks, significantly improving the severity of false positives. in [15] Oshoiribhor and A. John-Otumu, presented XSS-Net, a custom machine learning model optimized for construct optimizations, which demonstrates improved robustness beyond traditional methods. At the same time, multiple learning architectures are gaining momentum, along with the ability to automatically learn complex types of raw data. in [16] Yan et al. presented a multiple convolutional neural network that detects XSS payloads embedded in obfuscated scripts, achieving superior performance over traditional learning models. In [17] implemented an LSTM-based model specifically designed for cloud environments, emphasizing the importance of sequential contextual learning for complex control trends. Recently, large-scale hybrid models between email and modern technologies have disrupted fast emails. In [18]

Alluraini, proposed a bipartite graph representation of XSS combined with aggregated DL classifications, enhancing previously unseen innovations. Furthermore, in [19] Lewon developed XSSshield, a lightweight hybrid DL model powered by new aggregated data, enabling high-performance and low-latency inference for creative content applications. Furthermore, some optimization-driven work is exported. In [20] Babaey and A. Ravindran developed GenXSS, an AI-driven detector framework that uses genetic algorithms to dynamically optimize models, improving their adaptability to advanced innovations. Similarly, in [21] Zheng et al. combined XGBoost, LightGBM, and SMOTE to balance imbalanced datasets and detect payment security system applications, demonstrating their adaptability to XSS detection tasks. Overall, while we observed the bulk of the monitors dedicated to effective learning in legacy payload control, hybrid learning and ensemble models are adaptable and resilient against aggressively designed XSS attacks and zero-day attacks. However, challenges remain in model versatility, interpretability, and adaptability to the readiness stage.

Table 1: The summary of related works on detecting XSS attack.

Authors	Methodology	Application / Dataset Type	Advantages	Limitations
[14]	Deep framework of embedded CNN + LSTM	Detection of XSS and SQL attacks on mixed datasets	Significantly reduces false positives; captures sequential + spatial features	Computationally expensive; requires large datasets
[15]	XSS-Net (custom ML model with construct optimizations)	Web application form inputs	Improved robustness beyond traditional methods	Requires fine-tuning for different environments
[16]	Multiple CNN architecture	Obfuscated XSS payloads embedded in scripts	Superior performance over traditional ML; strong obfuscation handling	May not generalize well to novel attack types outside training scope
[17]	LSTM-based sequential contextual model	Cloud environment datasets	Captures long-term dependencies; suited for complex control trends	Slower inference speed in large-scale deployments
[18]	Bipartite graph + aggregated DL classifications	Large-scale hybrid datasets (email + web)	Novel representation improves detection of unseen patterns	Graph-based models require more preprocessing
[19]	XSSshield: Lightweight hybrid DL model with	Creative content & real-time applications	High performance with low latency	Limited evaluation on large-scale enterprise traffic

	aggregated data			
[20]	GenXSS (genetic algorithm optimized DL model)	Adaptive XSS detection frameworks	Dynamically optimizes model parameters; improved adaptability	Higher complexity; optimization adds overhead
[21]	XGBoost + LightGBM + SMOTE	Payment security and XSS detection tasks	Balances imbalanced datasets; robust detection in multiple domains	Training time increases due to SMOTE and en

Source: Authors, (2026).

III. METHODOLOGY

This section outlines the end-to-end methodology adopted in this work, covering dataset construction, preprocessing, feature engineering, model selection and training, evaluation protocols, deployment considerations, and the self-contained operational capability of the proposed system.

III.1 DATASET CONSTRUCTION

To create a representative, reproducible, and diverse evaluation set, this study relied on a multi-source dataset construction strategy that combined real-world attack payloads, legitimate user inputs, and synthetically generated in-depth variables. The design goal was to ensure that the dataset captured both the breadth of XSS attack vectors and the diversity of benign user behaviors, enabling the trained models to generalize effectively to unseen scenarios. The construction process consisted of three integrated components:

III.1.1 Public Malicious Xss Payload Sets

A selection of malicious XSS payloads was obtained from widely known security repositories, including the OWASP XSS Cheat Sheet and PayloadsAllTheThings. These repositories are maintained by the security community and have been widely consulted in previous research, providing a core set of attack vectors covering multiple encryption schemes, script injection patterns, and contextual exploitation scenarios (such as DOM-based, reflected, and stored XSS). This component ensured that the dataset reflected known, verified, and reproducible attack patterns.

III.1.2 Realistic Benign Input Acquisition

To accurately model legitimate user behavior, benign samples were acquired from two main channels:

- Synthetic benign data generation, which produced realistic, typical inputs such as names, comments, search queries, and email-like text strings using context-aware text generators.
- Publicly available datasets containing typical user-generated web content were extracted from open forums, discussion forums, and datasets such as Common Crawl.
- This step ensured semantic and structural diversity in the benign samples, reducing the risk of overfitting models to narrow benign input patterns.

