information

MDPI

Article

Unveiling XSS Threats: A Bipartite Graph Approach with Ensemble Deep Learning for Enhanced Detection

Wafa Alorainy

Durma College of Science and Humanities, Shaqra University, Shaqra 11961, Saudi Arabia;
w.al3rini@hotmail.com or waloraini@su.edu.sa

Abstract: Cross-Site Scripting (XSS) attacks are a common source of vulnerability for web applications, necessitating scalable mechanisms for detection. In this work, a new method based on bipartite graph-based feature extraction and an ensemble learning classifier containing CNN, LSTM, and GRU is introduced. Our proposed bipartite graph model is novel as the payloads constitute the first set, while the words constructing the payloads comprise the second set. This representation allows structural and contextual dependencies to be extracted so the model can recognize complex and obfuscated XSS payloads. Our method surpasses state-of-the-art methods by having 99.97% detection accuracy. It has a significantly increased ability to detect complicated payload variations by utilizing co-occurrence patterns and interdependence between smaller payload parts through the adoption of these bipartite features. In addition to improving the F1-score, recall, and precision associated with such methods, it also demonstrates the adaptability of graph-based representation in cybersecurity applications. Our findings highlight the possibility of integrating ensemble classifiers and refined feature engineering into a scalable, precise XSS detection system.

Keywords: cross-site scripting attacks; bipartite graph; machine learning; deep learning; artificial neural networks; web vulnerabilities; cybersecurity; attack detection

check for updates

Academic Editor: Sokratis Katsikas

Received: 20 December 2024
Revised: 19 January 2025
Accepted: 26 January 2025
Published: 31 January 2025

Citation: Alorainy, W. Unveiling XSS Threats: A Bipartite Graph Approach with Ensemble Deep Learning for Enhanced Detection. *Information* **2025**, *16*, 97. https://doi.org/10.3390/info16020097

1. Introduction

In the contemporary digital landscape, web applications have become essential for delivering various services. However, this has led to a concurrent surge in cyberattacks. Among the most common cybersecurity vulnerabilities is Cross-Site Scripting (XSS), which presents significant risks not only to end users but also to service providers. Recently, machine learning and deep learning (ML/DL) techniques have been employed to detect XSS attacks. Online statistics indicate that approximately 68% of websites may harbor XSS vulnerabilities [1]. High-profile local and international companies, such as Facebook, Twitter, Baidu, and Sohu, have suffered substantial financial setbacks due to XSS-related attacks.

Code injection-based assaults, or XSS attacks, insert malicious scripts known as vulnerabilities that compromise trusted online apps and their associated plugins or hosting servers. Through the exploration of compromised web applications on users' browsers, attackers can gain elevated access privileges and expose private user data, including usernames and passwords. XSS attacks generally occur when proper control measures for user inputs in input forms are lacking. The Open Web Application Security Project (OWASP) has been reporting vulnerabilities since 2003, and according to the most recent report released in 2021, XSS attacks are among the top 10 vulnerabilities.

Current XSS detection methods frequently depend on ML models, static rules, or regular expressions and may not work well with highly obfuscated or unique payloads. Usually, these techniques fall short when capturing the structural connections between various payload components, a process that is essential for spotting minute patterns that indicate an assault. The ignorance of the structural and relational aspects of the payloads is a major drawback of current methods. For example, many detection models handle each token separately or do not consider the interdependence between various payload elements, such as the way HTML tags work with JavaScript functions or how special characters are concatenated to move beyond sanitization. These relational patterns are frequently the distinguishing characteristics of XSS payloads in sophisticated attacks intended to circumvent conventional detection methods. As the complexity of modern software systems has increased, XSS attacks have become more sophisticated, leading to an increasing number of newly identified vulnerabilities. Given that XSS payloads consist of different tokens (such as HTML tags, JavaScript keywords, symbols, etc.), their detection can be handled as a text analysis problem. However, the contextual and relationship structures of tokens in a payload are missed by typical text analysis techniques, which treat tokens as independent or sequential entities. Thus, a detection framework that handles the information about the relationships between tokens in XSS payloads is needed.

To address this gap, an approach that explicitly models the relationships between tokens within and across payloads is required. The complexity of disguised payloads, where the relationships between tokens may reveal more information than the tokens alone, may be managed with graph-based representations that outperform linear token analysis, improving detection accuracy. It has been demonstrated that graph-based techniques greatly increase the detection accuracy and have become a potent weapon in XSS detection. Graphs are excellent for simulating intricate dependencies and structural links by representing data as nodes and edges, which is often crucial for differentiating between malicious and benign payloads. Graph-based approaches have been used in XSS detection to examine the relationships between different elements, including HTML tags, JavaScript keywords, and special characters. Such approaches can identify context and patterns that conventional approaches miss [2–5]. However, current graph-based methods frequently do not pay enough attention to the relationship between individual payloads and their constituent tokens [6].

Therefore, in this study, bipartite graphs were leveraged to model XSS payloads for detection. In this approach, one set of nodes in the bipartite graph represents the payloads and the other represents the tokens (e.g., HTML tags, attributes, special characters) extracted from the payloads. The edges between the two sets encode the occurrence of tokens in a given payload. This representation captures the relationships between payloads and their constituent tokens, enabling the analysis of both token-level and payload-level patterns. By analyzing the structure of the bipartite graph, the system can identify anomalous relationships or frequently co-occurring patterns indicative of XSS payloads. The author tries to answer the following question: How effective is the bipartite graph representation for identifying obfuscated or novel XSS payloads compared to traditional text analysis methods? The ensemble classifier of three DL models—CNN, LSTM, and GRU—is also employed, which may enhance the detection process. The proposed models provide significant performance improvements compared to other existing state-of-the-art methods, as they can detect XSS-based attacks while simultaneously showing increased accuracy and precision values. The remainder of this paper is organized as follows: Section 2 presents different XSS attack detection methods; Sections 3 and 4 presents the dataset and the proposed research methodology; Sections 5 and 6 present the results and discussion, respectively; and Section 7 presents the conclusion.

