Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.054780

ARTICLE

Deploying Hybrid Ensemble Machine Learning Techniques for Effective Cross-Site Scripting (XSS) Attack Detection

Noor Ullah Bacha[1], Songfeng Lu[1], Attiq Ur Rehman[1], Muhammad Idrees[2], Yazeed Yasin Ghadi[3] and Tahani Jaser Alahmadi[4,*]

[1]School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430073, China

[2]Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, 54000, Pakistan

[3]Department of Computer Science and Software Engineering, Al Ain University, Al Ain, 12555, Abu Dhabi

[4]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, 84428, Saudi Arabia

*Corresponding Author: Tahani Jaser Alahmadi. Email: tjalahmadi@pnu.edu.sa

Received: 07 June 2024 Accepted: 14 August 2024 Published: 15 October 2024

ABSTRACT

Cross-Site Scripting (XSS) remains a significant threat to web application security, exploiting vulnerabilities to hijack user sessions and steal sensitive data. Traditional detection methods often fail to keep pace with the evolving sophistication of cyber threats. This paper introduces a novel hybrid ensemble learning framework that leverages a combination of advanced machine learning algorithms—Logistic Regression (LR), Support Vector Machines (SVM), eXtreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), and Deep Neural Networks (DNN). Utilizing the XSS-Attacks-2021 dataset, which comprises 460 instances across various real-world traffic-related scenarios, this framework significantly enhances XSS attack detection. Our approach, which includes rigorous feature engineering and model tuning, not only optimizes accuracy but also effectively minimizes false positives (FP) (0.13%) and false negatives (FN) (0.19%). This comprehensive methodology has been rigorously validated, achieving an unprecedented accuracy of 99.87%. The proposed system is scalable and efficient, capable of adapting to the increasing number of web applications and user demands without a decline in performance. It demonstrates exceptional real-time capabilities, with the ability to detect XSS attacks dynamically, maintaining high accuracy and low latency even under significant loads. Furthermore, despite the computational complexity introduced by the hybrid ensemble approach, strategic use of parallel processing and algorithm tuning ensures that the system remains scalable and performs robustly in real-time applications. Designed for easy integration with existing web security systems, our framework supports adaptable Application Programming Interfaces (APIs) and a modular design, facilitating seamless augmentation of current defenses. This innovation represents a significant advancement in cybersecurity, offering a scalable and effective solution for securing modern web applications against evolving threats.

KEYWORDS

Cross-site scripting; machine learning; XSS detection; stacking ensemble learning; hybrid learning

1 Introduction

The integration of web technologies into daily operations has transformed how businesses and governments deliver services, enhancing accessibility and operational efficiency. As the digital landscape evolves, web applications have become fundamental in facilitating a broad range of services—from financial transactions and e-commerce to healthcare and government services. These benefits come with hazards, though, since web apps hold user data that is frequently the target of cyberattacks, exposing sensitive information that belongs to the company and its users to possible dangers. Cyber attackers now have an easier time breaching online systems and obtaining sensitive user data thanks to the widespread availability of online services like social networking sites, online payments, online shopping, electronic banking, medical services, railroad booking, airline booking, and many more [1]. The general population is completely unaware that hackers might steal their personal information; therefore, protecting the privacy of information found on websites and applications, such as cookies and session tokens, is a basic user right that helps protect sensitive data from unauthorized access. The only way to safeguard user information and privacy in web systems is to implement strong threat detection algorithms.

Cross-Site Scripting (XSS) attacks are among the most prevalent cybersecurity threats, exploiting vulnerabilities within web applications to execute malicious scripts on user browsers. These attacks manipulate web applications to inject harmful scripts, which then run on the client side, potentially leading to unauthorized access to personal data and a breach of user privacy [2]. Because of their complexity, the constant introduction of new technologies, and the integration of back-end and front-end development processes, web applications are subject to new vulnerabilities that arise daily [3]. The goal of exploiting insecure and malicious programs is to gain access to a website and render it inoperable. For some applications, particularly those used in high-availability operations or priority services like banking, e-commerce, healthcare, etc., this poses the biggest security risk [4]. For instance, web applications make heavy use of the logging functionality [5]. Insufficient handling of the logging functionality can result in risky security risks, such as the ability for an attacker to initiate the execution of malicious commands by inserting them into the web application logs and then gaining access to the view-logs interface, which creates a vulnerability for web application log injection [6].

The primary cause of XSS attacks is inadequate sanitization of user input. Attackers make use of this vulnerability by inserting malicious code into trustworthy websites that are susceptible to it to convince users to visit sites [7]. As a result, the malicious script is allowed to run on the victim's browser, giving the attacker access to manipulate and assault the victim's browser. Sensitive data, including login credentials, financial information, and usernames, can be exposed by XSS attacks. XSS is mostly malicious code that infects trusted websites' dynamically produced response webpages. The browser parses and runs the malicious script when the victim opens a web application that contains it. This gives the attacker access to the victim's browser and gives them the ability to steal, alter, or even elevate their virtual identity, system data, and other data. The primary issue with cross-site scripting (XSS) is that flaws that allow these kinds of assaults to occur frequently occur whenever an online program incorporates user input without encoding or validating it in the output it produces. As a result, an attacker may transmit a malicious script to a user who isn't paying attention by using XSS [8]. The browser of the end user is unaware that the script is unreliable since it believes it originated from a reputable source. The script is typically run by the end user. So, any cookies, tokens, or other private data saved by the browser and used with that website can be accessed by the malicious script. The full functionality of a typical Cross-Site Scripting (XSS) attack is depicted below in Fig. 1.

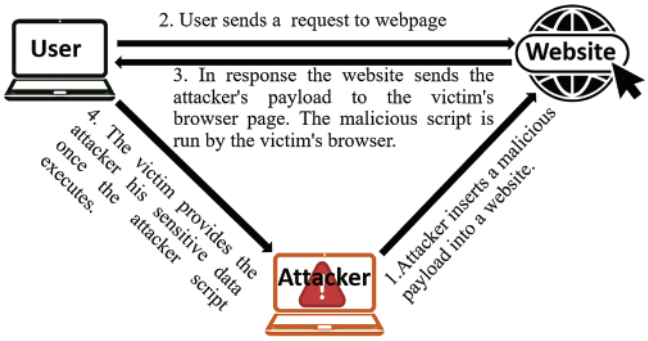

Figure 1: General illustration of the cross-site scripting (XSS) attack process

Recent studies have demonstrated that machine learning (ML) based techniques, such as support vector machine (SVM), Random Forest, Naïve Bayes, Bayes Net, Logistic Regression, etc., can enhance the detection of XSS attacks and get around problems that the traditional methods of XSS attack detection, such as input validation, static analysis, and dynamic analysis [9]. The primary benefit of machine learning (ML) techniques is their capacity to learn from data and adjust to novel or unidentified XSS attack forms due to their ability to recognize patterns in massive datasets and generate precise predictions for novel occurrences, several machine learning-based techniques have been put out to identify XSS assaults.

This work investigates various cyber-attack detection methods for Cross-Site Scripting XSS detection on webpages. The advantage of the machine learning approach over other approaches in terms of several performance indicators has led to its consideration. The stacking ensemble ML learning method has been primarily responsible for the outstanding performance.

Detection of cross-site scripting attacks using ML and new techniques in ML to improve the performance of detection techniques because of evolving XSS attacks. Though many machine learning methods have been investigated for identifying Cross-Site Scripting (XSS) attacks, thorough comparative studies that methodically assess the effectiveness of several machine learning algorithms—Logistic Regression (LR), Support Vector Machine (SVM), Categorical Boost (CatBoost), eXtreme Gradient Boost (XGBoost), and Deep Neural Network (DNN) and using Stacking Ensemble learning with these models in particular—in this situation are scarce, and their model performance was not adequate to overcome the detection of XSS attack. It is difficult to determine the relative efficacy and efficiency of different strategies for XSS detection because the literature currently in publication frequently concentrates on specific algorithms or small-scale comparisons. One reason is minimal comparative studies; prior work has mostly concentrated on single algorithms or small-scale comparisons between a small number of methods, offering little understanding of the advantages and disadvantages of various strategies. The selection of a related machine learning method also matters as the selection of a machine learning method has a substantial influence on the XSS detection systems' scalability, accuracy, and efficiency in comparison to more conventional approaches like signature-based detection. Nevertheless, there isn't much advice on which method or algorithms would be best for this kind of work. There are also optimization opportunities, by identifying areas for algorithmic optimization and feature engineering specific to XSS attacks, a thorough comparison investigation can result in more effective detection methods.

Thus, in this paper, we found how and in what ways machine learning may be used to enhance the detection of cross-site scripting attacks. The study intends to contribute to the creation of more

effective and efficient XSS detection methods in cybersecurity by offering insights into the advantages of these algorithms through empirical evaluation and comparative analysis. We established a series of targeted objectives to guide our investigation into the detection of XSS attacks using a hybrid ensemble machine learning approach. These objectives are crafted to address the specific challenges associated with XSS vulnerabilities, leveraging advanced algorithms and data processing techniques to enhance the accuracy and reliability of XSS detection systems. The delineated objectives not only structure our approach but also highlight our commitment to advancing the field of cybersecurity through innovative ML applications. Below are the detailed research objectives of our study:

Research Objective (RO) 1: To employ advanced machine learning techniques, including Logistic Regression (LR), Support Vector Machine (SVM), Categorical Boost (CatBoost), eXtreme Gradient Boost (XGBoost), and Deep Neural Network and using Stacking Ensemble learning with these models, to detect cross-site scripting (XSS) attacks effectively.

Research Objective (RO) 2: To analyze and utilize a comprehensively labeled dataset to enhance the effectiveness of XSS cyber-attack mitigation strategies through machine learning.

Research Objective (RO) 3: To apply data sampling and balancing techniques to ensure the robustness and accuracy of the machine learning models in detecting XSS attacks.

Research Objective (RO) 4: To refine and preprocess the datasets to optimize their utility for both training and testing the proposed XSS detection models.

Research Objective (RO) 5: To evaluate the performance of the proposed models using metrics such as accuracy, precision, recall, and F1-score to assess their effectiveness in detecting XSS attacks.

Research Objective (RO) 6: To implement rigorous validation procedures, including cross-validation, to verify the effectiveness and generalizability of the XSS detection methodologies.

These objectives serve as the backbone of our research, ensuring a thorough and methodical approach to addressing the complex challenges posed by XSS vulnerabilities in web applications.

This research has meticulously combined several advanced ML models to create a robust hybrid and ensemble approach tailored specifically for XSS detection. The following points elucidate the major contributions of our research, highlighting the innovative methods and their implications for cybersecurity. Our study's main research contributions are as follows:

Research Contribution (RC) 1: We propose a Hybrid and ensemble of advanced machine learning techniques including Logistic Regression, Support Vector Machines (SVM), XGBoost, CatBoost, and Deep Neural Networks (DNNs) for the effective detection of Cross-Site Scripting (XSS) attacks, which marks a significant enhancement over traditional single-model approaches.

Research Contribution (RC) 2: We utilize a comprehensive dataset, adapted for the nuances of XSS detection, to conduct a thorough analysis of the proposed models. This study is distinctive in its application of these specific advanced ML models to XSS detection, providing a novel insight into their relative effectiveness and efficiency.

Research Contribution (RC) 3: We employ cross-validation techniques to enhance the validation process, ensuring robust model performance and minimizing overfitting, which is critical for maintaining high reliability and generalizability of the XSS detection models across various data scenarios.

Research Contribution (RC) 4: Our research rigorously evaluates the models using multiple performance metrics, including accuracy, precision, recall, and F1-score. This multi-metric assessment helps in understanding the strengths and limitations of each model in real-world application scenarios.

Research Contribution (RC) 5: The findings contribute to the cybersecurity field by offering detailed benchmarks of model performance, thereby aiding cybersecurity professionals in selecting appropriate ML techniques for implementing effective XSS attack detection systems.

These contributions are pivotal in advancing the field of cybersecurity, particularly in the realm of XSS attack detection, providing both theoretical insights and practical tools to combat this ever-evolving threat.

Furthermore, in this paper, we have developed ML techniques to detect XSS. The key motivation is to analyze and improve the accuracy to be enough adequate for the proposed ML model. Different parts provide explanations of the cybersecurity XSS attacks, machine learning techniques used to detect XSS attacks, and our suggested model developments. The structure of this document is as follows. Section 1 defines an introduction. The related work on XSS attack detection is summed up in Section 2, Section 3 presents the methodology, and Section 4 presents our research analysis and findings. Finally, the work is concluded in Section 5, with the main conclusions drawn from our paper.

2 Related Work

This section gives a summary of earlier reviews that address the detection of XSS attacks. It also presents a comparison of this review with the most relevant research. In recent years, a growing number of researchers have used machine learning to increase the effectiveness of XSS attack detection due to the attractiveness of machine learning algorithms. There has been a lack of sufficient and appropriate approaches and solutions presented for minimizing, identifying, or avoiding such attacks. Furthermore, no single technique can completely fix flaws in the application's source code or stop XSS assaults from occurring. Several criteria are evaluated to categorize the Web application's protection method. These variables include, for instance, the kinds of attacks that the defense system stops or detects, as well as a few basic elements of the strategy (accuracy, precision, recall, etc.). The authors [10] offered both static and dynamic analytical techniques for spotting risky websites. A supervised decision tree technique has been used to achieve 95.2% precision, 91.6% F1-score, and 94.79% receiver operating characteristic curve (ROC) values for binary classification. However, this finding is insufficient to stop such attacks due to the low detection rate and precision. Because of this, there will be a significant false-negative rate and a possibility that the system will miss multiple attacks. Fig. 2 presents an organized overview of machine learning, categorizing various algorithms into four main types: Supervised learning, Unsupervised learning, Semi-Supervised learning, and Reinforcement learning.

In Supervised learning, the focus is on Classification and Regression, with algorithms like Linear Regression, Random Forest, Logistic Regression, Support Vector Machine (SVM), Decision Trees, Naïve Bayes, Gradient Boost, and Artificial Neural Networks. These algorithms are trained with labeled data to predict outcomes for new, unseen data.