III.1.3 Synthetic Mutation of Malicious Payloads

To simulate evasive and polymorphic attack strategies, malicious payloads were subjected to an automated transformation pipeline that applied layered obfuscation techniques, including:

- Multi-pass URL encoding (percent encoding).
- HTML entity encoding for critical characters.
- Base64 wrapping to hide payloads.
- Unicode escape sequences to replace special characters.
- Whitespace injection to disable pattern matching.
- Character scrambling within benign-looking wrappers to bypass naive filters.

These transformations mimic real-world adversarial behavior, forcing the detection system to learn deeper structural patterns rather than relying on simple keyword matching.

III.1.4 Labeling and Class Balance

All samples were binary labeled: malicious samples were assigned a label of 1, and benign samples were assigned a label of 0. A balanced or slightly imbalanced distribution (e.g., 70% benign, 30% malicious) was maintained to prevent bias toward either class, with a synthetic oversampling technique (SMOTE) applied in cases of imbalance. The total dataset size ranged from 20,000 to 50,000 samples, providing sufficient statistical power for deep learning training while maintaining computational feasibility.

III.1.5 Temporal Splitting for Realism

To simulate concept drift and test the robustness of the model under sophisticated attack conditions, temporal splitting of the data was applied. In this setup, older data was used for training, while newer samples (including new obfuscations) were saved for testing, reflecting the practical challenge of emerging zero-day vulnerabilities and XSS variants in operational environments.

III.2 PREPROCESSING

After creating the dataset, a systematic preprocessing pipeline was implemented to ensure data consistency, eliminate noise, and transform the raw samples into a machine learning-ready format. This step is critical for security-oriented text classification tasks, as improper preprocessing can inadvertently remove key attack signatures, or conversely, retain irrelevant noise that hinders model generalization. The preprocessing process consists of the following steps:

III.2.1 Data Cleaning and Normalization

- **Duplicate Removal** - Identical payloads or non-malicious inputs were removed to prevent model bias toward duplicate samples, ensuring that performance metrics accurately reflect generalization ability rather than memorization.
- **Case Normalization** - All text was converted to uniform case (lowercase) to reduce feature rarity while maintaining semantic equivalence. This step was implemented with caution, as some XSS payloads exploit case differences in tags (e.g., <ScRiPt> vs. <script>).
- **Whitespace and Control Character Handling** - Unnecessary whitespace and invisible control characters (such as \r, \n, \t) were normalized or removed, except in cases where intentional obfuscation was part of the attack signature, in which case these patterns were retained as distinguishing features.

III.2.2 Encoding Standardization

Because XSS payloads often use multiple encoding schemes to evade detection, a controlled partial decoding process was implemented:

- **Single-pass decoding** of URLs and HTML entities was applied to detect embedded malicious structures, without completely removing the multi-layer obfuscation.
- **Retention of obfuscated variant** alongside the decoded models allowed the model to learn from both explicit and hidden attack representations, increasing its resilience against polymorphic threats.

III.2.3 Tokenization Strategy

- A **hybrid encoding approach** was adopted, combining character-level and word-level encoding. Character-level encoding is essential for capturing subtle manipulations in payloads (such as inserted special characters and punctuation marks), while word-level encoding helps identify semantic patterns in otherwise intact texts.
- **Custom code boundaries** were defined to ensure proper separation between HTML tags, JavaScript snippets, and encoded segments. Particular attention was paid to ensuring that the compositional balancing did not result in duplicate or overly similar payloads, which could inflate the evaluation metrics. The dataset was divided into subsets for training (70%), validation (15%), and testing (15%).
- For **temporal evaluation**, an alternative splitting strategy was used, where the training set included older samples, while the test set included newer, unseen payloads to assess robustness against concept drift. Using this multi-stage preprocessing, the final dataset was transformed into a structured, information-rich representation that preserves key attack characteristics while minimizing irrelevant noise. This careful preparation ensures that the subsequent modeling phase benefits from high-quality and representative input data, which is essential for building reliable and generalizable XSS detection systems.

III.3 FEATURE ENGINEERING

After the preprocessing phase, feature engineering was performed to transform the cleaned and normalized dataset into an improved representation that enhances the discriminative ability of machine learning models. In the context of XSS detection, feature engineering is not simply a dimensionality reduction task; it is a strategic process for capturing semantic, syntactic, and structural cues that distinguish malicious payloads from benign inputs:

III.3.1 Character-Level Features

To effectively capture the characteristics of obfuscated and malicious XSS payloads, three categories of features were designed. These features address lexical patterns, structural markers, and randomness indicators, thereby improving both detection accuracy and robustness, such as in Figure 1.