Statement of Contribution

This work presents a novel method for detecting XSS in which bipartite graph modeling is used to capture the structural links between payloads and their constituent tokens. Our approach integrates structural insights from the bipartite graph with semantic features produced using Word2Vec and Doc2Vec embeddings. This is in contrast to other approaches that mainly concentrate on textual analysis or feature engineering. We use an ensemble learning architecture that combines the Convolutional Neural Network (CNN), LSTM, and GRU classifiers to improve the detection performance, thereby utilizing the advantages of recurrent and CNNs. Through thorough testing, we show that this integrated strategy performs better than current techniques in terms of its detection accuracy.

2. Literature

ML is a viable method for detecting XSS attacks in web applications. It has advantages over conventional techniques, including the capacity to learn from data and adjust to changing attack patterns [5]. One interesting field that is used for applying XSS attack detection techniques is supervised ML. In addition to boosting methods, researchers use Decision Trees (DTs) such as Random Forests (RFs), ADTree, and J48. Other popular supervised modes are Support Vector Machines (SVMs), Principal Component Analysis, kNN, mutual information, and Naïve Bayes.

Several studies have explored various ML algorithms for XSS detection. Kaur et al. [7] proposed an ML-based model that can recognize a malevolent attack path on a victim's device before the browser even parses it. They used the Linear Support Vector classification to detect cached and blind XSS attacks. During the feature generation phase, the authors pulled the scripts and JS events inserted by the hackers to execute attacks on the website. The experiment used a linear separable dataset, and the vulnerability web Mutillidae was used as the simulation platform. The detection accuracy was 95.4% with a recall value of 0.951 and a false positive rate of 0.111. Sharma et al. [8] focused on feature set extraction as a significant factor in identifying web-based assaults. This led the authors to propose a feature set extraction approach, which can significantly improve the results when applied with an ML-based intrusion detection model. They carried out an experiment using the Weka tool and CSIC HTTP 2010 dataset that consisted of three steps: (1) preprocessing of the dataset using a Python script before feeding it to the Weka for data modeling; (2) the extraction of features containing keywords such as GET, POST, DROP, DELETE, MODIFY, UNION DROP, etc., from the dataset; and (3) feeding of the data into the three ML models: Weka J48, OneR, and Naïve Bayes. J48 provided the best results among the classifiers. The effectiveness of J48 classifiers was also demonstrated by Alam and Pachauri [9], who employed the Weka tool to detect credit card payment fraud. J48 Decision Trees produce more correctly classified examples and require less time for model development.

For web threat detection, Yang et al. [10] created a convolutional gated-recurrent-unit (GRU) neural network for the detection of harmful URLs using characters as text classification features in order to increase the dependability and security of web applications by precisely identifying dangerous URLs. Given that dangerous keywords are exclusive to URLs, a feature representation approach to URLs based on malicious keywords was suggested. To acquire features on the time dimension, a GRU was utilized instead of the original pooling layer. The experimental findings demonstrate that the detection model, with an accuracy rate of over 99.6%, is very appropriate for high-precision classification tasks.

Research on OAuth vulnerabilities is also gaining traction. Munonye et al. [11] addressed OAuth vulnerabilities by performing a factor and Principal Component Analysis. They extracted the features most likely to influence the outcome. Various domain issues and OAuth workflow issues were identified using the Finite State Machine model. Features were extracted manually as there was no existing dataset available. The successful workflows were identified using the Gradient Boost Classifier (GBC), yielding an accuracy of 0.82 and an ROC curve value of 0.71. Wang et al. [12] proposed an approach that detects XSS worms on Online Social Network websites. Scripting functions on benign and malicious web pages cannot have the same frequency, as systems on these pages can be used for (punctuating) these pages. The researchers chose to use ADTree, as it has higher accuracy than other DTs and AdaBoost, and they decided to combine ADTree with AdaBoost. For classification, they used M1 (Adaptive Boosting) techniques, which act as a very potent classifier. Moreover, a feature extractor model was developed to learn the attributes from the web pages for automatic learning, since feature extraction and database generation play important roles in classification model generation. Benign and harmful samples were extracted from XXSed's Database and DMOZ. The researchers extracted four categories of characteristics from the web pages: (1) keyword features; (2) JavaScript features; (3) HTML tag features; and (4) URL features. They also identified sub-features within each category. The classification model was developed and evaluated with the Weka tool, and measurement was based on the tenfold cross-validation method. However, the proposed approach achieved a low detection rate with a high false-positive rate of 4.20%. The values were higher for M1, with a precision value of 0.941, recall of 0.939, and F-measure of 0.939 versus ADTree's precision value of 0.938, recall value of 0.936, and F-measure of 0.936.

Kascheev et al. [13] provided supervised ML approaches to detect XSS attacks using 30% and 70% of the test and training datasets generated based on the cross-validation method. To evaluate the model's performance, the following metrics were applied: F-measure, accuracy, completeness, and accurate responses. Their collection contains 200,000 lines of benign code and 40,000 pieces of malware. The more complex of the two libraries works with every request by first parsing it into Unicode characters and then splitting the query's arguments with regular expressions. However, searches that are deviant in many aspects are removed from the dataset. The original dataset was represented with the help of Word2Vec. Four ML techniques were compared: the SVM, DT, Logistic Regression (LR), and Naïve Bayes classifiers. The DT was the most effective. The metrics used were F-measure, recall, accuracy and precision. Banerjee et al. [14] explored the potential use of four ML algorithms (SVM, KNN, RF, and LR) in detecting cross-site scripting (XSS) attacks. They mapped True and False values in the dataset with the LR model. The experiment, carried out on a dataset with 24 attributes based on JavaScript and URL features, was performed with the Python and Scikit library. Of the four classifiers, the RF classifier was the most promising in terms of its low false positive rate and high accuracy.