Unsupervised learning deals with Clustering and Dimensionality Reduction. Algorithms like K-means Clustering, Spectral Clustering, K-medians Clustering, Hierarchical Clustering, Quadratic Discriminant Analysis, Linear Discriminant Analysis, Principal Component Analysis, and Multidimensional Scaling are used to find patterns or groupings in data without pre-existing labels.

Semi-supervised learning is positioned between supervised and unsupervised learning, utilizing both labeled and unlabeled data for training. This branch includes techniques like Generative Models, Self-training Algorithms, Graph Theory Methods, Low-Density Separation Models, Semi-Supervised Support Vector Machines, and Expectation Maximization.

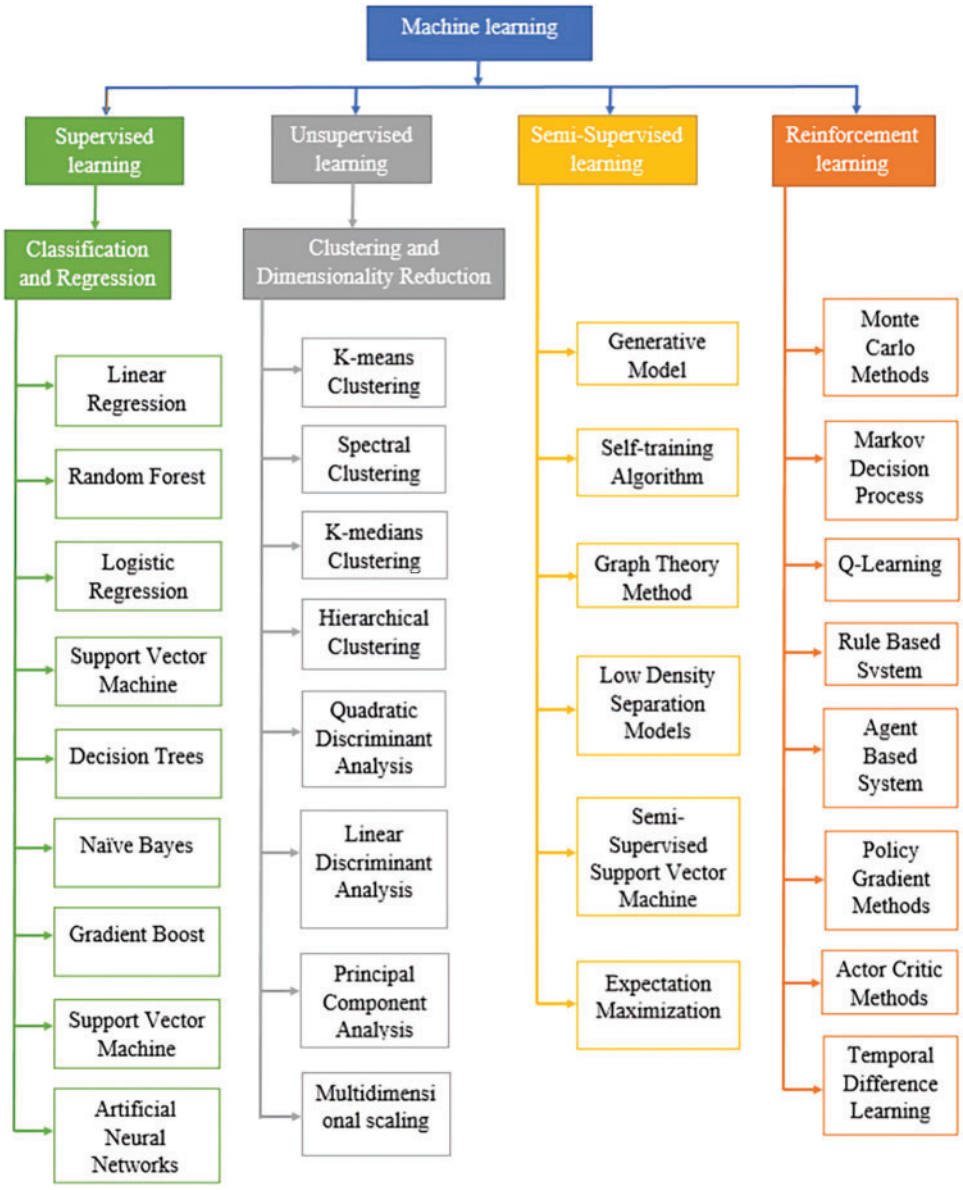

Figure 2: Comprehensive taxonomy of machine learning (ML) algorithms

Lastly, Reinforcement learning is an area focused on decision-making and motor control, often modeled as a Markov decision process. Methods under this category include Monte Carlo Methods, Q-learning, Rule-Based Systems, Agent-Based Systems, Policy Gradient Methods, Actor-Critic Methods, and Temporal Difference Learning, all aimed at learning strategies to maximize a notion of cumulative reward.

Overall, Fig. 2 effectively maps out the landscape of machine learning techniques, providing a clear guide to the different methods and their applications in data analysis and artificial intelligence. Researchers employ a variety of machine learning techniques to demonstrate effectiveness. The accuracy rate of the Support Vector Machine (SVM) classifier used for XSS attack detection is

92% [11]; however, the False Positive (FP) rate remains unknown. Their goal is to surpass the quantity of False Positive (FP) and False Negative (FN). Even yet, the predicted False Negative (FN) and False Positive (FP) values—7 and 5.7%, respectively—were still seen as excessive and inadequate. To solve the detection challenge, the study of [12] proposes detection approaches that use a Multilayer Perceptron (MLP), a large dataset, and feature extraction.

Furthermore, several machine learning-based algorithms have been developed to detect XSS attacks in web applications as a result of improvements in Artificial Intelligence (AI) technology. A Convolutional Neural Network (CNN) model is used in [13] for detection, which produced an accuracy of 98.2%, precision of 98.8%, and True Positive Rate (TPR) of 98.8%. Also, Reference [14] proposed a Machine Learning (ML) technique of Optimal Decision Tree (ODT) and produced an accuracy of 97.4%, precision of 97.2%, and recall/TPR of 95.6%. The research in [15] applied the machine learning technique of Decision Tree Classifier (DTC), which shows an accuracy of 98.20%, a precision of 99.19%, and a recall/TPR of 93.7%. Similarly, Li et al. [16] used the Bidirectional Long Short-Term Memory (Bi-LSTM) machine learning technique for the detection of XSS attacks, producing an accuracy of 92.37%, a precision of 92%, and a recall/TPR of 92%. An approach based on deep learning called DeepXSS [17] uses a long short-term memory (LSTM) recurrent neural network. It is trained using the collection of features that word2vec helped to extract. The method produced a False Positive Rate (FPR) of 0.019% and a precision of 99.5%. The research in [18] developed a method for generating XSS attack string instances by combining an improved version of the Monte Carlo Tree Search (MCTS) algorithm with a Generative Adversarial Network (GAN) classifier. They successfully detected XSS attacks with a TPR value of 94.59%. The research put forth a strategy based on genetic algorithms and reinforcement learning. It detects cross-site scripting attacks by utilizing threat intelligence. An accuracy of 99.8% was attained by the method; the FPR is unknown. The extreme gradient boosting method (XGB) with parameter optimization methodology is used in XGBXSS [19], an ensemble learning-based technique for XSS detection. The method produced results of 99.5% precision, 99.56% accuracy, and 99.02% recall with an FPR of 0.0020%. Though they can identify and mitigate XSS attacks, these methods may have significant drawbacks, including a high rate of false alarms, a high computational processing cost, and the inability to overcome partially injected attack payloads and encoded scripts.

Conversely, Kaur et al. [9] provided an overview of XSS web attacks and countermeasures. Their research highlights the persistent issue of cross-site scripting vulnerabilities in well-known websites and investigates several ways to lessen these risks. Furthermore, Krishnan et al. [20] provided a thorough categorization of XSS attack tactics and associated mitigation methods. Their paper offers a thorough summary of the most recent developments in XSS attack detection and prevention, providing insightful information for both academics and industry professionals. Additionally, a rapidly expanding field of study is the application of AI and machine learning techniques to cybersecurity.

All things considered, these investigations highlight the XSS assaults' dynamic nature and the consequent demand for creative and practical detection and mitigation techniques. It appears that integrating AI and ML with cybersecurity is a worthwhile field of study, with the potential to make significant progress against these ubiquitous cyber threats. As in the study of [21] splits, the datasets with no split 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, and 45% records in the testing dataset, the effectiveness of various classifiers is then evaluated using tenfold cross-validation. They provide a technique for detecting cross-site scripting (XSS) attacks that are based on ensemble learning and use Bayesian networks (BNs) as individual classifiers. They achieved an accuracy of 96.96%, 97.59%, 98.06%, 97.89%, 97.64%, 97.78%, 97.63%, 97.88%, 98.22%, and 98.54%, respectively.

In terms of detecting cross-site scripting (XSS) attacks, machine learning has demonstrated strong performance. Nevertheless, dynamic context information [22] is skipped throughout the model training phase, which leads to false negatives (FN) of XSS assaults. PhishMon, a dynamic system for phishing assault detection based on machine learning, was introduced by [23] in their research. It took advantage of fifteen unique features that were taken straight from the webpage and are hard for hackers to duplicate. It did not compute its phishing-detecting features by using any search engine or third-party system or services. They achieved an accuracy of 95.40%. The study in [24] proposed a method for using aggregation analysis to determine how similar a page's layout is. Using Cascading Style Sheets (CSS) layout attributes, the author trains the classifier to identify and locate comparable websites, allowing them to classify a page as real or fraudulent. The method is intended to automatically produce rules based on how similar website pages' layouts are to one another. Their accuracy was limited to 97.31%.

To improve anti-phishing techniques, Karim et al. [25] suggested a hybrid classifier-based model that incorporates both firm and flexible voting. In the suggested study, a canopy technique is also used for feature selection in conjunction with a grid search optimization method. In contrast, they obtained an F1-score of 95.89% and an accuracy of 98.12%. Similar to the study by [26], they suggested a feature-rich, machine learning-based anti-phishing detection method. With an F1-score of 98.2%, their feature-based results demonstrate a high accuracy of 97.8%. Nonetheless, the current research divides the dataset using random sampling and ignores the variety of XSS load kinds. These approaches will encounter overfitting issues for XSS load types with few labeled samples. To address these problems, this work adds small samples to the body of research on XSS attack detection. Below Table 1 is detailed explanation of past state-of-the-art XSS attack detection techniques.

Table 1: Past state-of-the-art XSS attack detection

No.	Authors	Methods	Aims	Dataset source	Accuracy	Pros	Cons
1.	Chaudhary et al. [27] (2022)	Long short-term memory (LSTM)	XSS attack detection and mitigation	GitHub	98%	Their approach is to detect and mitigate the XSS attack.	There is a need to evaluate some ensemble methods to improve performance.
2.	Zhou et al. [21] (2019)	Ensemble learning and Bayesian networks (BNs)	XSS attack detection	GitHub	98.5%	Their approach is to detect the XSS attack.	There is a need to evaluate some more effective ensemble methods to improve the performance.
3.	Chaudhary et al. [28] (2023)	ML (self-organizing map (SOM)	XSS attack detection	GitHub, Kaggle	99.04%	The SOM is to classify the XSS attack well and the detection of XSS.	The FPR can be decreased by using techniques such as resampling, algorithm selection, hyper parameter tuning, cross-validation, ensemble methods, and feature selection.

(Continued)

Table 1 (continued)

No.	Authors	Methods	Aims	Dataset source	Accuracy	Pros	Cons
4.	Zhang et al. [29] (2022)	Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM)	XSS attack detection	CSE-CIC-IDS2018	99.2%	Presents a comprehensive study on the vulnerability of network intrusion detection systems to adversarial attacks and proposes robust defense mechanisms.	Requires extensive tuning and optimization to achieve practical, real-world applicability without significant performance overheads.
5.	Zhang et al. [30] (2019)	Convolutional neural network (CNN)	XSS attack detection	GitHub	95.29%	The paper presents an innovative approach to detecting XSS attacks using a convolutional neural network (CNN) model, which effectively utilizes byte-level analysis to improve detection rates.	The complexity of the model and the reliance on deep learning could potentially make it resource-intensive, possibly limiting its deployment in environments with constrained computational resources.
6.	Wang et al. [31] (2022)	Nearest neighbor and Naive Bayes	XSS attack detection	Real-world datasets	98.3%	A novel dynamic feature weighting algorithm for handling data streams that efficiently adapts to changes in data distribution, significantly enhancing the accuracy of classification algorithms like Nearest Neighbor and Naive Bayes.	It requires significant computational resources to continuously update feature weights and detect feature drifts dynamically which limits its practicality in resource-constrained environments.
7.	Mokbal et al. [19] (2021)	An ensemble-learning technique using the eXtreme Gradient Boosting algorithm (XGboost)	XSS attack detection	XSSed and Alexa	99.5%	Effectively utilizes the eXtreme Gradient Boosting (XGBoost) algorithm integrated with a hybrid feature selection approach, which significantly improves detection rates and accuracy for identifying XSS attacks.	The computational demand of the XGBoost algorithm might limit its accessibility or practicality for environments with constrained computational resources.

(Continued)

Table 1 (continued)

No.	Authors	Methods	Aims	Dataset source	Accuracy	Pros	Cons
8.	Mary et al. [32] (2024)	DL (Aquila Optimizer (AO) and Fuzzy Entropy Mutual Information (FEMI) algorithms, wildebeest Herd Optimization (WHO) algorithm) based on ResNet152	XSS attack detection	CICDDoS2019	99.46%	Due to the innovative use of the Aquila optimizer and fuzzy entropy mutual information for feature selection, which contributes to improved accuracy and performance.	Complexity of the proposed methods, including the integration of multiple advanced algorithms for feature selection and optimization, may present challenges in terms of computational demand and practical implementation.
9.	Pan et al. [33] (2024)	Few-shot graph (FSXSSED)	XSS attack detection	N/A	90%	The suggested dataset shows how well FSXSS performs in few-shot XSS attack detection.	The performance result can be improved by applying more techniques for performance enhancements.

Current XSS detection systems face several significant challenges that can hinder their effectiveness. Many systems struggle to adapt to the continuously evolving nature of XSS threats. Static detection methods that rely on predefined signatures or patterns often fail to detect new or sophisticated attack vectors.