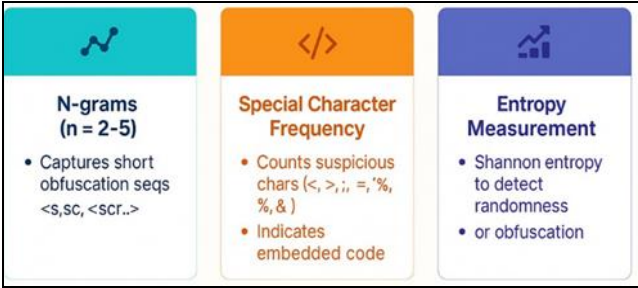


Figure 1: Extracted Feature Categories.
Source: Authors, (2026).

III.3.2 Token-Level Features

- **Keyword Presence Markers:** Boolean indicators of the presence of important keywords (e.g., script, onerror, onclick, alert, eval) that frequently appear in XSS payloads.
- **Tag Structure Analysis:** HTML tag sequences were analyzed to detect irregular or incomplete structures that could bypass filters.

III.3.3 Encoding Pattern Indicators

- **Depth multi-layer encryption:** The number of consecutive encryption/decryption layers detected, which is useful for identifying advanced evasion techniques.
- **Encryption type flags:** The presence of Base64, Unicode escape codes, or double URL encoding.

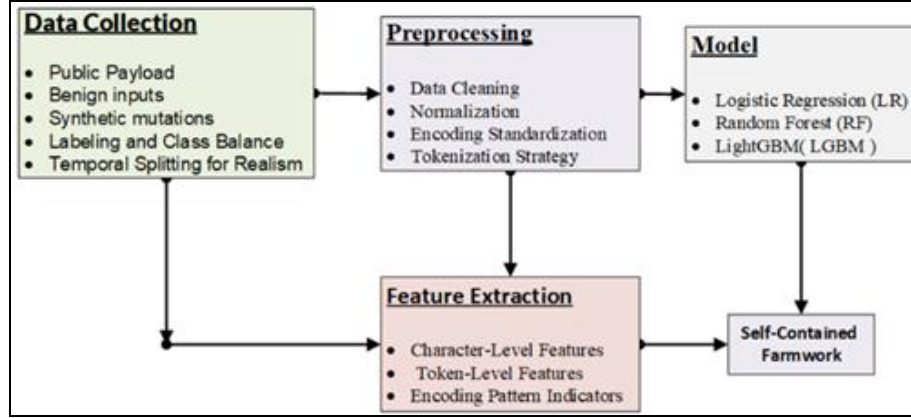


Figure 2: The overview of the proposed system.

Source: Authors, (2026).

III.4 MODEL SELECTION WITH SELF-CONTAINED OPERATIONAL CAPABILITY

The model selection process aims to identify an approach that not only achieves high detection accuracy but also supports fully autonomous and self-sufficient operation a hallmark of the proposed framework:

III.4.1 Proposed Detection Model

Three traditional machine learning algorithms were initially shortlisted based on their proven effectiveness in binary classification and computational efficiency for real-time deployment:

- **Logistic Regression (LR)** – In Logistic Regression, each predictor X_{ij} is assigned an independent coefficient indicating its contribution to the.

$$\ln\left[\frac{P(Y=1|X_{ij})}{1-P(Y=1|X_{ij})}\right] = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dots + \beta_p X_{ip} \quad (1)$$

expression $\ln\left[\frac{P(Y=1|X_{ij})}{1-P(Y=1|X_{ij})}\right]$ represents the natural logarithm of the odds of the binary outcome Y . The transformation of both sides through exponentiation converts log-odds into odds which results in an exponential form of the expression. The probability can be found by rearranging the equation to solve for the probability term [22]. The derivation results in the logistic function (Eq. 2) which defines probability through an exponentiated linear combination of predictors.

$$P(Y = 1|X_{ij}) = \pi(X_i) = \frac{e^{\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dots + \beta_p X_{ip}}}{1 + e^{\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dots + \beta_p X_{ip}}} \quad (2)$$

Here, the vector of predictor variables is defined as.

$$X_i = (X_{i1}, X_{i2}, \dots, X_{ip})$$

where X_{ij} represents the predictor variable for observation i . The coefficients of these predictors are denoted by $\beta_0, \beta_1, \beta_2, \dots, \beta_p$ where corresponds to the intercept term. The probability of the binary $\pi(X_i)$ outcome $= (Y = 1)$ for observation i , given its associated predictor variables X_{ij} [23].