The XSSClassifer was introduced in [15] to detect XSS attacks on social media platforms, e.g., Twitter and Facebook. An ML classifier was used for assault detection. Feature extraction was first performed with HTML text, SNSs, and URLs. During the evaluation phase, it was found that a higher accuracy occurred when the detection model employed the classifier after receiving SNS features. Out of the ten ML classifiers used to evaluate the detection model, tree classifiers such as the RF and ADTree yielded the best results with an accuracy of 97.2% and a false positive rate of 0.87 in real time. For XSS attack detection, Khan et al. [16] used four different ML classifiers, including the SVM, KNN, J48, and Naïve Bayes. The J48 achieved the highest accuracy (99.22%) when the dataset was divided into training and testing data. This work considered that the HTTP GET response passes to the server via the interceptor. If the JavaScript code is found to contain any malicious activity, the web page is disabled automatically before reaching the browser. XSS attacks are successfully detected in real time, no browser or platform is required. There is a low runtime overhead, and the model is lightweight. Features are extracted from the static analysis for the classifier's input. For the detection model, a yacc parser is used to turn the website's source code into a list of tokens. Lexical analysis is also performed to remove any code that has been obfuscated by hackers to obscure malicious behavior. As a component of security scanning, lexical analysis splits a source code into tokens before parsing. A study carried out by Alhamyani et al. [17] compared ensemble learning, RF, LR, SVMs, DTs, Extreme Gradient Boosting (XGBoost), Convolutional Neural Networks (CNNs), and Artificial Neural Networks (ANNs) with Multi-Layer Perceptron (MLP). To test the performance levels of the models, they were trained using a real-world dataset that was labeled as benign or malicious traffic. This was performed using the Information Gain (IG) and Analysis of Variance (ANOVA) feature selection approaches. The models were found to be very accurate: the accuracy level of the RF model was 99.78% and that of the ensemble models was over 99.64%. The results exceeded those of both previously proposed methods, showing that the proposed approach can efficiently protect web applications with reductions in FPs and FNs. Overall, this study presents an efficient, highly reliable, accurate ML-based system that significantly contributes to XSS detection.

Furthermore, several recent studies have introduced novel approaches to XSS attack detection that leverage advanced techniques such as graph-based models. Liu et al. [2] proposed a graph Convolutional Network-based Cross-Site Scripting payload detection model that can localize the payload in user-submitted content, which was referred to as the GraphXSS. They preprocessed the sample, put the processed data into a graph structure, and trained the Cross-Site Scripting detection model with the residual network and the Graph Convolutional Network (GCN). The AUC value of the GCN-based model in trials was 0.997 for small sample conditions. By adding the residual network structure to the detection model, the model stabilized and converged under the multi-layer and reached an accuracy rate of 0.996. Wang et al. [18] proposed the IGXSS (XSS payload detection model based on the inductive GCN), an XSS payload detection model based on inductive Graph Neural Networks. They considered the words and samples obtained through segmentation to be nodes and drew lines between them to form a graph. This allowed them to obtain a feature matrix of nodes and edges using only the information between nodes and not using other solutions such as pretrained word vectors. The obtained feature matrix was finally passed to a two-layer GCN to be trained and tested on multiple datasets with different sample distributions. Using real datasets, extensive experiments have shown that IGXSS outperforms other models across a range of sample distributions.

Tan et al. [1] proposed a detection technique based on a paths-attention-based sequence embedding model, named the PATS model, that is applied after sample-based construction at the model level for reflected XSS vulnerabilities. The model first converts vulnerability information into an intermediate representation of abstract syntax trees. It then enumerates all trees in the abstract syntax tree using syntactic routes, and these are transformed into vector representations via word embedding matrices. As it learns with Neural Networks, the model transforms passive defense into active defense by allocating correct weights arbitrarily to many sets of syntactic paths, where semantic features are extracted through attention processes to improve the training efficiency. Based on the experimental results, the paths-attention-based technique achieved an accuracy rate of 90.25% and an F1-score of 81.62% while halving the training time compared to conventional ML models to 30 min.

To improve the anti-phishing strategy, a hybrid classifier-based model using firm and flexible voting was proposed by Karim et al. [19]. A feature selection method that integrated the grid searching optimization and the canopy algorithm was employed. For reference, they achieved an accuracy level of 98.12% and an F1-score of 95.89%. They suggested a feature-rich, ML-based anti-phishing detection technique with an architecture supported by Shuk et al. [20]. When applied to the pretreatments used by the model to put the MM into a unique position, an accuracy value of 97.8% and an F1-score of 98.2% were achieved. However, the diversity of XSS load types was not considered, and the dataset was partitioned using Uniform Random Sampling. Bacha et al. [21] introduced a novel hybrid ensemble learning framework that utilizes various state-of-the-art ML algorithms such as Deep Neural Networks (DNNs), Extreme Gradient Boosting (XGBoost), SVMs, LR, and Categorical Boosting (CatBoost). By employing the XSS-Attacks-2021 dataset, which contains 460 samples of real-world traffic covering a wide range of materials, this system significantly enhances XSS attack detection. Beyond its boosting accuracy, the approach, which combines detailed feature crafting and model adjustment, unsurprisingly lowered the FP (0.13%) and FN (0.19%) rates and demonstrated a high accuracy rate of 99.87% during rigorous testing. Bakir et al. [22] introduced a new approach to XSS attack detection with the goal of providing efficient detection by using the strengths of Word2Vec embeddings as the feature extractor along with the Universal Sentence Encoder (USE), which can greatly improve the performance of ML and DL techniques. Using word-level representations from Word2Vec and the semantic understanding of sentences from USE, a comprehensive feature representation for XSS attack payloads was obtained. Their proposed technique examines both intricate word meanings for extreme feature extraction and more general phrase contexts for enhanced model performance. The obtained results demonstrate that their mixed embeddings method performs better than traditional XSS attack detection methods in terms of its recall, accuracy, precision, ROC, and F1-score. Table 1 summarizes previous studies, showing the study, the model used, and the performance level.