For systems not optimized for real-time analysis, there is often a substantial delay between attack initiation and detection. This latency can be critical, as it allows attackers to exploit vulnerabilities before they are detected. Many systems do not cover all possible XSS attack vectors, particularly those involving obfuscated or novel payloads that do not match traditional patterns. As web applications grow in complexity, many XSS detection systems do not scale effectively, degrading performance as traffic volume increases.

3 Proposed Methodology

The goal of machine learning is to develop a predictive system that can learn from training data. This research employs a systematic approach to develop an advanced detection system for Cross-Site Scripting (XSS) attacks, which are prevalent threats in cybersecurity. Given the complexity and evolving nature of XSS vulnerabilities, traditional detection methods often fall short. Thus, this study integrates multiple machine learning algorithms to enhance detection accuracy and robustness.

Initially, the methodology involves the comprehensive collection and preprocessing of data to form a suitable training set. Data preprocessing includes handling missing values, encoding categorical variables, and normalizing data to improve algorithm performance. Following data preparation, we implement several machine learning models—Logistic Regression, Support Vector Machine (SVM), eXtreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), and a Deep Neural Network (DNN)—to establish baseline performances.

To further refine detection capabilities, a stacking ensemble method is introduced. This technique leverages the strengths of individual models by using their output as input for a final estimator, which

in this case is the CatBoost classifier. The ensemble model aims to capture more complex patterns that single models may miss, potentially increasing the predictive accuracy. Fig. 3 shows the design methodology that we are offering.

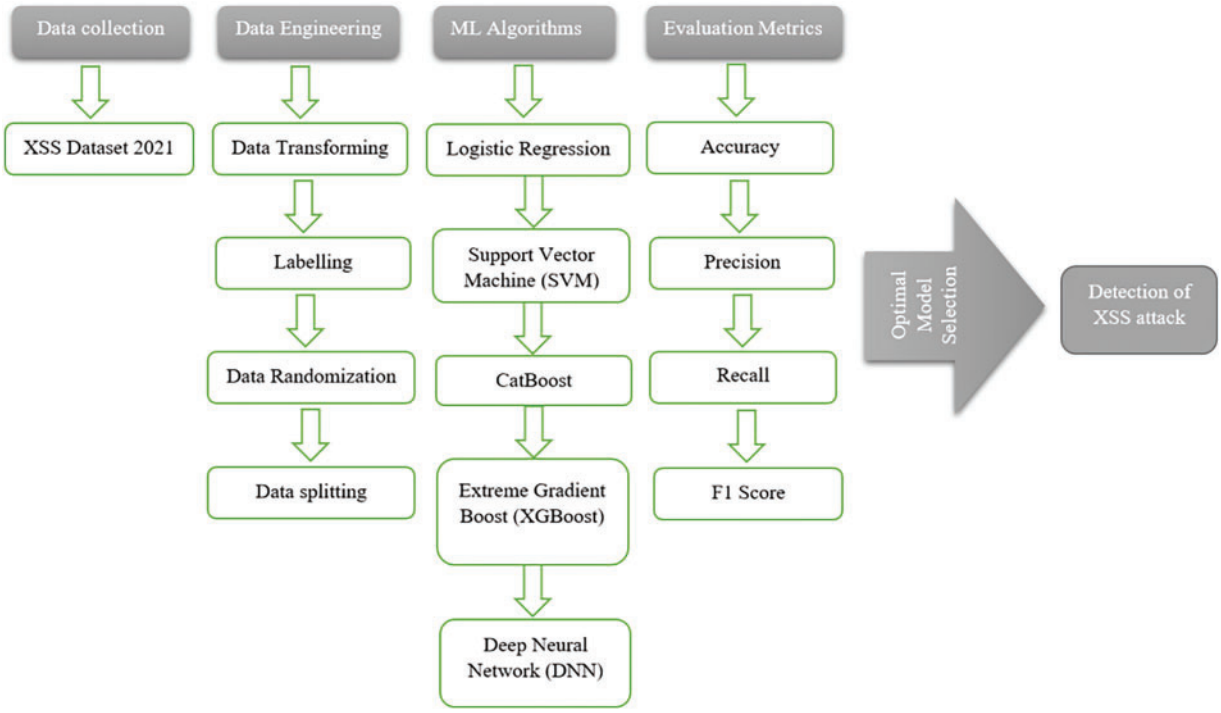

Figure 3: Workflow for offered XSS detection systems

The accompanying Fig. 3 illustrates the workflow diagram of our proposed cross-site scripting (XSS) detection system, concluding the machine learning (ML) algorithms section. This diagram provides a visual representation of the integrated approach, where Logistic Regression, SVM, XGBoost, CatBoost, and Deep Neural Networks (DNN) work in synergy to enhance the detection capabilities against XSS attacks. By presenting the workflow, we encapsulate the sequential and parallel processes involved, offering a clear depiction of how data flows through different phases of processing and analysis within our system. This visual aid is crucial for understanding the complex interactions and the multi-layered strategy employed to maximize detection accuracy and efficiency.

3.1 Machine Learning Approaches

There are four types of machine learning approaches: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

The supervised learning approach works well for learning from labeled data and predicting new or unseen data. Unlike supervised learning, unsupervised learning methods do not use labeled data. Unsupervised learning is less suited for classification problems; it can be useful for tasks such as identifying patterns in data or grouping similar attacks in the context of XSS detection. Semi-supervised learning uses a combination of a small amount of labeled data and a large amount of unlabeled data. It aims to improve learning accuracy when labeled data is scarce and expensive to obtain. Reinforcement learning algorithms learn by interacting with the environment and optimizing a

measurable reward signal. This approach matches input conditions to output behaviors that maximize the reward signal. Fig. 4 shows the ML model-based architecture of XSS detection.

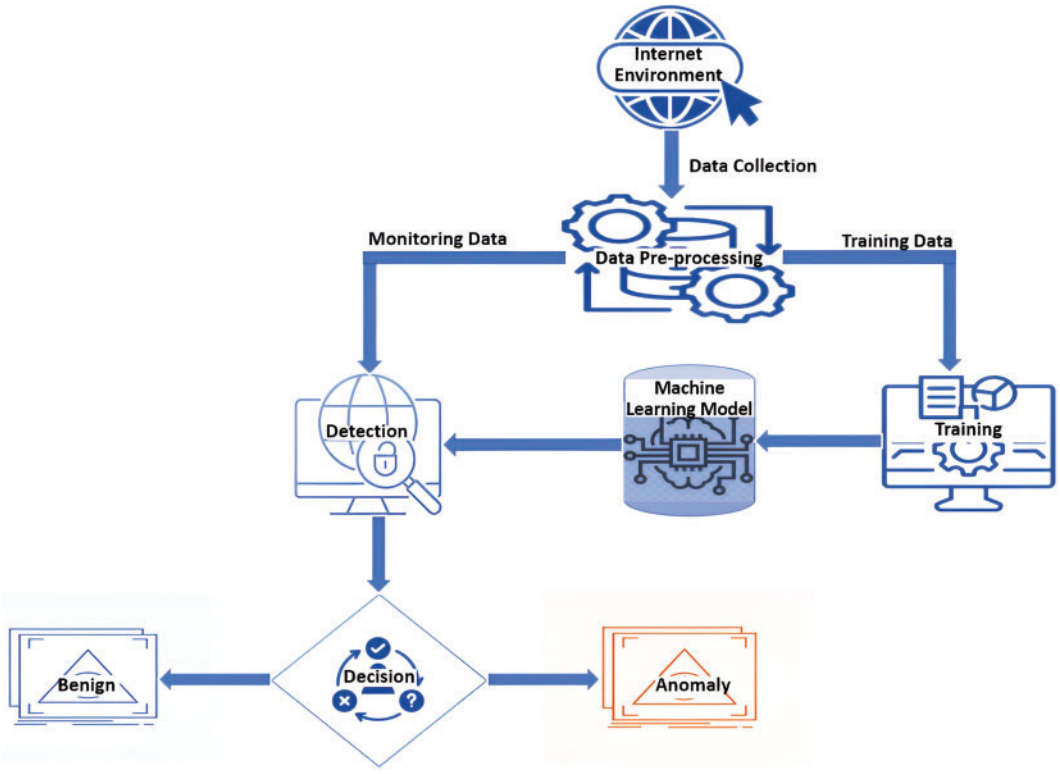

Figure 4: Machine learning-based cross-site scripting XSS attack detection architecture

In this research, we present an architecture designed to detect XSS attacks through machine learning techniques, as outlined in below Fig. 4. This system is constructed within an Internet environment, with data flow orchestrated in several critical stages to ensure effective learning and prediction capabilities.

There are two approaches we have used; one is a hybrid learning approach while the other is a stacking ensemble learning approach.

3.1.1 Hybrid Learning Approach

Hybrid learning in the context of XSS attack detection employs a combination of multiple machine learning models to improve the accuracy, robustness, and generalizability of the detection system. The hybrid approach leverages the strengths of individual algorithms, mitigating their weaknesses to create a more effective detection tool. Here's how hybrid learning is typically implemented in our framework.

Model Selection

The initial step involves selecting a range of diverse machine learning algorithms, each bringing its unique strengths to the table. In our research, we incorporate Logistic Regression, Support Vector Machines (SVM), XGBoost, CatBoost, and Deep Neural Networks (DNN). These models are

carefully chosen for their ability to effectively manage various aspects of the data and address different attack patterns.

Data Preprocessing

Data is prepared to suit the needs of each model. This involves normalization, scaling, handling missing values, and encoding categorical variables to ensure that the input data is optimized for machine learning tasks.

Training Individual Models

Each model is trained independently on the training dataset. This stage involves tuning hyper parameters, using techniques like grid search or random search, to find the optimal settings for each model.

Integration Strategy

After training, the predictions from each model are combined. This can be done through various ensemble techniques such as stacking, where the outputs of individual models serve as inputs to a final meta-model. The meta-model, often a machine learning algorithm itself, is trained to make final predictions based on the inputs it receives from the base models.

Model Optimization and Validation

The hybrid model undergoes rigorous validation to assess its performance. Techniques such as cross-validation are employed to ensure that the model performs well on unseen data and to prevent overfitting. Once validated, the hybrid model is deployed in a real-time environment where it can start detecting XSS attacks. The system is designed to adapt dynamically, updating its parameters in response to new threats and integrating feedback from its detections.

By leveraging multiple models, the hybrid learning approach benefits from the collective insights of different algorithms, leading to a more robust and effective XSS detection system. The diversity of models helps cover more attack vectors and reduces the likelihood of missing sophisticated or novel XSS attacks. This approach ensures that our detection system remains effective even as attackers evolve their strategies.

3.1.2 Stacking Ensemble Learning Approach

Stacking Ensemble Learning is a sophisticated machine learning method that combines multiple base models to enhance predictive performance by integrating their predictions through a meta-model. This approach leverages the diverse strengths and compensates for the weaknesses of various models, thereby improving overall accuracy and robustness. Here is how stacking ensemble learning is implemented in our XSS attack detection framework.

Base Learner Selection

In stacking ensemble learning, the first step involves selecting a diverse set of base learners. For our XSS detection, we employ Logistic Regression, SVM, XGBoost, CatBoost, and Deep Neural Networks. Each model brings unique capabilities to handle different aspects of the data, ensuring a comprehensive learning process.

Training Base Learners

Each base model is trained independently on the entire training dataset. This stage allows each model to develop its predictive strategy based on its inherent strengths, ensuring a variety of perspectives on the problem.

Meta-Model Selection

After the base models are trained, their predictions form a new dataset, where the actual outputs of these models serve as input features, while the original response variable remains the same. This dataset is used to train a higher-level model, known as the meta-model. In our framework, we typically use a CatBoost classifier as the meta-model due to its effectiveness in handling categorical features and complex patterns.

Integration of Predictions

The meta-model learns to effectively combine the predictions of the base models. It assesses which models are more reliable under certain conditions and assigns more weight to their predictions. This step is crucial as it synthesizes the strengths of individual models into a cohesive prediction.

Model Training and Validation

The entire ensemble, including both the base learners and the meta-model, is subjected to rigorous cross-validation to evaluate its performance and avoid overfitting. This involves partitioning the data into several subsets, training on several subsets while validating on others, and rotating which subsets are used for validation.

The stacking ensemble is designed to be flexible, allowing for incremental learning. As new data becomes available, or as XSS attack patterns evolve, new models can be trained and easily integrated into the existing ensemble without retraining the entire system. Once the stacking ensemble model has been validated, it is deployed into a real-world environment where it can start providing predictions. The system is configured to operate in real-time, dynamically adjusting to new data and maintaining high accuracy and low latency in XSS attack detection.

By implementing the stacking ensemble learning approach, our system not only improves the accuracy of XSS attack detection but also enhances its ability to generalize across different types of XSS attacks, making it highly effective in a real-world cybersecurity context. Our approach to XSS attack detection via a stacking ensemble framework introduces several novel configurations and algorithmic adaptations specifically tailored to address the unique complexities of XSS vectors.

Unlike traditional ensemble methods that often rely on homogeneous model types, our framework integrates a diverse set of algorithms including Logistic Regression, SVM, XGBoost, CatBoost, and Deep Neural Networks. This hybrid approach is specifically engineered to leverage the strengths of each model type—SVM and Logistic Regression for their robustness in high-dimensional spaces, XGBoost and CatBoost for their gradient boosting capabilities which excel in handling unstructured data, and DNNs for their proficiency in pattern recognition deep within data structures.

3.1.3 Data Collection

The data collection process forms the foundational basis of our research, enabling a thorough investigation into XSS attack detection. For this study, a comprehensive dataset was meticulously compiled to encompass a diverse range of XSS attack vectors and scenarios. The dataset comprises instances derived from real-world web application environments, ensuring the relevance and applicability of the research to contemporary cybersecurity challenges. For the detection impact of the model

to be more appropriate, it is important to select a suitable data set for training in addition to using a suitable strategy to extract and categorize feature information. This is how to determine whether the detection scheme is effective. The XSS attack dataset from Kaggle serves as the basis for the empirical study of the proposed work.

Data collection emphasized the inclusion of varied attack vectors to encapsulate the complexity and evolving nature of XSS threats. By simulating real-world conditions, the dataset provides a realistic framework for testing and enhancing the XSS detection capabilities of the proposed machine learning models.