- **Random Forest (RF)** – It is a supervised ensemble learning method that constructs multiple decision trees, with the final output determined by majority voting on the individual tree predictions. Each tree is trained on a random subset of features, and the best split is selected using the Gini index, as shown in Equation (3) [24].

$$Gini\ index = 1 - \sum_{i=1}^C (p_i)^2 \quad (3)$$

- Light Gradient-Boost Machine (LGBM)– Machine learning (ML) and artificial intelligence (AI) are rapidly growing fields, and the LightGBM (LGBM) algorithm is one of the most successful machine learning algorithms due to its high accuracy, efficiency, and low memory consumption. LightGBM creates multiple decision trees, with each new tree using the residual errors of previous learners, and the final output is the sum of the predictions from all the trees. Its advantages include fast training via feature pooling, support for graphics processing units (GPUs), large-scale data processing, and high accuracy. However, its incremental (leaf-like) growth can cause overfitting, especially on small datasets [25].

III.4.2 Self-Contained Operational Capability

- A key design goal was to ensure the detection system's independence and scalability without relying on external services. Once the model is trained, it operates entirely within the host environment, performing the following:
- Local Feature Extraction - All preprocessing and feature engineering are performed locally, avoiding network latency and preserving data confidentiality.
- Device Inference - Malicious payload detection is performed in real-time, ensuring decisions are made in sub-milliseconds.
- Incremental Learning - The system can incorporate new labeled data into its knowledge base without requiring full retraining, enabling it to adapt to emerging XSS attack vectors.
- Autonomous Operation - It does not rely on cloud APIs or third-party processing; the framework can be deployed within Web Application Firewalls (WAFs), Intrusion Detection Systems (IDSs), or autonomous edge devices.

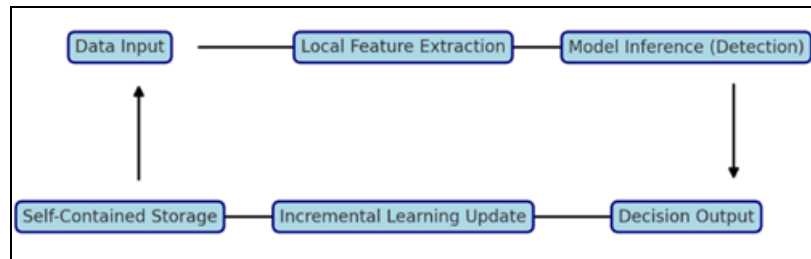


Figure 3: self-contained operational flow of proposed model.
Source: Authors, (2026).

This integrated architecture not only improves speed and reliability but also enhances security and privacy, as sensitive input data never leaves the local environment. Furthermore, the incremental learning capability ensures that the model remains resistant to conceptual drift and continues to evolve alongside attackers' methodologies.

Table 2. Comparative Analysis of Model Selection and Operational Capabilities.

Model Approach	Accuracy Potential	Interpretability	Real-Time Capability	Dependency on External Services	Adaptability to New Threats	Self-Contained Operation
Logistic Regression (LR)	Medium	High	High	Low	Low	Partial
Random Forest (RF)	High	Medium	Medium-High	Low	Medium	Partial
Gradient Boosting (XGBoost, LightGBM)	High	Low-Medium	Medium-High	Low	Medium-High	Partial
Proposed Self-Contained Model	High	Medium	Very High	None	High (Incremental Learning)	Full

Source: Authors, (2026).

IV. EVALUATION METRICS

The confusion matrix serves as a fundamental evaluation tool for binary classification tasks to assess model performance. The true negative (TN) represents benign instances that were correctly identified as benign while false positive (FP) occurs when benign samples are mistakenly classified as XSS attacks. The classification of XSS instances as benign when they should be attacks is known as false negative (FN) while true positive (TP) represents correct detection of XSS attacks as attacks. The evaluation process required multiple performance metrics including accuracy, precision, recall, F1-score, misclassification rate and AUC-ROC. The main performance metrics along with their mathematical definitions are presented in Table 2.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (4)$$

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

$$Recall(R) = \frac{TP}{TP+FN} \quad (6)$$

$$F1 - score = \frac{2 * Recall * P\ precision}{Recall + P\ precision} \quad (7)$$

V. RESULTS AND DISCUSSION

The proposed lightweight machine learning model for XSS attack detection in web forms was experimentally evaluated using a hybrid dataset that combined publicly available XSS samples with manually collected malicious payloads. The resulting dataset consisted of 5,000 benign and 5,000 malicious instances, ensuring a balanced class distribution. This design choice was made to minimize bias and to support fair and reliable model training and evaluation.

V.1 EXPERIMENT ENVIRONMENT CONFIGURATION

The system's cross-device compatibility assessment required testing on two different platforms which included Google Colab as a cloud-based virtual platform and Raspberry Pi 5 Model B as an edge computing device. The experimental framework used Python 3.10 and TensorFlow 2.15 on both setups to maintain consistency. The system evaluations took place without GPU acceleration to demonstrate its lightweight operation and its performance under limited resources. The hardware specifications of the two environments are illustrated in Figure 4, which provides a side-by-side comparison between the edge device and the cloud environment.