Table 1. Summary of Literature on Machine Learning (ML) for Cross-Site Scripting (XSS) Detection.

Study	Model	Performance
Kaur et al. [7]	Linear SVM	Accuracy: 95.4%, Recall: 0.951, FPR: 0.111
Sharma et al. [8]	J48, OneR, Naïve Bayes	Best results with J48
Munonye et al. [11]	Gradient Boosting Classifier (GBC)	Accuracy: 0.82, ROC: 0.71
Wang et al. [12]	ADTree + AdaBoost	Precision: 0.941, Recall: 0.939, F-Measure: 0.939
Yang et al. [10]	GRU	Accuracy: 0.996
Kascheev et al. [13]	SVM, Decision Tree, Logistic Regression, Naïve Bayes	Best results with Decision Tree
Banerjee et al. [14]	SVM, KNN, Random Forest, Logistic Regression	Random Forest: Low FPR, Good accuracy
Rathore et al. [15]	Tree Classifiers (Random Forest, ADTree)	Accuracy: 97.2%, FPR: 0.87
Khan et al. [16]	SVM, KNN, J48, Naïve Bayes	J48: Accuracy: 99.22%
Alhamyani et al. [17]	Random Forest, SVM, XGBoost, CNN, MLP	Random Forest: Accuracy: 99.78%
Liu et al. [2]	Graph Convolutional Network (GCN)	Accuracy: 0.996
Wang et al. [18]	Inductive GCN (IGXSS)	Outperforms other models
Tan et al. [1]	Paths-Attention (PATS)	Accuracy: 90.25%, F1-Score: 81.62%
Karim et al. [19]	Hybrid Classifier	Accuracy: 98.12%, F1-Score: 95.89%
Bacha et al. [21]	Hybrid Ensemble (DNN, XGBoost, SVM, etc.)	Accuracy: 99.87%, FP: 0.13%, FN: 0.19%
Bakir et al. [22]	Word2Vec + USE Embeddings	Improved performance (Accuracy, Recall, F1-score)

These methods experience overfitting problems for the XSS load type with few labeled samples. To alleviate this problem, this work contributes to the body of research on XSS attack detection. Many researchers have developed models based on state-of-the-art ML methods. Hence, work in the field of XSS attack detection has matured. These models show great accuracy and recall scores on some datasets, ranging from text embedding and combinations with DL methods to complex architectures that mix GCNs and residual networks. However, such approaches often suffer from disadvantages, such as bad generalization, reduced FP rates, and poor detection of complex attack patterns. In addition, the latest survey research [23–25] highlights the need for innovative feature extraction and models that detect more complex data. To address these challenges, this research introduces a novel feature set that extracts valuable information from XSS payloads. Moreover, we propose a lightweight attack detection model powered by an advanced DLarchitecture, with the aim of enhancing the effectiveness and efficiency of XSS vulnerability detection.

3. Dataset

This investigation utilized the XSS DL dataset, which was published on the Kaggle repository by SYED SAQLAIN HUSSAIN SHAH. This is a comprehensive dataset that was obtained from the OWASP and PortSwigger XSS attack cheat sheets, containing a total of 13,685 entries. It was employed for our tests and assessments of XSS attack detection. It served as a comprehensive web content collection, encompassing both XSS attack examples and secure samples. We included a diverse range of XSS attack vectors to assess the performance of our models in multiple attack scenarios. Using this dataset, we were able to perform extensive testing and capture informative results regarding the efficiency and efficacy of our XSS attack detection methods. Figure 1 shows the distribution of the dataset.

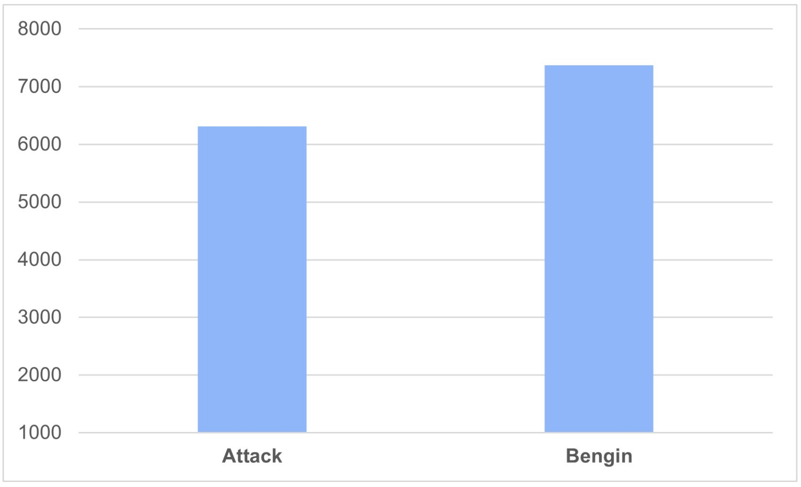

Figure 1. The distribution of the XSS dataset.

4. Methods

4.1. The Proposed Model

The four primary processes in the suggested model are preprocessing, bipartite creation and projection, embedding extraction, and classification using the ensemble learner. Figure 2 provides a general illustration of our model. Further information on the methods used can be found in the following subsections.