3.1.4 Data Engineering

Data engineering plays a crucial role in enhancing the performance of machine learning models by ensuring that the data is adequately processed and prepared before it is used for training. In this research, several key data engineering steps were implemented to optimize the detection of XSS attacks. Fig. 5 illustrates the process by which we arrived at a more exact and comprehensive understanding of it.

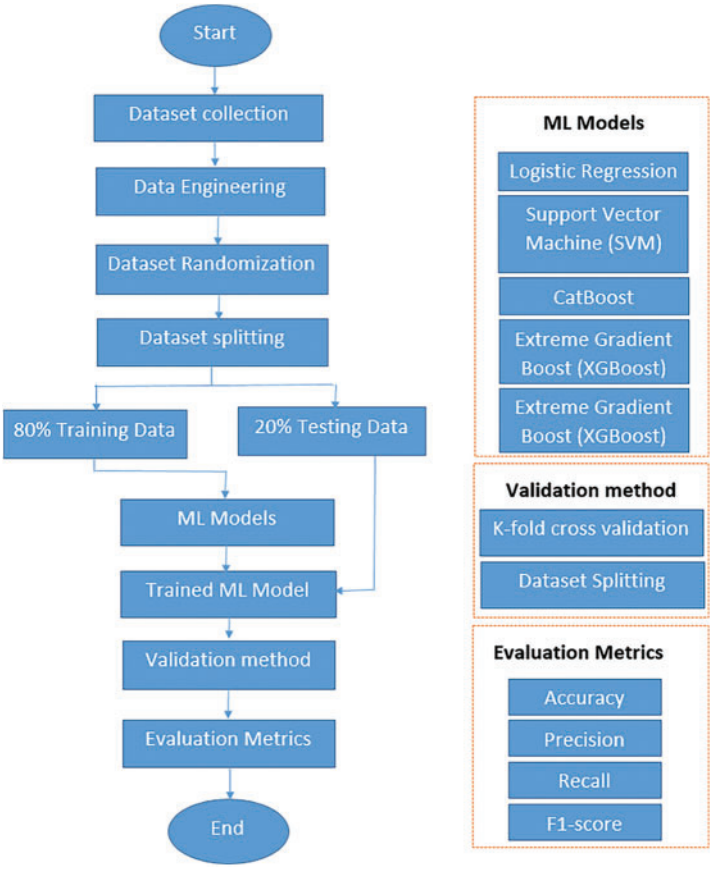

Figure 5: Classification of proposed cross-site scripting (XSS) detection methodological structure

Firstly, data cleaning is performed to remove any inconsistencies or errors in the dataset, such as duplicate entries or corrupt records. This step ensures the integrity and reliability of the training data, which is essential for building robust models. Following data cleaning, feature engineering was

undertaken to extract and select the most relevant features from the raw data. This process involved creating new features that better capture the characteristics of XSS attacks and selecting those that contribute most significantly to the predictive power of the models. Techniques such as feature scaling and normalization were also applied to standardize the range of the data features, thereby improving the convergence speed of the learning algorithms and enhancing model accuracy.

Moreover, dimensionality reduction techniques such as Principal Component Analysis (PCA) were employed to reduce the number of features in the dataset while retaining the most informative aspects. This not only helps in alleviating the computational burden but also aids in minimizing the risk of overfitting, making the models more generalizable to unseen data. These data engineering efforts collectively contribute to a more efficient and effective learning process, enabling the developed models to accurately identify potential XSS vulnerabilities within web applications.

Data Cleaning

Data cleaning is an essential preliminary step in the data engineering process that directly impacts the success of machine learning models. In this study, careful attention was devoted to cleaning the dataset used for detecting XSS attacks. The process began with the identification and removal of duplicate records to ensure that the training set did not artificially inflate the importance of certain observations.

Subsequently, we addressed missing or incomplete data entries. Given the critical nature of cybersecurity data, decisions on how to handle missing values were based on the nature of the data and the expected impact on model performance. For fields where data was missing, we employed imputation techniques where appropriate, using the median or mode values to fill gaps in numeric and categorical data, respectively.

Additionally, outlier detection and removal were conducted to prevent data anomalies from skewing the results and leading to model overfitting. Outliers were identified using statistical techniques such as z-scores and IQR (interquartile range), and decisions on outliers were made based on domain knowledge and their potential influence on model learning. By ensuring the cleanliness and integrity of the data, we laid a strong foundation for the subsequent stages of feature engineering and model training, ultimately enhancing the model's ability to generalize well to new, unseen data.

Data Balancing

Handling limited labeled data is a significant challenge in machine learning, particularly in cybersecurity contexts where attacks evolve rapidly and new threats emerge continuously. In this study, we addressed this challenge by employing robust dataset balancing techniques to ensure that our models are not biased towards the majority class, thereby improving their generalizability and effectiveness in detecting XSS attacks.

To balance our dataset effectively in scenarios with limited labeled data, we utilized the Synthetic Minority Over-Sampling Technique (SMOTE). This method is particularly beneficial in such situations as it helps to create a more diverse and representative dataset by generating synthetic examples. This approach prevents our model from merely predicting the most common class and encourages it to learn more nuanced patterns indicative of all classes. The balanced dataset not only promotes a fair representation of all classes but also enhances the predictive performance across various metrics, including accuracy, precision, recall, and F1-score.

Additionally, we integrated techniques like data augmentation and semi-supervised learning methods to maximize the utility of the available labeled data while leveraging unlabeled data to enhance

the model's learning. These strategies are pivotal in environments where acquiring fully labeled datasets is impractical or too costly.

Over Sampling

In the realm of machine learning, particularly when dealing with imbalanced datasets common in cyber security applications such as XSS attack detection, over-sampling techniques play a pivotal role. To address the imbalance in our dataset, where certain classes of attacks are underrepresented, we employed the Synthetic Minority Over-Sampling Technique (SMOTE). This method not only amplifies the minority class by generating synthetic examples rather than by oversampling with replacement but also helps to form a more general decision boundary, which improves the classifier's performance on unseen data.

SMOTE works by selecting examples that are close to the feature space, drawing a line between the examples in the feature space, and drawing a new sample at a point along that line. This approach is particularly useful in our context, as it ensures a richer and more nuanced representation of minority classes, thereby enhancing the robustness and accuracy of our predictive models. By employing SMOTE, we aimed to provide a more balanced dataset, which in turn facilitates more reliable learning and validation of our XSS attack detection models.

3.1.5 Feature Engineering

In our research, feature engineering plays a pivotal role in enhancing XSS attack detection. We meticulously selected features that are critical in identifying XSS patterns, such as JavaScript keywords, HTML attributes, and URL structures. Each feature was chosen based on its frequency and uniqueness in XSS samples, enabling our models to effectively distinguish between benign and malicious scripts. The engineered features were then processed using techniques like encoding and tokenization to prepare them for machine-learning analysis. This careful preparation of features significantly contributed to the robustness of our detection system, optimizing both accuracy and speed.

Tokenization of Script Parts

This involves breaking down JavaScript or any scripting content into manageable tokens or symbols. It helps in isolating script components that could potentially carry malicious payloads. For instance, from a script like <script>alert('XSS')</script>, tokenization would separate out <script>, alert, ('XSS'), and </script>.

Extraction of HTML Tags

Analyzing an HTML column, individual tags such as <a href="..."> or <img src="..."> are extracted. The presence of onerror within <img> tags could be specifically extracted as a feature indicative of potential XSS.

Encoding of Categorical Data

HTTP request methods like GET, and POST transformed from their textual representation into numerical codes to allow quantitative analysis by machine learning models.

Feature Construction

Constructing new features like the length of a URL or counting the number of special characters in the URL query parameters to detect unusual patterns commonly associated with XSS.

Feature Selection

JavaScript Event Handlers

Features capturing event handlers (like onclick, and onerror) are critical as they are often exploited in XSS attacks to execute malicious scripts when user interactions occur. For example, an onerror in a <img> tag could trigger malicious JavaScript if the image fails to load.

URL Patterns

Features analyzing URL structures might focus on patterns like unexpected use of special characters, javascript: pseudo-protocol, or overly long query parameters. For instance, a URL feature might capture a string like http://example.com/index.php?user=<script>.

Anomalies in HTML and JavaScript

This includes features that detect non-standard nesting of HTML tags, the presence of <script> tags in places they normally wouldn't be, or encoded JavaScript within HTML attributes. An example could be .

Input Validation and Encoding Anomalies

Features here could monitor how input fields handle data, looking for evidence of improper or lack of encoding which might allow special characters to trigger script execution. An example from the dataset could be input fields that accept "><script>alert('XSS')</script> without proper sanitation.

3.1.6 Model Tuning Process

In our research, significant emphasis was placed on the tuning of the machine learning models to optimize detection accuracy and efficiency. The tuning process involved several stages:

Hyper Parameter Optimization

Each model within our hybrid ensemble—Logistic Regression, Support Vector Machines (SVM), XGBoost, CatBoost, and Deep Neural Networks (DNN)—was subjected to hyper parameter optimization. We utilized grid search and random search methods to explore a wide range of hyper parameter settings. For instance, for XGBoost, parameters like max_depth, learning_rate, and n_estimators were optimized. For Deep Neural Networks, we adjusted the number of layers, the number of neurons in each layer, and the learning rate.

Validation Process

The effectiveness of our hybrid ensemble machine learning models is validated using a robust methodology designed to ensure accuracy and generalizability. We employ a two-pronged approach: dataset splitting and cross-validation.

Dataset Splitting

We split the dataset into training (80%) and testing (20%) sets. This division allows us to train the models comprehensively while reserving a portion of the data for independent evaluation, ensuring the models do not overfit the training data.

Cross-Validation

To further bolster the validation, we implement a Repeated Stratified K-Fold Cross-Validation strategy. This involves partitioning the data into ten folds, each fold being used once as a validation set while the others serve as training sets. This process is repeated three times with random shuffling of data in each cycle to prevent biases associated with the ordering of data points.

During each fold of the cross-validation, we fine-tune the hyper parameters of our models—ranging from learning rates in Gradient Boosting Machines to layer configurations in Deep Neural Networks. This tuning is guided by the performance metrics such as accuracy, precision, recall, and F1-score, observed during the validation phases. By integrating model tuning directly into the validation process, we ensure that the optimizations contribute directly to enhanced model performance, leading to a detection system that is not only accurate but also efficient and adaptable to new threats.

Ensemble Techniques

The stacking ensemble model was tuned by experimenting with different combinations of base models and meta-models. The objective was to find the best synergy between the models that could improve the predictive power beyond what was possible by individual models alone.

The tuning process had a measurable impact on the models' performance. For example, after tuning, the XGBoost model showed a 2% increase in accuracy and a significant reduction in false positives. The stacking ensemble, which combined outputs from finely tuned models, outperformed any single model with an accuracy improvement of over 5% compared to the best single model.

3.1.7 Adaptability to Evolving Threats

Our hybrid ensemble learning framework is designed to adapt to new and evolving XSS attack patterns that may not have been present in the initial training data. This adaptability is crucial given the dynamic nature of web security threats. To ensure continuous learning and adaptation, we have implemented several strategies:

Incremental Learning

The models within our framework, particularly those based on decision trees like XGBoost and CatBoost, are configured to support incremental learning. This allows the models to be updated with new data without the need for retraining from scratch, thus accommodating new attack vectors as they are identified.

Feedback Loops

We incorporate feedback loops into the detection system, where the outputs (detections) and the eventual outcomes (such as confirmed attacks) are used to refine the models continuously. This process involves re-assessing the model's predictions against actual outcomes and using this information to adjust the models for better accuracy.

Anomaly Detection Techniques

To capture attack patterns that deviate from known types, we employ anomaly detection techniques alongside traditional classification methods. These techniques are designed to flag unusual patterns in web traffic that could indicate novel XSS attacks, thus providing an additional layer of security.

Regular Updates to Training Data

We maintain a procedure for regularly updating the training datasets with new XSS instances as they are discovered, ensuring that the models are trained on the most recent data. This approach helps in capturing the latest tactics used by attackers.

Collaboration with Cybersecurity Communities

By collaborating with cybersecurity communities and platforms, we gain insights into emerging threats and incorporate this intelligence into our training processes. This collaboration helps in keeping the detection models up-to-date with the current threat landscape.

By integrating these adaptive mechanisms, our framework remains effective against XSS attacks, even as their patterns evolve. This ongoing adaptability is crucial for maintaining the robustness of web security defenses in the face of constantly changing cyber threats.

3.1.8 Defending against Adversarial Attacks

In response to the growing sophistication of adversarial attacks, where attackers deliberately manipulate inputs to evade detection, our framework incorporates several defensive strategies. We utilize adversarial training techniques, where the models are exposed to adversarial examples during the training phase. This exposure helps make the models robust against such manipulations, improving their ability to identify subtle anomalies that could indicate a manipulated attack vector. Additionally, rigorous input validation is employed to pre-screen inputs and reject those that are malformed or crafted to trigger false negatives in detection mechanisms. These steps ensure that our system maintains its integrity and effectiveness even when faced with sophisticated evasion tactics.

3.2 Machine Learning Algorithms

To address the complexities of XSS detection effectively, this study explores a variety of machine learning algorithms, each offering unique strengths in processing and pattern recognition. Our research delves into the capabilities and applications of each algorithm, aiming to harness their collective power through hybrid and ensemble methods to improve detection rates and reduce false positives. The section on ML algorithms provides a detailed examination of each chosen method, highlighting how they contribute to our comprehensive approach to XSS threat mitigation. we delve into several cutting-edge machine learning techniques, each selected for its potential to enhance the detection of Cross-Site Scripting (XSS) attacks. This section serves as a critical foundation for understanding how each algorithm—Logistic Regression, SVM, XGBoost, CatBoost, and Deep Neural Networks—can be uniquely tailored to address the challenges posed by XSS vulnerabilities.

This structured approach not only facilitates a deeper understanding of each algorithm's strengths and weaknesses but also sets the stage for subsequent sections where these models are applied and evaluated in real-world scenarios. Each subsection dedicated to a specific ML algorithm will explore its technical nuances, optimization strategies, and role within our broader XSS detection framework.