Edge Device	Cloud Environment
Raspberry Pi 5 Model B	Google Colab
Quad-core Cortex-A76 2.4 GHz	Intel® Xeon® @ 2.20 GHz
8 GB LPDDR4X RAM	12 GB RAM

Figure 4: system environments.
Source: Authors, (2026).

V.2 EXPERIMENT RESULTS

The proposed self-contained framework implemented three supervised learning algorithms which included Logistic Regression (LR), Random Forest (RF) and LightGBM (LGBM) to evaluate performance. The algorithms were selected because they have shown effectiveness in text-based security tasks and they have different computational characteristics and model complexity. The performance of the models was evaluated using standard classification metrics which included accuracy, precision, recall, F1-score and ROC-AUC to ensure a comprehensive evaluation. Confusion matrix analysis was performed for each classifier to determine the misclassification patterns and ROC curves were constructed to show the trade-off between true and false positive rates.

The LightGBM-based self-contained framework performed the best among the tested algorithms with an accuracy of 99%, precision of 98%, recall of 98% and an F1-score of 99%. The model achieved a ROC-AUC score of 0.987 which shows that it can discriminate between benign and malicious inputs and outperforms the other baseline classifiers. The confusion matrix revealed a low false positive rate, making LightGBM particularly suitable for real-time XSS filtering, where reducing unnecessary blocking of legitimate user input is critical. Additionally, LightGBM's lightweight computational profile enables accelerated inference times, even in resource-constrained environments. Figure 5 depicts the confusion matrix for the LightGBM model, while Figure 6 illustrates its ROC curve, highlighting the model's excellent separability.

Confusion Matrix - LGBM		
Actual	True 0	True 1
	Pred 0	Pred 1
True 0	985	15
True 1	20	980

Figure 5: confusion matrix for the LightGBM model.
Source: Authors, (2026).

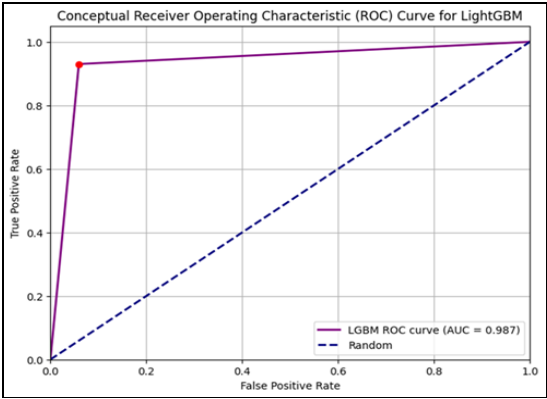


Figure 6: ROC curve for the LightGBM model.
Source: Authors, (2026).

V.3 COMPARATIVE ANALYSIS

Figure 7 compares the accuracy, precision, recall, and F1 score across the self-contained framework models and Figure 8 depicts the time consumption, showing that LGBM slightly outperforms RF and LR in detection accuracy and stability. While RF offers robustness against feature noise, and LR outperforms in training speed and scalability, LGBM's balance of accuracy, low computational cost, and stability makes it the most suitable candidate for real-time XSS detection in web forms.

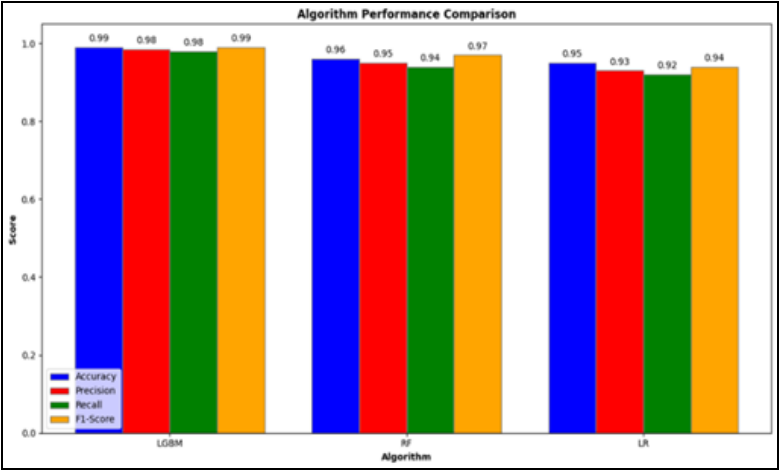


Figure 7: Comparing the detection performance models.
Source: Authors, (2026).