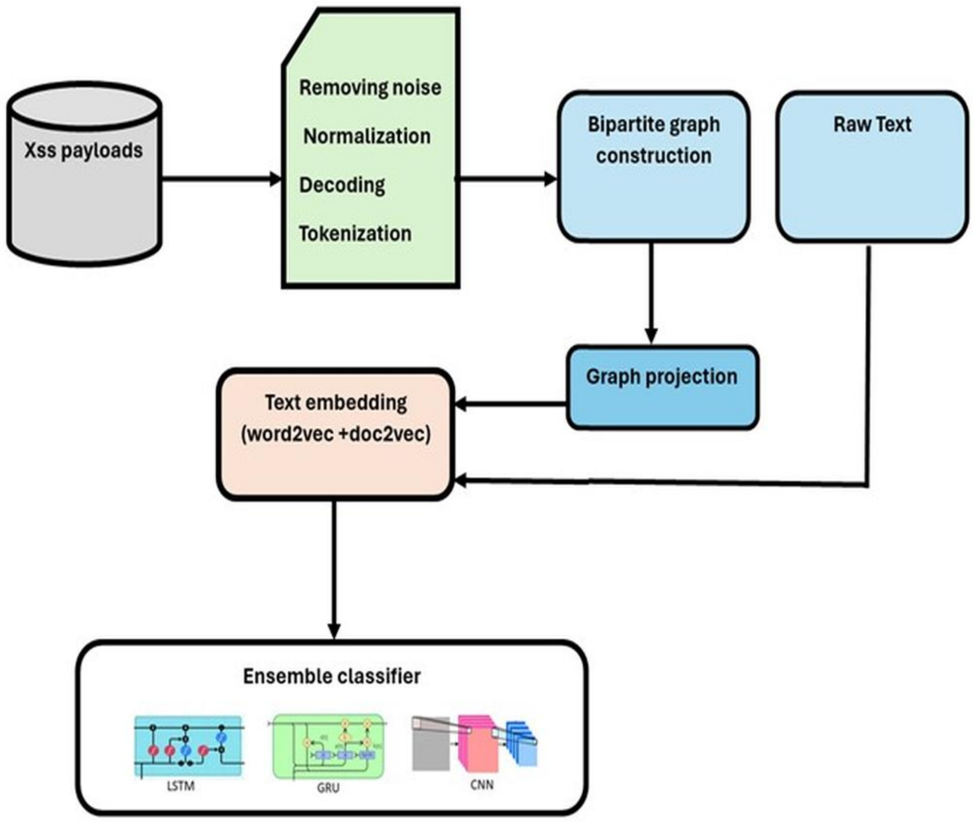

Figure 2. An overview of the proposed model.

4.2. Pre-Processing

4.2.1. Removing Noisy Data/Normalization

In order to confuse the detection models and ultimately lead them to categorize such vectors as innocuous scripts, the attacker purposefully added noise data. An example of noise data is when an attacker uses a newline symbol to split the attack vector into multiple lines. They can also use many other disturbing symbols, such as closing tags >, starting tags <, **, /, ", ', uppercasing the letters, or using numbers, many URI links, or IP addresses to circumvent the detection system and carry out the attack once the browsers' parsers have executed it. It is important to remember that this step might be used following the decoding procedure. This is due to the possibility of the attacker adding noisy data to the payload prior to the encoding process, which would make them more difficult to identify.

4.2.2. Decoding

The most popular technique used by attackers to produce evolved or modified XSS attacks is encoding. Since the browser automatically decodes and escapes the scripts to be parsed, this mechanism gives the attacker the ability to avoid detection by traditional systems such as the Web Application Firewall (WAF). The attacker uses this feature to effectively conceal and carry out evolved XSS attacks. Because the payload is converted back into its original context form during the decoding phase, XSS attacks can be detected with more precision. Decoding methods were implemented using URL decoding, HTML entity decoding, Unicode decoding, and Base64 decoding. An example of XSS decoding is as follows:

```
%3Cscript%3Ealert('XSS')%\C/script%3E → <script>alert('XSS')</script>
```

4.2.3. Tokenization

In XSS detection, tokenization is used to break down the payloads into smaller units for analysis when working with payloads with no spacing. This is because payloads with no spacing can be difficult to analyze as a whole, and tokenizing them allows for a more detailed analysis of the individual words or phrases that make up the payload. In this study, the word level was applied. An example of the tokenization process is as follows:

```
Example Tokens:
<script>, alert, ('XSS'), </script>
```

The decoding and tokenization procedure is illustrated in Figure 3.

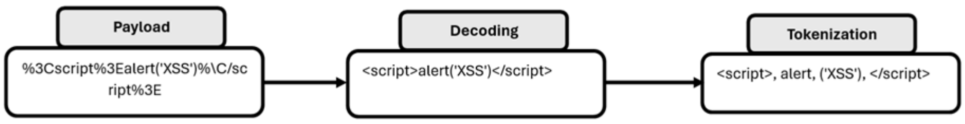
Payload
%3Cscript%3Ealert('XSS')%\C/script%3E
Decoding
<script>alert('XSS')</script>
Tokenization
<script>, alert, ('XSS'), </script>

Figure 3. Decoding and tokenization procedure.

4.3. Feature Extraction

4.3.1. Graph Construction

To construct the graph nodes, we used the bipartite model. In the field of graph theory, a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets, U and V, that is, every edge connects a vertex in U to one in V. Vertex sets U and V are usually called the parts of the graph. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles. In generic networks (one mode, simple network), node one is connected to node two by a tie that represents the relationship (e.g., token/token). Mathematically, we let $P = \{p_1, p_2, \ldots, p_m\}$ represent the set of XSS payloads and $W = \{w_1, w_2, \ldots, w_n\}$ represent the unique words extracted from these payloads. A bipartite graph $G = (P \cup W, E)$ is constructed, where $E \subseteq P \times W$ represents the relationships between the payloads and their constituent words.