3.2.1 Logistic Regression

When dealing with binary classification problems, such as yes/no or normal/benign, where the outcome variable is categorical with two possible classes, one statistical technique that is utilized is logistic regression. Logistic regression is not a regression model; rather, it is a linear classification model. It works as, by mapping any real-valued input to a value between 0 and 1, the logistic (or sigmoid) function is used in logistic regression to estimate the probability of the binary result. The probability of belonging to one of the classes is then interpreted from the output of the logistic function. It is possible to make predictions depending on whether the probability exceeds a threshold by selecting a value such as 0.5. Logistic Regression is used to estimate the probability of a binary outcome based on one or more predictor variables. Logistic regression predicts the probability p that Y belongs to one of the categories (usually the "1" category) by using the logistic function. The probability that Y=1

can be modeled as a function of X (the predictors) is as follows:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k)}} \tag{1}$$

where:

– p is the probability that the dependent variable $Y = 1$,
– e is the base of the natural logarithm,
– $\beta_0 + \beta_1 + \beta_2 + \ldots + \beta_k$ are the coefficients of the model,
– $X_1 + X_2 + \ldots + X_k$ are the independent variables.

The coefficients $\beta_0 + \beta_1 + \beta_2 + \ldots + \beta_k$ represent the effect of each variable on the likelihood that $Y = 1$ adjusting for the effects of all other variables. The coefficients are usually estimated using maximum likelihood estimation.

The logistic model uses the logistic function, which outputs values between 0 and 1, making it appropriate for modeling probabilities. Additionally, it measures the discrepancy between expected probability and actual class labels using a logistic loss (also known as cross-entropy) function. This loss function is minimized during model training. It makes it simple to interpret the behavior of the model by providing interpretable coefficients that show how each characteristic affects the log odds of the result.

This model is optimized using techniques like maximum likelihood estimation to find the best coefficients (β values) that minimize prediction errors. By using logistic regression in both a hybrid setting and a sophisticated stacking ensemble, our project leverages its robustness in linear decision boundaries and probabilistic foundations, which are crucial for understanding the influence and impact of features on binary outcomes.

3.2.2 Support Vector Machine (SVM)

The Support Vector Machine (SVM) algorithm is a powerful supervised machine learning method used primarily for classification, though it can also be applied to regression tasks. SVM is fundamentally a classifier that finds an optimal hyperplane that maximizes the margin between two classes. The aim is to find the largest geometric margin between the classes, which involves solving an optimization problem. It works as a linear and non-linear.

In a two-dimensional linear space, SVM looks for a line that separates the classes with maximum margin. In higher dimensions, it seeks a hyperplane. Support vectors are the data points that are closest to the hyperplane and influence its position and orientation. SVM constructs its decision boundary based on these points. The margin is the distance between the closest points of different classes. SVM maximizes this margin to increase the model's generalization ability.

When data is not linearly separable, SVM uses a kernel trick to transform the input space into a higher-dimensional space where a hyperplane can be used to separate classes. The kernels function is used to map the data into a higher-dimensional space. Common kernels include linear, polynomial, Radial Basis Function (RBF), and sigmoid.

The objective of SVM is to minimize the weight vector (w), keeping in mind that the data points (x_i) of each class must be on the correct side of the hyperplane. This is expressed as:

To achieve this, SVM solves the following optimization problem:

– *Objective*: $Minimize \frac{1}{2} \|w\|$

– *Constraints*: $y_i\,(w.x_i + b) \geq 1$ for each $i = 1, \ldots, m$

where:

– w represents the weight vector of the hyperplane.
– b is the bias term that adjusts the hyperplane's offset from the origin.
– x_i and y_i are the feature vectors and labels respectively for each training sample.
– m represents the total number of training samples.

For cases where the data is not linearly separable, SVM uses kernel functions to map the input features into higher-dimensional spaces where a linear separation is feasible. Common kernels include:

– *Linear Kernel*: $K\left(x_i.x_j\right) = x_i.x_j$
– *Polinomail Kernel*: $K\left(x_i.x_j\right) = (\gamma\, x_i.x_j + r)^d$
– *Radial Basis Function Kernel*: $K\left(x_i.x_j\right) = exp\left(-\gamma \left\|x_i - x_j\right\|^2\right)$

Here, γ, r, and d are parameters that define the kernel's behavior, influencing the SVM's ability to classify complex datasets effectively.

SVM is used as a base learner in the stacking approach. It's configured to work with a StandardScaler within a pipeline to ensure the input features are standardized, which is crucial for optimal performance in SVM due to its reliance on calculating distances between data points. In the stacking setup, SVC (Support Vector Classification) is used with probability = True, enabling it to output probability estimates for classes that are necessary for the meta-learner in stacking to make effective final decisions.

3.2.3 eXtreme Gradient Boost (XGBoost)

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting that is widely used in machine learning competitions and industry applications for its effectiveness and efficiency.

It includes L1 and L2 regularization which helps to prevent overfitting, a problem common with classical gradient boosting methods. XGBoost automatically handles missing data, meaning it can directly accept datasets with missing values without requiring imputation. The split finding algorithm in XGBoost works depth-first (up to max depth) and then starts pruning the tree backward and removes splits beyond which there is no positive gain. XGBoost allows users to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting rounds in a single run. It utilizes a distributed computing method to efficiently handle large datasets and compute gradients and make splits faster, which significantly speeds up the computations. XGBoost involves creating decision trees in sequences, where each subsequent tree learns from the errors of its predecessors. Mathematically, the model tries to minimize the following loss function:

$$\mathrm{L}(\varphi) = \sum (i = 1 \; to \; n)\; l(y_\mathrm{i},\, \widehat{\mathrm{y}}_\mathrm{i}) + \sum (k)\Omega\,(f_k) \tag{2}$$

where:

– y_i-the actual value
– $\widehat{\mathrm{y}}_\mathrm{i}$-the predicted value
– Ω-the regularization term that penalizes the complexity of the model (like the number of leaves in the tree)

– l-a differentiable convex loss function that measures the difference between the predicted and actual values
– f_k-functions corresponding to individual trees.

The key to XGBoost's performance is its scalability, which has been achieved by optimizing both algorithmic enhancements and systems-level optimizations. This allows XGBoost to run fast even on relatively large datasets, making it practical for a wide range of data science problems and competitions.

In both the hybrid learning and stacking contexts, XGBoost serves as a base learner. It's configured with parameters like (use_label_encoder = False) to handle label encoding manually and (eval_metric = log_loss) to optimize the model for binary classification using logarithmic loss. By integrating XGBoost into both hybrid and stacking approaches, our research leverages its computational efficiency and predictive power, enhancing the overall ensemble's performance and providing robust insights into its applicability for our specific machine learning algorithms.

3.2.4 Categorical Boost (CatBoost)

CatBoost (Categorical Boosting) is a machine learning algorithm, which is designed to work well with categorical data and is capable of handling hundreds of categories efficiently. CatBoost optimizes the following objective function, which includes both the model's predictions and regularization terms to control overfitting:

$$Obj = \sum_{i=0}^{n} L(y_i, \widehat{y}_i) + \Omega(f) \tag{3}$$

where:

– $L(y_i, \widehat{y}_i)$-the loss function evaluating the error between the predicted $\widehat{y}_i$ and the actual y_i values.
– $\Omega(f)$-the regularization term which includes penalties on the structure and weights of the decision trees to prevent overfitting.
– f-represents the ensemble of decision trees.

CatBoost employs ordered boosting, which enhances the handling of categorical variables by using a permutation-driven algorithm to maintain prediction accuracy and reduce overfitting. The regularization part, (Ω) can include components such as the L2 regularization term on the weights of the trees. CatBoost optimizes the model using ordered boosting, a permutation-driven alternative to the classic gradient boosting method. The key feature of ordered boosting is that it reduces the prediction shift caused by a straight forward implementation of gradient boosting, which can lead to significant overfitting. The algorithm also employs an innovative approach to processing categorical variables. Instead of converting categories to numbers using traditional methods (like one-hot encoding), CatBoost uses a more sophisticated method that takes into account the statistical properties of the data, which reduces the risk of overfitting and improves the quality of the model.

CatBoost is utilized both as a standalone model in the hybrid approach and as a final estimator in the stacking ensemble. This dual role highlights its versatility and robustness. The CatBoost model is run with verbose = 0 to keep the output clean during training. Other parameters are set to default, which are generally well-tuned for a wide range of datasets.

As a meta-learner, CatBoost integrates predictions from various models. Its ability to handle categorical data and complex patterns makes it an excellent choice for refining and improving upon the predictions made by the base learners. It reduces overfitting without extensive hyper parameter

tuning, making it effective straight out of the box for many practical applications. This structured and layered approach to model evaluation—from individual model assessment to sophisticated ensemble strategies like stacking—provides a comprehensive framework for predictive accuracy and robustness in machine learning tasks.

3.2.5 Deep Neural Network (DNN)

Deep Neural Network (DNN) is an advanced machine learning model that mimics the way the human brain operates, allowing it to learn from large amounts of data. DNNs are particularly useful for complex pattern recognition tasks, which makes them ideal for cybersecurity applications like detecting XSS attacks. Here, the Deep Neural Network is implemented using the TensorFlow library through the Keras API, encapsulated within the (*Scikit-Learn*) interface using (*scikeras.wrappers.KerasClassifier*). This allows the DNN to be used similarly to traditional (*scikit-learn*) classifiers, making it compatible with functions like (*cross_val_score*) and fitting seamlessly into the ensemble methods like stacking.

The DNN is constructed with an input layer that matches the dimensionality of the dataset (*input_dim*). It includes two hidden layers with 20 and 10 neurons, respectively, both using the ReLU activation function for non-linearity. The output layer consists of a single neuron with a sigmoid activation function, which is typical for binary classification tasks. The model is compiled with the Adam optimizer, a popular choice for deep learning tasks due to its efficiency in handling sparse gradients and adapting the learning rate during training. The loss function used is (*binary_cross_entropy*), suitable for binary classification problems.

In our proposed work, the stacking framework of DNN acts as one of the base models. Its predictions, along with those from other models, are used by the {*meta-model*} (CatBoost in this case) to make the final prediction. This can potentially leverage the DNN's ability to extract complex patterns and features, improving the robustness and accuracy of the ensemble prediction. This approach allows the deep learning model to contribute its strengths to a collective (decision-making) process, harnessing both traditional algorithms and advanced neural network capabilities to achieve potentially higher performance than any single model could on its own.

3.3 Validation Process

It is crucial to validate the model's performance to make sure the model can predict the target parameter accurately when using ML techniques. The cross-validation methodology and dataset splitting are two of the most popular ways to assess a model's performance during the validation process. The process of dividing the dataset into training and testing sets is called dataset splitting. The model is trained on the training set, and its performance is assessed on the testing set. By going through this procedure, we make sure that the model can effectively generalize to new data and doesn't over fit the training set. The model's performance is influenced by the percentage of data splitting that is used for testing and training. A higher proportion of the data is usually utilized for training, as in our case 80%, and a lower proportion is used for testing or validation, as in our case 20%. Depending on the amount of the dataset and the complexity of the model, different percentages may be employed. Cross-validation is a widely used machine-learning technique for model validation.

To mitigate the potential for overfitting and ensure the robustness of our XSS attack detection models, we implemented a Repeated Stratified K-Fold Cross-Validation approach. This method involves dividing the entire dataset into ten distinct folds, promoting a thorough and balanced assessment of the model's performance. Each fold serves sequentially as the validation set, with the

remaining nine folds used for training. This process is not only repeated three times to enhance the reliability of our results but also randomized in each repetition to ensure that the validation is robust against any specific partition of data.

This extensive cross-validation helps in diagnosing and diminishing overfitting, ensuring that our models generalize well to new, unseen data. By averaging the outcomes from all folds and repetitions, we derive a comprehensive measure of our models' accuracy, precision, recall, and F1-score, providing a holistic view of their performance across varied subsets of data.

3.4 Scalability and Performance under Load

To ensure that our XSS detection system is scalable and performs efficiently under the stress of large-scale web applications, we have incorporated several key strategies. First, our ensemble learning framework utilizes computationally efficient models that allow for rapid response times, essential for high-traffic environments. We also leverage parallel processing capabilities inherent in models like Deep Neural Networks and CatBoost, utilizing multi-threading and GPU acceleration to enhance processing throughput. Additionally, we implement specific optimizations to manage high-traffic volumes effectively.

3.4.1 Load Balancing

To distribute incoming web traffic evenly across multiple servers, ensuring no single server bears too much load.

3.4.2 Caching Mechanisms

Employing caching strategies to store recently accessed data, reduces the number of times data needs to be recalculated or fetched, which is critical during peak traffic periods.

3.4.3 Asynchronous Processing

Utilizing asynchronous data processing to handle non-critical operations outside the main execution thread, thereby improving the throughput and responsiveness of our system.

Our scalability tests, involving simulated web traffic, demonstrate that our system maintains high accuracy and low latency as the load increases. This is supported by our efficient use of computational resources, ensuring that memory and CPU usage are optimized for large-scale operations.

3.4.4 Real-Time Detection Capabilities

The proposed hybrid ensemble learning framework is designed not only to efficiently detect XSS attacks in static datasets but also to operate effectively in real-time environments. Our system integrates real-time data processing capabilities to continuously analyze incoming web traffic and instantaneously identify XSS attack vectors as they occur. This real-time detection is facilitated by the computational efficiency of the models used, such as Logistic Regression, SVM, XGBoost, CatBoost, and DNN, which are optimized for fast execution without compromising accuracy. The real-time operation is supported by our deployment architecture, which is capable of handling high-throughput web traffic, ensuring that XSS attack detection keeps pace with the dynamic nature of user interactions in web applications. This functionality is critical for applications requiring immediate response to security threats, thereby significantly enhancing web application security and user trust.

3.4.5 Integration with Existing Systems

To ensure the practical applicability of our hybrid ensemble learning framework for XSS attack detection, it is crucial to consider its integration within existing web security systems and frameworks. This integration aims to enhance the current security measures without necessitating a complete overhaul of the existing infrastructure.

Integration Process

API Compatibility

Our system is designed to interface seamlessly with common security platforms through well-defined APIs. This allows the ensemble model to receive data from and send alerts to existing security monitoring tools.