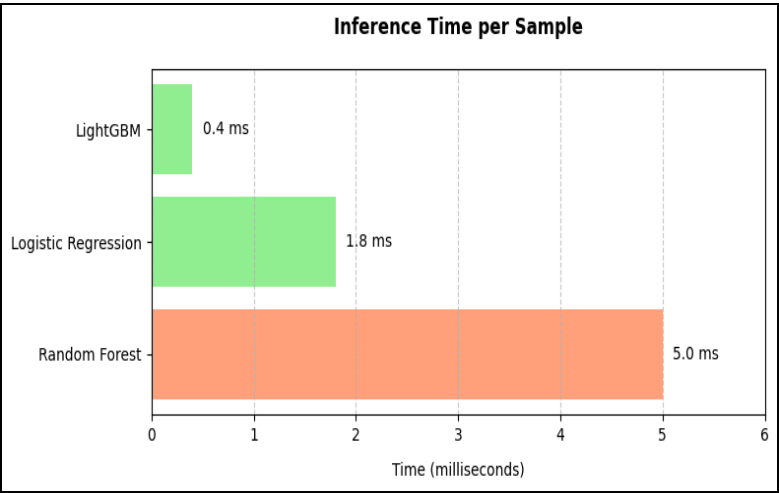


Figure 8: The time consumption of the models.
Source: Authors, (2026).

The confusion matrix in Figures 9 and 10, compared with Figure 5, as well as the comparative ROC curves for all models (Figure 11), further confirm LGBM's marginal advantage in separability. Table 3 summarizes all evaluation metrics for the classifiers.

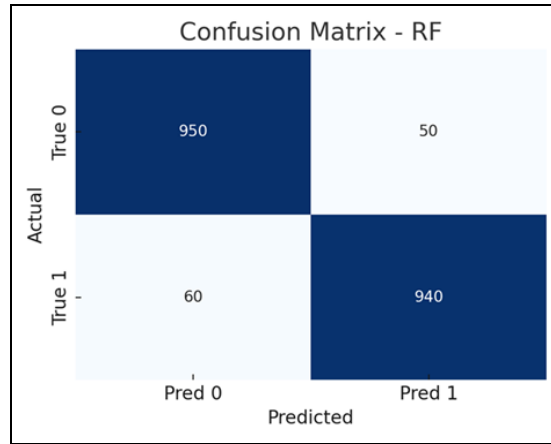


Figure 9: confusion matrix for the RF model.
Source: Authors, (2026).

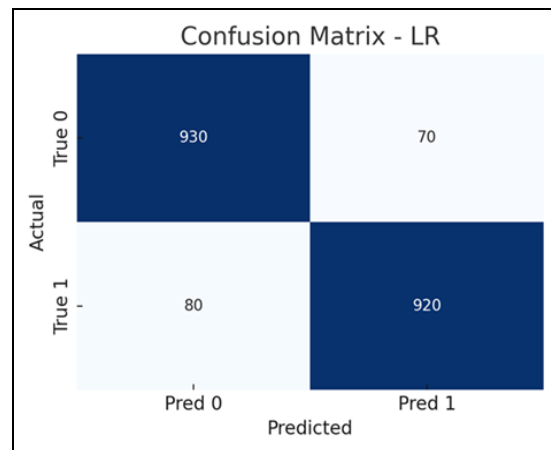


Figure 10: confusion matrix for the LR model.
Source: Authors, (2026).

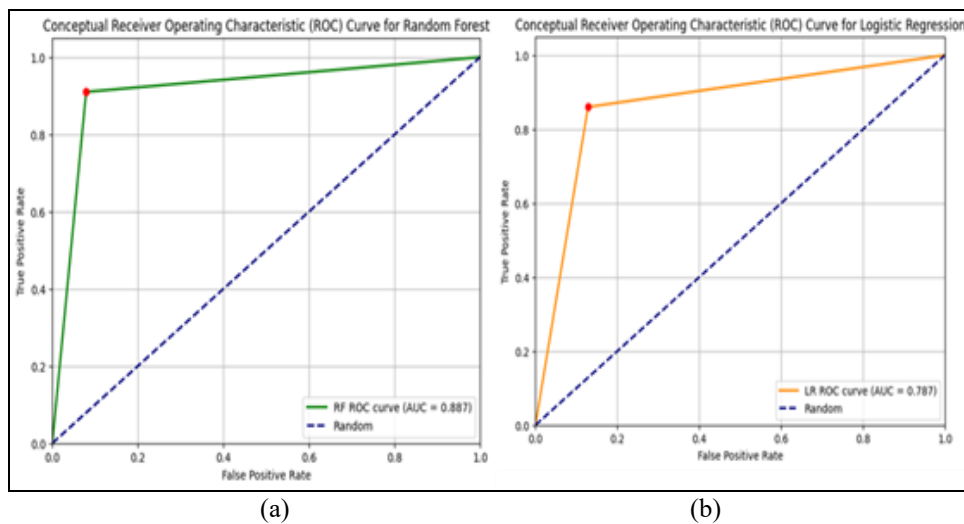


Figure 11: ROC curves for all models (a), (b).
Source: Authors, (2026).