An edge $(p_i, w_j) \in E$ exists if and only if $w_j \in p_i$. The adjacency matrix A of the bipartite graph is defined as

$$A_{ij} = \begin{cases} 1 & \text{if } w_j \in p_i, \\ 0 & \text{otherwise.} \end{cases}$$

In a bipartite network, node one is connected to node two by an affiliation relationship, e.g., node A and node B belong to the same payload. Two sets of nodes are defined: In sentence nodes (payload nodes), each sentence in the dataset represents a node in one set. In word nodes, each unique word or 'token' across the dataset represents a node in the second set. In the second set, the edges connect each sentence node to the word nodes that are present in the sentence. These connections create edges in the bipartite graph and represent the relationship between sentences and words. The steps of constructing the bipartite network are explained in Algorithm 1.

Algorithm 1 Constructing a Bipartite Graph from XXS Payloads

```
1. Input: Set of XXS payloads $P = \{p_1, p_2, \ldots, p_n\}$, set of tokens $T = \{t_1, t_2, \ldots, t_m\}$
2. Output: Bipartite graph $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of edges.
3. Initialize:
   (a) $V \leftarrow \{P \cup T\}$ ▷ Initialize the set of nodes as the union of payloads and tokens
   (b) $E \leftarrow \{\}$ ▷ Initialize the set of edges as empty
4. For each payload $p_i \in P$:
   (a) For each token $t_j \in T$:
       i. If token $t_j$ appears in payload $p_i$, add an edge:
              $E \leftarrow E \cup \{(p_i, t_j)\}$
          ▷ Connect payload $p_i$ to token $t_j$
5. Optional: Add additional edges based on other relationships between payloads:
   (a) For each pair of payloads $p_i, p_j \in P$, if they share common tokens:
           $E \leftarrow E \cup \{(p_i, p_j)\}$
       ▷ Connect payloads that share common tokens, if needed
6. Return: Bipartite graph $G = (V, E)$ ▷ Return the constructed bipartite graph
```

4.3.2. Bipartite Graph Projection

For the convenience of directly showing the relations among a particular set of nodes, the bipartite network is typically compressed by one-mode projecting. The one-mode projection onto the X projection indicates a network containing only X nodes, where two X nodes are connected when they have at least one common neighboring Y node.

A bipartite graph's projection stage is essential because it reduces a complicated two-layer structure to a single-layer graph that emphasizes relevant relationships, making subsequent tasks (such as classification and embeddings) more effective and understandable.

To illustrate, we let $G = (P \cup W, E)$ represent a bipartite graph, where P is the set of XSS payloads, W is the set of words, and $E \subseteq P \times W$ is the edge set.

Adjacency Matrix of the Bipartite Graph

The adjacency matrix A of G is defined as

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between } p_i \text{ and } w_j, \\ 0 & \text{otherwise.} \end{cases}$$

Projection onto Payloads (P)

The projection of G onto the set P is a graph $G_P = (P, E_P)$, where

$$E_P = \{(p_i, p_k) : \exists w_j \in W \text{ such that } (p_i, w_j) \in E \text{ and } (p_k, w_j) \in E\}.$$

The adjacency matrix for the projection onto P is given by

$$A_P = A \cdot A^T,$$

where A^T is the transpose of A.

Projection onto Words (W)

Similarly, the projection of G onto the set W is a graph $G_W = (W, E_W)$, where

$$E_W = \{(w_j, w_l) : \exists p_i \in P \text{ such that } (p_i, w_j) \in E \text{ and } (p_i, w_l) \in E\}.$$

The adjacency matrix for the projection onto W is given by

$$A_W = A^T \cdot A.$$

Figure 4 shows the resulting networks of the X and Y projections. The simplest method is to project the bipartite network onto an unweighted network, as we are not considering the frequency with which a collaboration is repeated. After the projection, we obtain the tokens following a new relationship, which is an affiliation relationship. Assuming that the payloads' phrases are Y and their words are X, the Ys is discarded.

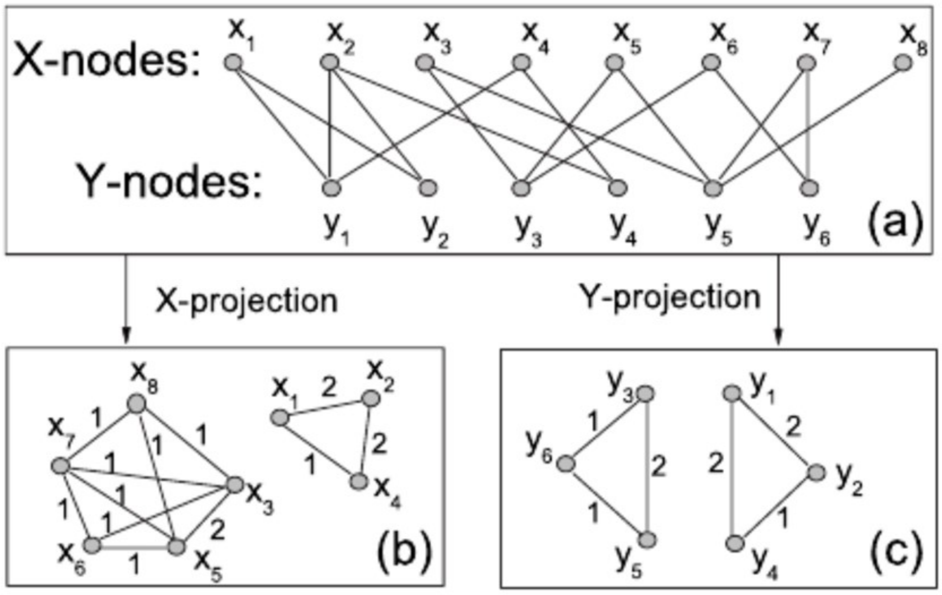

Figure 4. Illustration of a bipartite network (**a**) and its X projection (**b**) and Y projection (**c**). In this example, the Ys are the XSS payloads, and the Xs are the tokens that belong to the payloads. This Figure was produced in [26].