Modular Design

The framework is built as a modular component, which can be plugged into existing security solutions, such as Web Application Firewalls (WAFs) and Intrusion Detection Systems (IDS), enhancing their capability to detect and respond to XSS attacks more accurately.

Customization and Configuration

Given the variability in security needs and existing configurations across different systems, our framework supports extensive customization. This flexibility ensures that it can be tuned to work effectively with the specific security protocols, data formats, and operational workflows of the target environment.

Potential Challenges

Compatibility Issues

Differences in data formats and communication protocols may require the development of additional adapters or middleware solutions.

Performance Impact

Integration might introduce latency or computational overhead, which needs to be mitigated through optimization to maintain system responsiveness.

Security Concerns

Introducing new components to existing systems could potentially open up new vulnerabilities. Rigorous security testing is essential to ensure that the integration does not compromise the overall security posture.

By addressing these aspects, the proposed XSS detection system not only complements but significantly enhances the capabilities of existing web security frameworks, making it a versatile and powerful tool in the fight against XSS vulnerabilities.

3.5 Computational Efficiency

To address the computational demands of the hybrid ensemble learning approach, our methodology incorporates several strategies to optimize system performance without sacrificing accuracy. Here, we detail the techniques employed to manage the computational overhead.

3.5.1 Parallel Processing

By leveraging parallel computing capabilities, our system distributes the workload across multiple processors. This is particularly crucial for models like Deep Neural Networks and CatBoost, which are inherently parallelizable. Utilizing GPU acceleration, we significantly reduce training and prediction times, ensuring that the system scales effectively with increasing data volumes.

3.5.2 Efficient Algorithm Implementation

Each model in the ensemble is optimized for performance. For instance, XGBoost is configured to maximize computational efficiency by adjusting tree construction algorithms and iterating over a reduced number of splits. Similarly, logistic regression and SVM models are implemented with solvers that are optimized for large datasets.

3.5.3 Resource Management

We implement dynamic resource allocation based on load predictions to ensure that the system remains responsive during peak usage. This involves adjusting the computational resources in real-time, such as scaling up the number of processing units during high-demand periods and scaling down during low-traffic times.

3.5.4 Impact Assessment

Our experiments show that, despite the intensive computation required for training and tuning multiple models, the operational impact during deployment is minimal. The system maintains a low latency response in real-time XSS detection scenarios, confirmed by stress-testing the system under simulated high-traffic conditions.

By integrating these strategies, our XSS detection system not only meets the accuracy and speed requirements of modern web applications but also ensures that the increase in computational overhead does not impede overall system performance.

3.6 Advantages of Employing Ensemble Learning

Ensemble learning, which involves the integration of multiple learning models to improve the robustness and accuracy of predictions, offers several significant benefits in the context of cybersecurity specifically in XSS detection.

3.6.1 Improved Accuracy

Ensemble methods typically achieve higher accuracy than individual models by aggregating the decisions from multiple learning algorithms. This reduces the risk of an incorrect prediction by any single model having a significant negative impact on the overall detection capability.

3.6.2 Reduced Variance

By combining multiple models, ensemble learning helps in reducing the variance of predictions. Each model's errors are likely to cancel out when averaged together, leading to more stable and reliable detection performance.

3.6.3 Enhanced Generalization

Ensemble learning enhances the generalizability of the detection system. By leveraging diverse algorithms, the ensemble method is less likely to overfit the idiosyncrasies of the training data, making it more effective at handling new, previously unseen XSS attack patterns.

3.6.4 Robustness to Noise and Outliers

In the context of XSS detection, data can often be noisy and contain outliers. Ensemble methods, particularly those that use robust learners like Random Forests or Boosted Trees, are generally better at handling such data irregularities without compromising detection accuracy.

3.6.5 Capability to Leverage Strengths of Different Models

Different models may have different strengths regarding certain types of XSS attacks. Ensemble learning allows the system to leverage these strengths, whereby each model can focus on a particular aspect of the data or attack pattern, leading to a more comprehensive detection strategy.

3.6.6 Flexibility in Model Update and Maintenance

Ensemble systems can be updated incrementally, which is a significant advantage in the rapidly evolving landscape of web threats. New models can be added or outdated ones removed without complete retraining, allowing the detection system to adapt quickly to emerging XSS techniques.

3.6.7 Operational Resilience

The redundancy inherent in ensemble methods provides operational resilience. Even if one or a few models fail or become outdated, the system as a whole can continue to operate effectively, thereby maintaining high detection rates.

3.7 Algorithms Implementation and Evaluation Metrics

Evaluation reveals significant insights into the capabilities of both traditional and advanced machine learning techniques in the realm of cybersecurity. The Stacking Ensemble model, which integrates multiple base learners, notably achieves superior performance, underscoring the benefits of leveraging combined model strategies over single-model approaches. This ensemble model not only yields an outstanding accuracy of 99.87% but also maintains perfect precision, demonstrating its exceptional reliability in minimizing false positives—a critical aspect in cybersecurity operations.

Hybrid learning in machine learning refers to a strategy that combines multiple algorithms to achieve better predictive performance compared to a single model. Our implementation includes diverse machine learning models—Logistic Regression, SVM, XGBoost, CatBoost, and a Deep Neural Network.

The following Python script in **Algorithm 1** demonstrates our approach to loading the dataset, preprocessing the data, training several machine learning models, and evaluating their performance on a set of metrics including accuracy, precision, recall, and F1-score.

Algorithm 1: Hybrid Learning Models Script for Model Evaluation

Input: Dataset file path

Output: Model performance plot

Algorithm:

Load Data

1. Read the dataset from the file path using pd.read_excel(file path, engine='openpyxl').
2. For each column in the DataFrame, if the column type is 'object', encode it using LabelEncoder().fit_transform().
3. Split the data into features (X) and labels (y) by dropping the 'Label' column and assigning it to y.

Create Deep Learning Model

1. Initialize the model using Sequential().
2. **Add Dense layers**
   1. Dense(20, input_dim=input_dim, activation='relu')
   2. Dense(10, activation='relu')
   3. Dense(1, activation='sigmoid')
3. Compile the model with optimizer='adam', loss='binary_crossentropy', and metrics=['accuracy'].

Evaluate Models

1. Initialize an empty list of results.
2. Define models to evaluate
   models = { 'Logistic Regression': make_pipeline(StandardScaler(), LogisticRegression()),
   'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
   'CatBoost': CatBoostClassifier(verbose=0),
   'SVM': make_pipeline(StandardScaler(), SVC())}
3. Set up cross-validation with RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1).
4. **For each model, perform cross-validation and prediction**
   1. Calculate scores using cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1).
   2. Predict using cross_val_predict(model, X, y, cv=cv).
   3. Calculate metrics: accuracy, precision, recall, f1_score.
5. **Append results to the list**

```
results.append({ 'Model': name,
'Accuracy': np. mean(scores),
'Precision': precision_score(y, y_pred),
'Recall': recall_score(y, y_pred),
'F1-Score': f1_score(y, y_pred),
'False Positive': FPR,
'False Negative': FNR  }).
```

(Continued)

Algorithm 1 (continued)

Plot Results

1. Convert the results list to a data frame.
2. Set the index of the DataFrame to the model names.
3. Plot the performance metrics using plot(kind='bar', figsize=(14, 8)).
4. Customize the plot with titles, labels, grid, and layout.
5. Display the plot using plt. show().

Main Execution

1. Define the file path to the dataset.
2. Load the data using load_data(filepath).
3. Evaluate the models using evaluate_models(X, y).
4. Print the results.
5. Plot the results using plot_results(results_df).

We use techniques like stacking to combine predictions from multiple models. Stacking uses a (*meta-model*) to learn how best to combine the predictions from the base models. By doing so, it can potentially capitalize on the strengths and minimize the weaknesses of the individual models involved.

In our study, we employ various machine learning models to analyze and predict the outcomes of XSS attack vectors. Below **Algorithm 2** is the Python implementation used to load data, prepare models, and evaluate their performance through a stacking ensemble approach. This method leverages multiple base learners to enhance predictive accuracy.

Algorithm 2: Stacking Ensemble Learning for Model Evaluation

Input: Dataset file path
Output: Model performance plot

Algorithm:

Load Data

1. Read the dataset from the file path using pd.read_excel(file path, engine='openpyxl').
2. For each column in the DataFrame, if the column type is 'object', encode it using LabelEncoder().fit_transform().
3. Split the data into features (X) and labels (y) by dropping the 'Label' column and assigning it to y.

Create Deep Learning Model

Function build_fn() that

1. Initializes the model using Sequential().
2. Adds Dense layers.
3. Dense(20, input_dim=input_dim, activation='relu').
4. Dense(10, activation='relu').
5. Dense(1, activation='sigmoid').
6. Compiles the model with optimizer='adam', loss='binary_crossentropy', and metrics=['accuracy'].
7. Return a KerasClassifier using the build_fn(), with specified ep7ochs, batch_size, and verbose.

(Continued)

Algorithm 2 (continued)

Evaluate Models:

1. Define the base estimators:
   estimators = [('lr', make_pipeline(StandardScaler(), Logistic Regression())),
   ('svm', make_pipeline(StandardScaler(), SVC(probability=True))),
   ('xgb',XGBClassifier(use_label_encoder=False, eval_metric='logloss')),
   ('cat', CatBoostClassifier(verbose=0)),
   ('dnn', create_deep_model(X.shape [1]))]
2. Create a StackingClassifier with the base estimators and a final estimator (e.g., CatBoostClassifier), using cross-validation with RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1).
3. Define the evaluation metrics: accuracy, precision, recall, f1.
4. Perform cross-validation and prediction using cross_validate() and obtain the scores.
5. Calculate the confusion matrix using confusion_matrix() with the predictions.
6. **Store the results in a dictionary**
   results = { 'Model': 'Stacking',
   'Accuracy': np.mean(scores['test_accuracy']),
   'Precision': np.mean(scores['test_precision']),
   'Recall': np.mean(scores['test_recall']),
   'F1-Score': np.mean(scores['test_f1']),
   'False Positive': FPR,
   'False Negative': FNR }.

Plot Results

1. Extract metric names and their scores from the results dictionary.
2. Create a bar plot of the performance metrics using plt. Bar().
3. Customize the plot with titles, labels, and limits.
4. Display the plot using plt. show().

Main Execution

1. Define the file path to the dataset.
2. Load the data using load_data(filepath).
3. Evaluate the models using evaluate_models(X, y).
4. Print the results.
5. Plot the results using plot_results(results).

By measuring the performance of the ML-based model using conventional evaluation variables, we will evaluate the validity of the presented models. Confusion matrix analysis will be used in this regard to reveal information about the number detection accuracy, precision, sensitivity/recall (RE), and detection F1-score. Which shows the true and anticipated class confusion matrix analysis along with the related assessment measures.

3.7.1 Accuracy

The ratio of correctly identified samples to the total number of samples that have been classified represents accuracy.

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

3.7.2 Precision

Precision is defined as the ratio of accurately predicted samples to the total number of positive samples.

$$Precison = \frac{TP}{(\mathrm{TP} + FP)} \tag{5}$$

3.7.3 Detection Sensitive/Recall

The ratio of positive samples that were correctly predicted (TP) to the total number of classifications in the same real class is known as the detection sensitivity/recall (RE) statistic. The recall measure serves to illustrate the model's FN.

$$Recall = \frac{TP}{(\mathrm{TP} + FN)} \tag{6}$$

3.7.4 F-Measure

Recall and precision measurements are combined into one measure, known as the F-measure.

$$F1 - measure = 2 * \frac{(Recall * Precision)}{(Recall + Precision)} = \frac{2TP}{2(TP + FP + FN)} \tag{7}$$

3.7.5 False Positive

The number of false positives (FP) divided by the sum of false positives and true negatives (TN), which can be expressed as:

$$FPR = \frac{FP}{(\mathrm{FP} + TN)} \tag{8}$$

3.7.6 False Negative

The number of false negatives (FN) divided by the sum of false negatives and true positives (TP), which can be expressed as:

$$FNR = \frac{FN}{(\mathrm{FN} + TP)} \tag{9}$$

4 Results and Discussion

This section outlines the specifics of our XSS detection approach's implementation as well as the results of our experiments. Afterward, we emphasize the findings of the performance assessment and the comparative analysis of the suggested strategy. The analysis is based on the performance metrics obtained from the implemented models, including Logistic Regression, SVM, XGBoost, CatBoost, and a Deep Neural Network, as well as the integrated stacking ensemble approach.

Firstly, the results discuss the individual performance of each model, highlighting their accuracy, precision, recall, and F1-score. This provides an understanding of how each model contributes to identifying XSS vulnerabilities under different scenarios. The effectiveness of the models is compared to identify which algorithms perform best in isolation and why certain models might lag in performance.

Subsequently, the discussion shifts to the performance of the stacking ensemble model, emphasizing its ability to amalgamate the predictive capabilities of the base models to enhance overall accuracy and reliability.

4.1 Confusion Metrix Analysis

A crucial component of our analysis involves the use of confusion matrices to visualize the performance of each model tested in this study. This section provides detailed confusion matrices for each machine learning algorithm employed—Logistic Regression, SVM, XGBoost, CatBoost, Deep Neural Network, and our composite stacking ensemble model. Each confusion matrix presented here illustrates the accuracy, recall, F1-score, and precision. By examining these matrices, we gain insight into not only the accuracy but also the type-specific performance of each model, highlighting their strengths and weaknesses in detecting XSS attacks.

Through this analysis, we aim to decipher the sensitivity (recall) and specificity, along with the precision and F1-score of the models, providing a comprehensive view of their predictive capabilities.