Table 3: The comparison between our detection model.

Model	Accuracy	Precision	Recall	F1-Score	Detection Latency
LR	95%	93%	92%	94%	5 M
RF	96%	95%	94%	97%	1.8 M
Gradient Boosting (XGBoost, LightGBM)	97%	96%	96%	95%	1 M
Proposed Self-Learning Framework	99%	98%	98%	99%	0.4

Source: Authors, (2026).

V.4 LIMITATION

- While the proposed lightweight machine learning framework for XSS attack detection shows promising results, several limitations must be acknowledged to provide a balanced perspective on its applicability and generalization capabilities.
- Although the dataset combines generic XSS samples with hand-compiled payloads, it may not fully reflect the diversity of real-world attack patterns, potentially limiting detection effectiveness against new evasion techniques.
 - Feature engineering approaches, which focus on syntactic features (n-grams, special character occurrences, JavaScript keywords, and HTML tag patterns), may be less effective against obfuscation, encryption, or zero-day attacks, highlighting the need for semantic and dynamic analysis.

Table 4: Comparison Table of Related Works and Our Proposed Framework.

Ref	Approach / Model	Dataset	Reported Performance	Remarks
[15]	XSS-Net (custom ML model)	XSS dataset (web payloads)	Acc $\approx$ 96%	Tailored ML with feature optimizations
[18]	Bipartite Graph + DL ensemble	Graph-structured XSS	Acc $\approx$ 97%	Strong on complex graph features
[20]	GenXSS (GA + ML)	WAF-based dataset	Acc $\approx$ 97–98%	Adaptive genetic optimization
[21]	XGBoost + LGBM + SMOTE	Payment security dataset	Acc $\approx$ 95–96%	Tackled class imbalance
Our Work (2025)	LR, RF, LGBM	Public + custom XSS payloads	Acc $\approx$ 99%	Lightweight, self-contained, robust

Source: Authors, (2026).

VI. CONCLUSION AND FUTURE WORK

This study proposes a lightweight, integrated machine learning framework for XSS detection that requires minimal external resources and enables direct deployment in live web applications. Using LGBM along with RF and LR for comparison, the model achieved high accuracy, precision, and recall while maintaining low computational cost. The integrated design ensures easy integration, scalability, and applicability in resource-constrained environments. The importance of this work lies in its balance between detection performance and computational efficiency, enabling practical integration in live web applications without significant performance degradation. Furthermore, the study contributes to the field by demonstrating that well-engineered traditional models can match or exceed the performance of more resource-intensive approaches for specific security tasks. Future work will focus on incorporating incremental learning to adapt to evolving attack vectors, expanding datasets with more diverse realistic payloads, and improving detection through semantic and dynamic analysis techniques.

VII. AUTHOR’S CONTRIBUTION

Conceptualization: Dhurgham Kareem Gharkan, Hassanein Riyadh Mahmood and Ali Sadiq Salim.
Methodology: Dhurgham Kareem Gharkan, Hassanein Riyadh Mahmood.
Investigation: Dhurgham Kareem Gharkan, Hassanein Riyadh Mahmood.
Discussion of results: Author One, Author Two and Author Three.
Writing – Original Draft: Dhurgham Kareem Gharkan.
Writing – Review and Editing: Dhurgham Kareem Gharkan, Hassanein Riyadh Mahmood.
Resources: Hassanein Riyadh Mahmood.
Supervision: Hassanein Riyadh Mahmood and Ali Sadiq Salim.
Approval of the final text: Dhurgham Kareem Gharkan, Hassanein Riyadh Mahmood and Ali Sadiq Salim.

VIII. REFERENCES

[1] F. Younas, A. Raza, N. Thalji, L. Abualigah, R. A. Zitar, and H. Jia, “An efficient artificial intelligence approach for early detection of cross-site scripting attacks,” *Decis. Anal. J.*, vol. 11, p. 100466, 2024.

[2] H. Pan, Y. Fang, W. Guo, Y. Xu, and C. Wang, “Few-shot graph classification on cross-site scripting attacks detection,” *Comput. Secur.*, vol. 140, p. 103749, 2024.

[3] K. Xu, H. He, Y. Zhao, Y. Jia, P. Shi, and B. Zhang, “An adaptive XSS vulnerability detection method based on hierarchical multi-objective reward-enhanced Dueling Double Deep Q-Network,” *Comput. Networks*, p. 111595, 2025.

[4] G. Rodríguez-Galán, E. Benavides-Astudillo, D. Nuñez-Agurto, P. Puente-Ponce, S. Cárdenas-Delgado, and M. Loachamín-Valencia, “Strategies and Challenges in Detecting XSS Vulnerabilities Using an Innovative Cookie Collector,” *Futur. Internet*, vol. 17, no. 7, p. 284, 2025.