We used the igraph package v1.3.0 for the bipartite graph generation and analysis because of its effectiveness with large-scale graph topologies. In addition, the following hardware setup was used to guarantee scalability and effective computing for the graph with approximately 100,000 payloads and the words that correspond to them: Processor, 2.50 GHz 12th Gen Intel® Core™ i9-12900H; memory, no less than 32 GB of RAM to avoid memory bottlenecks and support huge graph structures; and operating System, Ubuntu 20.04.

4.3.3. Embedding Extraction

Two embedding techniques were used to capture the text's syntactic and semantic relationships:

Word2Vec:

Individual words were transformed into dense vector representations with fixed dimensionality using a pretrained Word2Vec model. Better feature representation for sequential models was made possible by these embeddings' retention of the contextual meaning. To guarantee the availability of relevant word embeddings, we employed a Word2Vec model that was pretrained on a sizable corpus of text data in our implementation. We used the 'XSS dataset' (https://github.com/fawaz2015/XSS-dataset?utm_source=chatgpt.com (accessed on 27 December 2024) created by Fawaz et al. [27] to train the Word2Vec model. This dataset is made up of 138,569 records that were carefully selected for AI-based XSS detection, containing both malicious and benign examples. To provide a varied web content depiction, the data were gathered using a powerful Python scraping framework that used a revolutionary random walk and random jumping technique. Of the 150,000 crawled pages in the dataset, 100,000 were chosen at random, and 38,569 of the malicious samples came from sites such as XSSed and Open Bug Bounty. A complete collection of 167 characteristics was obtained by employing dynamic feature extraction techniques. This dataset offers a diverse and comprehensive corpus of text that can be used to train algorithms that are intended to identify XSS vulnerabilities.

Words with comparable meanings were placed closer to one another in the vector space after the model transformed each word into a fixed-size vector of 100. The text was tokenized into individual words, and each word's presence in the pretrained Word2Vec model's vocabulary was examined in order to obtain word embeddings for the XSS attack payload. The relevant word embedding was extracted if a word was found. Words that were absent from the model's vocabulary were either left out or had zero vectors initialized in their embeddings. In order to help the model to identify important patterns and contextual information, Word2Vec was used to capture the semantic meanings of words within the XSS attack payload. The sentence-level (Doc2vec) semantic understanding was enhanced by these word-level embeddings, which were a crucial part of the feature representation. By combining these two methods, a thorough and potent feature representation was produced, facilitating the efficient detection of XSS threats. Here, an explanation of Doc2vec is presented.

Doc2Vec:

Doc2Vec was used to create sentence-level or document-level embeddings in addition to word embeddings to offer a high-level semantic summary of the entire text. Algorithm 2 explains the process.

The following parameters were derived from earlier research [28–31] and are summarized in Table 2.

Table 2. Common Settings for Embedding Methods in XSS Detection Research.

Parameter	Word2Vec	Doc2Vec
Vector Size	200 (common range: 100–300)	200 (common range: 100–300)
Window Size	5 (common range: 3–10)	10 (common range: 5–15)
Min Count	1 (to capture rare XSS-related words)	1 (to capture rare XSS patterns)
Algorithm	Skip-Gram	PV-DM (Distributed Memory)
Training	Negative Sampling (5–10 samples)	Negative Sampling (5–10 samples)
Epochs	20 (range: 10–50)	20 (range: 10-50)
Learning Rate	0.025 (range: 0.01–0.05)	0.025 (range: 0.01–0.05)
Workers	4–8 threads	4–8 threads

Algorithm 2 Extracting Features from the Bipartite Graph with Concatenation

```
1. Input: Bipartite graph $G$, Pre-trained Word2Vec model $W2V$, Doc2Vec model $D2V$
2. Output: Feature matrix $F$
3. Initialize: $F \leftarrow \{\}$    ▷ Initialize feature matrix
4. For each payload node $payload \in$ GETPAYLOADNODES($G$):
   (a) $tokens \leftarrow$ GETCONNECTEDTOKENS($G, payload$)    ▷ Tokens linked to this payload
   (b) $token\_vectors \leftarrow \{\}$    ▷ Initialize the list to store Word2Vec embeddings for tokens
   (c) For each token $token \in tokens$:
       i. If $token \in W2V$:
          A. Append $W2V[token]$ to $token\_vectors$
       ii. Else:
          A. Append $ZeroVector$ to $token\_vectors$    ▷ Fallback for unknown tokens
   (d) $payload\_vector \leftarrow D2V[payload]$ ▷ Get Doc2Vec representation of the payload
   (e) $graph\_features \leftarrow$ COMPUTEGRAPHFEATURES($G, payload$)    ▷ Compute graph-based features, e.g., degree, edge weights, etc.
   (f) $token\_embedding \leftarrow$ MEAN($token\_vectors$)    ▷ Compute the average of Word2Vec embeddings of tokens
   (g) $combined\_features \leftarrow$ CONCATENATE($payload\_vector, token\_embedding$)    ▷ Concatenate all features into one final feature vector for the payload
   (h) Append $(payload, combined\_features)$ to $F$    ▷ Store the final feature vector for the payload
5. Return: $F$    ▷ Return the feature matrix
```