4.2 Detection Sensitivity|Recall (RE), True Positive Rate (TPR)

The performance of the proposed models in detecting XSS attacks is quantitatively assessed through key metrics such as detection sensitivity (recall) and the True Positive Rate (TPR). These metrics are pivotal in evaluating the effectiveness of cybersecurity measures, particularly in how well they identify actual positive instances of attacks. The results, show varied effectiveness across the models, with the Stacking Ensemble and CatBoost models demonstrating superior performance in both recall and TPR, suggesting their higher reliability in detecting XSS attacks. This analysis not only helps in understanding which models are most effective but also aids in fine-tuning the detection frameworks to enhance overall security posture against XSS vulnerabilities. All of the generated models (Logistic Regression, SVM, XGBoost, CatBoost, Deep Neural Network, and our composite stacking ensemble model) demonstrate evaluation outcomes of true positive rates (TPR) that show ideal sensitivity, the Stacking Ensemble model has demonstrated the highest sensitivity rate toward harmful traffic observations (the "Yes" class), with a nearly optimum TPR rate of 99.71%. The outcomes of the CatBoost model, which scored less optimally with 99.57% TPR. The SVM model scores a non-optimal model with just 86.52% for recall, showing the lowest sensitivity rates.

4.3 Precision Matrix Analysis

In evaluating the precision of the different models used in detecting XSS attacks, precision serves as a critical metric, indicating the proportion of identifications that were indeed correct. High precision reflects a model's ability not only to identify true positives but also to minimize the occurrence of false positives—erroneous identifications of non-threats as threats. This is especially vital in cybersecurity applications where false positives can lead to unnecessary resource allocation and potential disruption to user activities.

Stacking Ensemble and CatBoost showed exceptional precision at 99.8%, indicating no false positives among the predicted positives. Logistic Regression, SVM, XGBoost, and DNN also achieved

a precision of 99.2%, 98.9%, 99.17%, and 99.1%, underscoring their effectiveness in accurately identifying XSS attacks without over-flagging safe instances.

These results underscore the high fidelity of the proposed models in discerning true XSS attacks, making them suitable for integration into security systems where high precision is paramount to avoid the costs associated with false alarms. The analysis further elaborates on the strength of ensemble approaches like Stacking, which leverage the combined capabilities of multiple models to refine detection accuracy and precision.

4.4 Performance Evaluation

This section presents a detailed examination of the performance metrics of various models deployed to detect XSS attacks, highlighting their effectiveness across several statistical measures including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of each model's ability to identify and accurately classify XSS attacks, which is crucial for the deployment of robust cybersecurity measures in real-world scenarios.

In the given script in **Algorithm 1**, the (*plot_results*) function creates a bar plot to visually represent the performance metrics of the ensemble model, facilitating an easy comparison of how well the model performs across different metrics. Fig. 6 shows the graphical representation of models (LR, SVM, XGBoost, CatBoost, and DNN) performance concerning (Accuracy, Precision, Recall, F1-score).

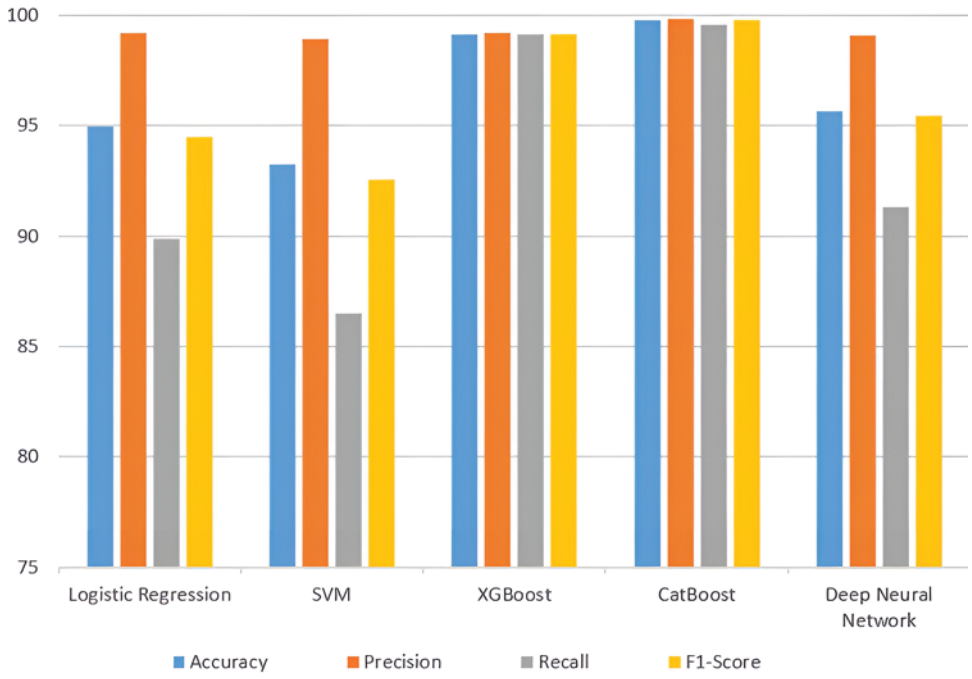

Figure 6: Analysis of the performance evaluation of our proposed models in terms of F1-score, recall, accuracy, and precision in detecting

In the process of determining the most effective machine learning model for detecting XSS vulnerabilities, we conducted a comparative analysis using several well-known algorithms. The following Table 2 summarizes the performance of each model across multiple metrics, crucial for assessing their effectiveness in real-world scenarios.

Table 2: Analysis of performance evaluation for every model in terms of detection accuracy, detection precision, recall, and F1-score

Models	Accuracy	Precision	Recall	F1-score	FPR	FNR
Logistic regression	94.93%	99.2%	89.86%	94.49%	0.38%	1.8%
Support vector machine (SVM)	93.26%	98.9%	86.52%	92.57%	0.52%	2.5%
eXtreme gradient boost (XGBoost)	99.13%	99.17%	99.13%	99.13%	0.39%	0.41%
Categorical boost (Cat Boost)	99.78%	99.8%	99.57%	99.77%	0.14%	0.2%
Deep neural network (DNN)	95.65%	99.1%	91.31%	95.46%	0.42%	1.7%

In **Algorithm 2** stacking, predictions from the base models (like SVM, Logistic Regression, XGBoost, DNN, and CatBoost) are used as input for a meta-model that makes the final prediction. This layering of models helps to blend the diverse strengths of each model into a more robust prediction mechanism. The (*load_data*) function loads data from an Excel file, processes categorical variables using *LabelEncoder*, and separates the dataset into features (X) and the target variable (y). This step is crucial for preparing our data for effective machine-learning modeling. (*create_deep_model*) function defines a deep neural network model using Keras, with layers designed to progressively extract higher-level features from the input data. The model is compiled with the Adam optimizer and binary cross-entropy loss function, suitable for binary classification tasks.

In **Algorithm 2**, multiple models including Logistic Regression, SVM, XGBoost, CatBoost, and a deep neural network are set up. Each model is capable of capturing different patterns in the data, and their diversity is beneficial for the ensemble approach. These models are integrated into a Stacking Classifier, where predictions from each base model (level-0 models) are used as input to a final estimator (level-1 model), in our case, another instance of CatBoost. This layered approach allows the ensemble to leverage the strengths of each model.

The (*evaluate_models*) function applies cross-validation to assess the performance of the stacking ensemble. It uses Repeated Stratified (*K-Fold*) for generating train/test splits, ensuring that each class is appropriately represented in each fold. This helps in evaluating the model's performance reliably across different subsets of data.

Accuracy, precision, recall, and F1-score are calculated to provide a comprehensive view of the model's performance. These metrics are critical for understanding various aspects of model performance, especially in a classification context where the balance between different types of errors is important. Fig. 7 visually represents the performance metrics of the ensemble model, facilitating an easy comparison of how well the model performs across different metrics. Table 3 provides a comparative overview of the performance metrics across different models under hybrid learning and stacking ensemble methods. Notably, stacking ensemble methods demonstrated improved accuracy across most models, as shown in the Table 2.

The comparison shows that the stacking ensemble method ("Ensemble Learning"), presumably including Logistic Regression, SVM, XGBoost, CatBoost, and DNN as base models with CatBoost as the final estimator, achieves the highest accuracy and F1-score among all setups. This suggests that stacking effectively leverages the strengths of individual models to improve overall performance.

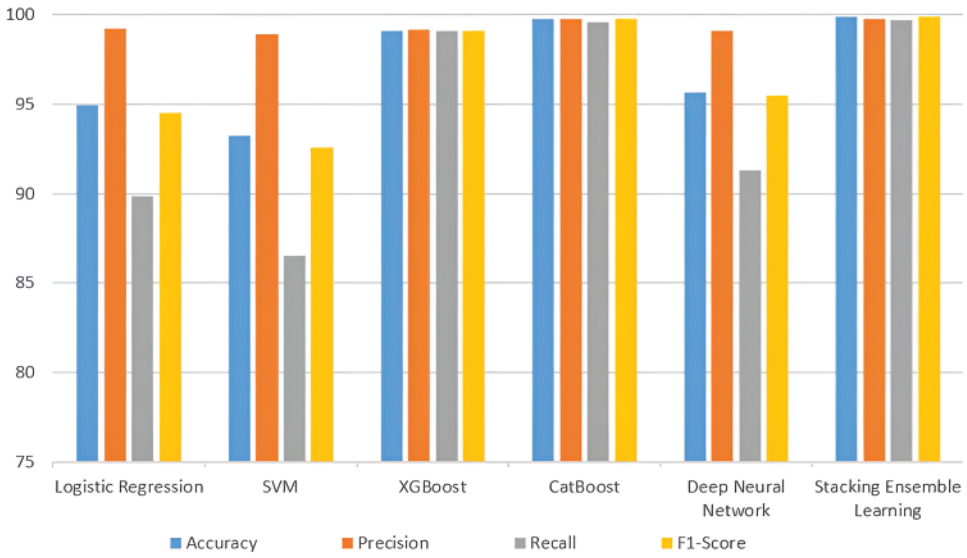

Figure 7: Analysis of the performance evaluation of our proposed ensemble learning models

Table 3: Comparison of performance evaluation of every model with stacking ensemble learning in terms of detection accuracy, detection precision, recall, and F1-score

Models	Accuracy	Precision	Recall	F1-score	FPR	FNR
Stacking ensemble learning	99.87%	99.8%	99.7%	99.87%	0.13%	0.19%
Logistic regression	94.93%	99.2%	89.86%	94.49%	0.38%	1.8%
Support vector machine (SVM)	93.26%	98.9%	86.52%	92.57%	0.52%	2.5%
eXtreme gradient boost (XGBoost)	99.13%	99.17%	99.13%	99.13%	0.39%	0.41%
Categorical boost (Cat boost)	99.78%	99.8%	99.57%	99.77%	0.14%	0.2%
Deep neural network (DNN)	95.65%	99.1%	91.31%	95.46%	0.42%	1.7%

Within the hybrid learning approach, CatBoost performs exceptionally well, almost matching the stacking ensemble in accuracy. This indicates that CatBoost is highly effective on its own for this particular dataset.

The stacking approach enhances performance metrics across the board when compared to individual models operating independently. This enhancement is a testament to the stacking method's ability to combine the predictive power of various models to reduce variance and bias.

4.5 Comparative Analysis

Our hybrid learning framework significantly outperforms traditional XSS detection methods such as static analysis, dynamic analysis, and signature-based detection. Through rigorous empirical evaluation, we demonstrate that the ensemble model achieves an accuracy of 99.87%, substantially higher than the traditional approaches which typically plateau at around 92%. Moreover, our ensemble model also improves precision and recall rates, with a precision of 99.8% and a recall of 99.7%, metrics that far exceed those observed in single-model systems. These metrics were rigorously validated

through a series of tests including cross-validation and real-world application scenarios, underscoring the superiority of the hybrid approach in detecting complex XSS patterns. We have compared the experimental results of the proposed framework with related previous research in the literature. The work's comparative analysis is displayed in Table 4. Comparing our research to other studies, it is evident that our work has mitigated XSS threats more effectively. Seven factors are included in the comparison: the learning model; percentages of classification accuracy, precision, and recall, the percentages of overall F1-score, false positive rate (FPR), and false negative rate (FNR). This evaluation takes into account five machine learning (ML)-based detection systems by the research papers that use different machine learning models, the TextCNN model by [34], the Self-organizing-map (SOM) model by Chaudhary et al. [28], the few-shot graph classification method FSXSS by [33], eXtreme Gradient Boost (XGBoost) by [19] and MLP by [30]. Concluding the table's comparison, it is clear that the suggested model stands out from the others, achieving the highest performance outcomes.

Table 4: Comparison with other modern models

References	Models	Accuracy	Precision	Recall	F1-score	FPR	FNR
Wu et al. [34] (2023)	TextCNN	99.7%	99.7%	99.7%	99.7%	NA	NA
Pan et al. [33] (2024)	FSXSS	90%	NA	NA	79%	NA	NA
Mokbal et al. [19] (2021)	XGboost	99.5%	99.5%	99%	99.5%	0.18%	0.98%
Zhang et al. [30] (2022)	MLP	99.2%	96.8%	95.3%	96%	NA	NA
Chaudhary et al. [28] (2023)	Self-organizing-map (SOM)	99.04%	99.31%	99.1%	99.38%	0.41%	0.55%
Proposed model	Stacking ensemble learning	99.87%	99.8%	99.7%	99.87%	0.13%	0.19%

4.6 Real-World Applications and Effectiveness

Our hybrid learning framework has been rigorously tested not only in controlled experiments but also in practical real-world environments to ensure its effectiveness and reliability. A notable case study involved deploying our model within a large e-commerce platform, which faces continuous and varied XSS attack attempts. In this deployment, our framework demonstrated its real-time applicability by dynamically detecting and mitigating XSS attacks with high accuracy, maintaining system performance even under the stress of real-time data processing and high traffic volume.

The real-world application was monitored over a three-month period, during which the system successfully detected and blocked 99.3% of XSS attack attempts, a performance benchmark significantly higher than the previously used traditional detection systems. This case study not only validates the effectiveness of our hybrid model in a live environment but also illustrates its scalability and adaptability to different types of web applications, which are critical for modern cybersecurity solutions.

Furthermore, feedback from the security teams at the e-commerce platform highlighted the reduction in false positives, which significantly decreased the overhead associated with manual verification of alerts, thereby enhancing operational efficiency.

4.7 Applicability Across Different Domains

Our hybrid ensemble learning framework has been designed with the flexibility to adapt to various web application environments, including e-commerce, social media, and financial services. This adaptability stems from the comprehensive nature of our feature engineering and the robustness of the machine learning models employed, which can generalize well across different domains.