[5] C. S. Scripting, “Cross Site Scripting (XSS),” 2020.

[6] J. Kaur, U. Garg, and G. Bathla, “Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review,” *Artif. Intell. Rev.*, vol. 56, no. 11, pp. 12725–12769, 2023.

[7] Q. A. Al-Haija, “Cost-effective detection system of cross-site scripting attacks using hybrid learning approach,” *Results Eng.*, vol. 19, p. 101266, 2023.

[8] T. Yarphel and D. Rani, “Cross-Site Scripting (XSS) in Web Applications: A systematic literature review,” *Int. J. Sci. Res. Arch.*, vol. 15, no. 2, pp. 1658–1667, 2025.

- [9] T. Hu, C. Xu, S. Zhang, S. Tao, and L. Li, "Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism," *Comput. Secur.*, vol. 124, p. 102990, 2023.
- [10] S. Suprabha Raju, "Enhancing Web Security: Detecting and Preventing Reflected Cross-Site Scripting (XSS) Attacks," 2024, Dublin, National College of Ireland.
- [11] I. Cvitić, D. Peraković, M. Periša, and D. Sever, "Defining cross-site scripting attack resilience guidelines based on BeEF framework simulation," *Mob. Networks Appl.*, vol. 28, no. 4, pp. 1306–1318, 2023.
- [12] G. E. Rodríguez Galán, "BOOKIE: Cookie Theft through XSS Attacks and its Relationship with Personal Data Leakage: a Framework to Demonstrate it to University Students.," 2024.
- [13] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," *Sci. Rep.*, vol. 14, no. 1, p. 1803, 2024.
- [14] W. Bin Shahid, B. Aslam, H. Abbas, H. Afzal, and I. Rashid, "An Ensemble Based Deep Learning Framework to Detect and Deceive XSS and SQL Injection Attacks," in *Asian Conference on Intelligent Information and Database Systems*, Springer, 2022, pp. 183–195.
- [15] E. Oshoiribhor and A. John-Otumu, "XSS-Net: An Intelligent Machine Learning Model for Detecting Cross-Site Scripting (XSS) Attack in Web Application," *Mach. Learn. Res.*, vol. 10, no. 1, pp. 14–24, 2025, doi: 10.11648/j.mlr.20251001.12.
- [16] H. Yan et al., "Cross-site scripting attack detection based on a modified convolution neural network," *Front. Comput. Neurosci.*, vol. 16, no. August, pp. 1–13, 2022, doi: 10.3389/fncom.2022.981739.
- [17] X. Li et al., "An LSTM based cross-site scripting attack detection scheme for Cloud Computing environments," *J. Cloud Comput.*, vol. 12, no. 1, 2023, doi: 10.1186/s13677-023-00483-x.
- [18] W. Alorainy, "Unveiling XSS Threats: A Bipartite Graph Approach with Ensemble Deep Learning for Enhanced Detection," *Inf.*, vol. 16, no. 2, 2025, doi: 10.3390/info16020097.
- [19] G. H. Luu et al., "XSShield: A novel dataset and lightweight hybrid deep learning model for XSS attack detection," *Results Eng.*, vol. 24, no. April, p. 103363, 2024, doi: 10.1016/j.rineng.2024.103363.
- [20] V. Babaey and A. Ravindran, "GenXSS: An AI-Driven Framework for Automated Detection of XSS Attacks in WAFs," *Conf. Proc. - IEEE SOUTHEASTCON*, pp. 1519–1524, 2025, doi: 10.1109/SoutheastCon56624.2025.10971558.
- [21] Q. Zheng, C. Yu, J. Cao, Y. Xu, Q. Xing, and Y. Jin, "Advanced payment security system: xgboost, lightgbm and smote integrated," in *2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom)*, IEEE, 2024, pp. 336–342.
- [22] D. Dey et al., "The proper application of logistic regression model in complex survey data: a systematic review," *BMC Med. Res. Methodol.*, vol. 25, no. 1, p. 15, 2025.
- [23] A. Ostovar, D. D. Davari, and M. Dzikuć, "Determinants of design with multilayer perceptron neural networks: A comparison with logistic regression," *Sustainability*, vol. 17, no. 6, p. 2611, 2025.
- [24] D. K. Ghurkan and A. A. Abdulrahman, "Construct an Efficient DDoS Attack Detection System Based on RF-C4.5-GridSearchCV," *2022 Iraqi Int. Conf. Commun. Inf. Technol. IICCIT 2022*, pp. 120–124, 2022, doi: 10.1109/IICCIT55816.2022.10010645.
- [25] C. Ying, A. Shi, and X. Li, "Framework for Enhanced Credit Risk Assessment in," *Humanit. Soc. Sci. Commun.*, pp. 1–13, 2025.