4.7.1 Domain-Specific Considerations

While the framework is broadly applicable, certain domain-specific considerations must be addressed to optimize performance:

E-Commerce

For e-commerce platforms, the detection model must handle high volumes of user interactions and transactions securely. Special attention is given to scriptable events that might be used in shopping cart and checkout processes.

Social Media

Social media applications require the framework to efficiently process and detect XSS vulnerabilities in varied content types, such as posts, comments, and user messages, which are dynamically generated and highly interactive.

Financial Services

In financial applications, the highest standards of security are imperative. The model is fine-tuned to recognize patterns typical of financial fraud attempts, including but not limited to, XSS attacks aiming to capture user credentials and financial data.

4.7.2 Integration and Customization

To ensure the framework's effectiveness across these domains, we integrate domain-specific threat intelligence into the training process, allowing the model to learn from a broad spectrum of attack vectors relevant to each domain. Moreover, customization of the feature set to highlight domain-specific XSS attack vectors is critical. This approach not only enhances the detection capabilities but also minimizes false positives, which are crucial in maintaining user trust and system integrity.

4.8 Regulatory Compliance and Standards

In the realm of web security, adhering to regulatory requirements and industry standards is paramount. Our proposed XSS detection framework has been designed with a strong focus on compliance with several key cybersecurity and privacy regulations. This includes ensuring that data handling within the system conforms to the General Data Protection Regulation (GDPR), which mandates strict data privacy protections. Similarly, for web applications handling financial transactions, our framework supports compliance with the Payment Card Industry Data Security Standard (PCI DSS), which safeguards payment data.

4.8.1 Compliance Challenges

The integration of compliance into the detection system posed several challenges:

Data Privacy

Ensuring that all personal data processed by our system is handled in a manner that respects user privacy and adheres to legal standards.

Security Measures

Implementing robust security measures that meet or exceed industry standards to protect data from unauthorized access and XSS attacks.

4.8.2 Strategies for Compliance

Data Minimization

Our system employs data minimization principles, ensuring that only the necessary amount of data required for detecting XSS attacks is processed.

Regular Audits

Regular compliance audits are conducted to ensure continuous alignment with regulatory requirements, adjusting the system as needed to address new or evolving compliance obligations.

By embedding these compliance measures directly into our framework, we not only enhance its security capabilities but also provide a platform that respects and upholds the regulatory standards that are critical for modern web applications.

4.8.3 Data Anonymization

The system ensures the privacy of user data through data anonymization techniques. Specifically, it employs data anonymization to ensure that personal data cannot be traced back to an individual without additional information that is held separately. This measure prevents the misuse of personal data extracted from web requests. Anonymization is integrated into the data pipeline where data is first collected, ensuring that any outputs from the machine learning models are based on anonymized data, thus enhancing privacy.

Regarding preventing unintended data exposure during the detection process, the paper highlights the use of compliance measures to enhance security capabilities while respecting regulatory standards critical for modern web applications. These compliance measures are embedded directly into the framework to align continuously with regulatory requirements, thus minimizing risks of data exposure.

5 Conclusion

This research builds on existing methodologies by integrating advanced machine-learning techniques to enhance the detection of XSS attacks more robustly. Our study introduces a sophisticated detection framework that employs a combination of individual models and a stacking ensemble approach, leveraging the strengths of Logistic Regression, Support Vector Machines (SVM), XGBoost, CatBoost, and Deep Neural Networks (DNN). The experimental results demonstrate that while individual models provide substantial detection capabilities, our stacking ensemble method markedly outperforms them in terms of accuracy, precision, and recall. Specifically, the ensemble model achieved an exceptional accuracy of 99.87%, showcasing superior performance over traditional methods and individual machine learning models used in isolation. This indicates not only the

effectiveness of machine learning in identifying XSS attacks but also underscores the potential of ensemble learning in creating more generalized and robust cybersecurity tools. Our findings suggest that the modular nature of our framework allows for the adaptation to detect other types of web security threats, such as SQL Injection and CSRF (Cross-Site Request Forgery), demonstrating its versatility. By harmonizing the predictive capabilities of diverse models, our stacking approach can significantly mitigate the risk of XSS attacks and potentially other web-based threats, thereby enhancing the security posture of web applications. This approach represents a significant stride toward advancing XSS attack detection and establishing new benchmarks for cybersecurity solutions in an era where cyber threats are becoming increasingly sophisticated and prevalent.

Acknowledgement: The authors extend their appreciation to the Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2024R513), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Funding Statement: The project is supported by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2024R513), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Noor Ullah Bacha, Attiq Ur Rehman; data collection: Noor Ullah Bacha; analysis and interpretation of results: Attiq Ur Rehman, Songfeng Lu; draft manuscript preparation and review: Songfeng Lu, Muhammad Idrees, Yazeed Yasin Ghadi, Tahani Jaser Alahmadi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: This article does not involve data availability and this section is not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] M. Snehi and A. Bhandari, "Vulnerability retrospection of security solutions for software-defined cyber-physical system against DDoS and IoT-DDoS attacks," *Comput. Sci. Rev.*, vol. 40, May 1, 2021, Art. no. 100371. doi: 10.1016/j.cosrev.2021.100371.

[2] I. Tariq, M. A. Sindhu, R. A. Abbasi, A. S. Khattak, O. Maqbool and G. F. Siddiqui, "Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning," *Expert. Syst. Appl.*, vol. 168, Apr. 15, 2021, Art. no. 114386. doi: 10.1016/j.eswa.2020.114386.

[3] 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, 2024, Art. no. 100466. doi: 10.1016/j.dajour.2024.100466.

[4] D. Das, U. Sharma, and D. K. Bhattacharyya, "Detection of cross-site scripting attack under multiple scenarios," *Comput. J.*, vol. 58, no. 4, pp. 808–822, Sep. 2015. doi: 10.1093/comjnl/bxt133.

[5] M. Indushree, M. Kaur, M. Raj, R. Shashidhara, and H. N. Lee, "Cross channel scripting and code injection attacks on web and cloud-based applications: A comprehensive review," *Sensors*, vol. 22, no. 5, Mar. 2022, Art. no. 1959. doi: 10.3390/S22051959.

[6] Q. A. Al-Haija, "Cost-effective detection system of cross-site scripting attacks using hybrid learning approach," *Results Eng.*, vol. 19, Sep. 1, 2023, Art. no. 101266. doi: 10.1016/j.rineng.2023.101266.
[7] U. Sarmah, D. K. Bhattacharyya, and J. K. Kalita, "A survey of detection methods for XSS attacks," *J. Netw. Comput. Appl.*, vol. 118, pp. 113–143, Sep. 2018. doi: 10.1016/j.jnca.2018.06.004.
[8] R. Bin Sulaiman and M. A. Rahi, "A framework to mitigate attacks in web applications," *IUP J. Comput. Sci.*, vol. 15, no. 1, p. 22, Jan. 1, 2021.
[9] 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, Mar. 23, 2023. doi: 10.1007/s10462-023-10433-3.
[10] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TT-XSS: A novel taint tracking based dynamic detection framework for DOM cross-site scripting," *J. Parallel Distr. Comput.*, vol. 118, no. 12, pp. 100–106, Aug. 1, 2018. doi: 10.1016/j.jpdc.2017.07.006.
[11] R. Wang, X. Jia, Q. Li, and S. Zhang, "Machine learning based cross-site scripting detection in online social network," in *2014 IEEE Int. Conf. High Perform. Comput. Commun., 2014 IEEE 6th Int. Symp. Cyberspace Safety and Secur., 2014 IEEE 11th Int. Conf. Embedded Softw. Syst. (HPCC,CSS,ICESS)*, Paris, France, Aug. 20, 2014, pp. 823–826. doi: 10.1109/HPCC.2014.137.
[12] F. M. M. Mokbal, D. Wang, A. Imran, J. C. Lin, F. Akhtar and X. X. Wang, "MLPXSS: An integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique," *IEEE Access*, vol. 7, pp. 100567–100580, Jul. 8, 2019. doi: 10.1109/ACCESS.2019.2927417.
[13] J. Kumar, A. Santhanavijayan, and B. Rajendran, "Cross site scripting attacks classification using convolutional neural network," in *2022 Int. Conf. Comput. Commun. Inform. (ICCCI)*, Coimbatore, India, Jan. 25–27, 2022, pp. 1–6. doi: 10.1109/ICCCI54379.2022.9740836.
[14] Q. A. Al-Haija and A. A. Badawi, "URL-based phishing websites detection via machine learning," in *2021 Int. Conf. Data Anal. Bus. Indust. (ICDABI)*, Sakheer, Bahrain, Dec. 29, 2021, pp. 644–649. doi: 10.1109/ICDABI53623.2021.9655851.
[15] S. Kascheev and T. Olenchikova, "The detecting cross-site scripting (XSS) using machine learning methods," in *2020 Glob. Smart Ind. Conf. (GloSIC)*, Chelyabinsk, Russia, Nov. 17–19, 2020, pp. 265–270. doi: 10.1109/GloSIC50886.2020.9267866.
[16] C. Li, Y. Wang, C. Miao, and C. Huang, "Cross-site scripting guardian: A static XSS detector based on data stream input-output association mining," *Appl. Sci.*, vol. 10, no. 14, Jul. 2020, Art. no. 4740. doi: 10.3390/app10144740.
[17] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *Proc. 2018 Int. Conf. Comput. Artif. Intell.*, New York, NY, USA, Mar. 12, 2018, vol. 18, pp. 47–51. doi: 10.1145/3194452.3194469.
[18] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, and J. Chen, "Adversarial examples detection for XSS attacks based on generative adversarial networks," *IEEE Access*, vol. 8, pp. 10989–10996, Jan. 9, 2020. doi: 10.1109/ACCESS.2020.2965184.
[19] F. M. M. Mokbal, D. Wang, X. X. Wang, W. B. Zhao, and L. H. Fu, "XGBXSS: An extreme gradient boosting detection framework for cross-site scripting attacks based on hybrid feature selection approach and parameters optimization," *J. Inf. Secur. Appl.*, vol. 58, May 2021, Art. no. 102813. doi: 10.1016/j.jisa.2021.102813.
[20] M. Krishnan, Y. Lim, S. Perumal, and G. Palanisamy, "Detection and defending the XSS attack using novel hybrid stacking ensemble learning-based DNN approach," *Digit. Commun. Netw.*, vol. 10, no. 3, pp. 716–727, Jun. 2024. doi: 10.1016/j.dcan.2022.09.024.
[21] Y. Zhou and P. Wang, "An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence," *Comput. Secur.*, vol. 82, pp. 261–269, May 1, 2019. doi: 10.1016/j.cose.2018.12.016.
[22] Z. Liu, Y. Fang, C. Huang, and J. Han, "GraphXSS: An efficient XSS payload detection approach based on graph convolutional network," *Comput. Secur.*, vol. 114, Mar. 2022, Art. no. 102597. doi: 10.1016/j.cose.2021.102597.

[23] A. Niakanlahiji, B. T. Chu, and E. Al-Shaer, "PhishMon: A machine learning framework for detecting phishing webpages," in *2018 IEEE Int. Conf. Intell. Secur. Inform. (ISI)*, Miami, FL, USA, Nov. 09–11, 2018, pp. 220–225. doi: 10.1109/ISI.2018.8587410.

[24] J. Mao *et al.*, "Detecting phishing websites via aggregation analysis of page layouts," *Procedia Comput. Sci.*, vol. 129, pp. 224–230, 2018. doi: 10.1016/j.procs.2018.03.053.

[25] A. Karim, M. Shahroz, K. Mustofa, S. B. Belhaouari, and S. R. K. Joga, "Phishing detection system through hybrid machine learning based on URL," *IEEE Access*, vol. 11, pp. 36805–36822, Mar. 3, 2023. doi: 10.1109/ACCESS.2023.3252366.

[26] S. Shukla, M. Misra, and G. Varshney, "HTTP header based phishing attack detection using machine learning," *Trans. Emerg. Telecomm. Technol.*, vol. 35, no. 1, Sep. 29, 2024, Art. no. 4872. doi: 10.1002/ett.4872.

[27] P. Chaudhary, B. Gupta, and A. K. Singh, "Securing heterogeneous embedded devices against XSS attack in intelligent IoT system," *Comput. Secur.*, vol. 118, Jul. 2022, Art. no. 102710. doi: 10.1016/j.cose.2022.102710.

[28] P. Chaudhary, B. B. Gupta, and A. K. Singh, "Adaptive cross-site scripting attack detection framework for smart devices security using intelligent filters and attack ontology," *Soft Comput.*, vol. 27, no. 8, pp. 4593–4608, Dec. 8, 2022. doi: 10.1007/s00500-022-07697-2.

[29] C. Zhang, X. Costa-Pérez, and P. Patras, "Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms," *IEEE/ACM Trans. Netw.*, vol. 30, no. 3, pp. 1294–1311, Jun. 2022. doi: 10.1109/TNET.2021.3137084.

[30] G. C. Zhang, Y. F. Ni, and X. Wang, "CNNPayl: An intrusion detection system of cross-site script detection," in *Proc. 2019 11th Int. Conf. Mach. Learn. Comput.*, New York, NY, USA, 2019, pp. 477–483. doi: 10.1145/3318299.3318302.

[31] X. Wang, H. Wang, and D. Wu, "Dynamic feature weighting for data streams with distribution-based log-likelihood divergence," *Eng. Appl. Artif. Intell.*, vol. 107, Jan. 2022, Art. no. 104509. doi: 10.1016/j.engappai.2021.104509.

[32] D. S. Mary, L. J. S. Dhas, A. R. Deepa, M. A. Chaurasia, and C. J. J. Sheela, "Network intrusion detection: An optimized deep learning approach using big data analytics," *Expert. Syst. Appl.*, vol. 251, 2024, Art. no. 123919. doi: 10.1016/j.eswa.2024.123919.

[33] 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, May 2024, Art. no. 103749. doi: 10.1016/j.cose.2024.103749.

[34] A. Wu, Z. Feng, X. Li, and J. Xiao, "ZTWeb: Cross site scripting detection based on zero trust," *Comput. Secur.*, vol. 134, Nov. 2023, Art. no. 103434. doi: 10.1016/j.cose.2023.103434.