4.4. Classification

4.4.1. Machine Learning

ML techniques are essential for the categorization methods used for XSS attack detection. These methods make it possible to create predictive algorithms that can accurately and automatically categorize web material as either benign or possibly harmful. Labeled training data are used by ML algorithms to identify patterns and relationships that can be used to improve categorization. To categorize and identify XSS attacks, we investigated a number of ML methods, such as LR, SVMs, RFs, DTs, Extreme Gradient Boosting (XGBoost), and MLP. Word2vec and Doc2vec feature vectors were used to train the aforementioned algorithms. Classical ML algorithms are used more often than other learning techniques such as DL and ensemble learning, possibly because ML algorithms are often simpler to deploy and develop than other methods [5]. A variety of classifiers, such as DT, RF, SVM, KNN, and LR, were used by Thajeel et al. [5] to evaluate the feature selection decisions made by a multi-agent reinforcement learning-based dynamic feature selection model intended for XSS attack detection. The DT classifier performed better than the other models evaluated. Moreover, XGBoost is widely utilized for both classification and regression problems and is a reliable approach. XGBoost was used as an XSS detection framework by Rozi et al. [32]. Additionally, Mokbal et al. [27] introduced a web-based framework for XSS attack detection utilizing XGBoost alongside an extreme parameter optimization strategy that employs the grid-search technique. The classifiers examined in this study were thoroughly tested for XSS detection purposes, prompting us to evaluate them using our proposed feature set. The hyperparameters associated with the selected ML algorithms are detailed in Table 3. Most of these parameters were used by Bakir et al. [22].

Table 3. ML Classifier Hyperparameter Settings.

Classifier	Hyperparameter Setting
RF (Random Forest)	• n_estimators: 100 • max_depth: None • min_samples_split: 2 • min_samples_leaf: 1 • max_features: 'auto'
DT (Decision Tree)	• max_depth: None • min_samples_split: 2 • min_samples_leaf: 1
MLP (Multi-Layer Perceptron)	• hidden_layer_sizes: 100 • activation: relu • solver: adam • alpha: 0.0001
SVM (Support Vector Machine)	• gamma: scale • degree: 3
LR (Logistic Regression)	• penalty: l2 • C: 1.0 • solver: lbfgs • max_iter: 100
XGBoost	• n_estimators: 100 • max_depth: 6 • learning_rate: 0.1 • subsample: 0.8 • colsample_bytree: 0.8
KNN (K-Nearest Neighbors)	• n_neighbors: 5 • weights: 'uniform' • algorithm: 'auto' • leaf_size: 30 • metric: 'minkowski' • p: 2

4.4.2. DeepLearning

Because DL techniques can learn complex structures and representations from data, they have become effective tools for XSS attack detection. These deep Neural Network-based methods can automatically extract hierarchical features and capture complex relationships in web material. CNNs and Recurrent Neural Networks (RNNs) are two basic DL architectures that were used in our study to categorize online content and identify XSS attacks. Feature representations from the Word2vec, and Doc2vec models were used to train these architectures. We improved the model's ability to recognize minor signs of XSS attacks by utilizing DL, which improved the reliability and accuracy of detection. DL models can defend against XSS attacks and strengthen web application security because of their capacity to learn from vast amounts of data and identify complex patterns. We aimed to highlight the XSS attack detection speed by using straightforward DL architectures and reduce the computational steps required to detect and stop XSS attacks without sacrificing performance by choosing simpler architectures. These architectures were created to quickly identify XSS assaults in real-time scenarios by striking a balance between the accuracy and computing efficiency [22]. In this study, we used CNN, LSTM, and GRU for classification purposes.

CNN is an artificial neural network (ANN) that uses a convolutional kernel to slide over input characteristics. Convolutional, pooling, and fully linked layers make up CNNs. The pooling layer minimizes the size of convolved data while preserving significant features, whereas the convolutional layer computes various feature maps. CNNs introduce the nonlinearity needed to handle nonlinearly separable problems/features using a variety of activation functions, including ReLu, Maxout, Tanh, and Sigmoid. The fully connected layer uses the softmax function to classify the target classes after determining the associations between features [33]. According to Dong et al. [34], CNNs perform well in computer vision and intrusion detection systems, lowering the computational complexity and accelerating training and prediction times through the extraction of local features. By suggesting CNN models with various network architectures and hyperparameters, numerous studies have successfully used CNNs in the security area to identify XSS threats. The CNN model is the most well-known and widely used algorithm for XSS attack detection [35–38]. According to Hochreiter et al. [39], LSTM is an RNN that can learn and train long-range temporal dynamics in sequences of any length. By employing a gate function that selectively permits the flow through of a subset of the data, LSTM was developed to address gradient vanishing in conventional RNNs [40]. Before sending the long-term and short-term information to the next cell, the input, output, and forget gates in the LSTM model serve as filters to determine which information should be kept or discarded. Unwanted and unnecessary information can be eliminated by the gates. For classification and prediction using time-series data, LSTM has emerged as one of the most popular RNN variations.

Web application and XSS threats have been identified using the GRU. A CNN-GRU model was used by Niu and L as a combination technique for web attack detection. The GRU captures the serialization relationship of the context in the web event sentence. A detection approach was proposed by W. Yang et al. [10] in which a CNN is used to extract features from the abstract level of the URL, and a GRU is used as a pooling layer to retain the important characteristics while preserving the context correlation. Each level's characteristics are fully extracted using a mixture of convolution windows of varying lengths. To make the most of the extracted features, the features obtained from the convolution windows must be integrated.

Three convolutional layers make up the CNN model. Each layer has 128, 64, and 32 filters with a relu activation function, accordingly. A max pooling layer comes after each convolutional layer. After that, a flatten layer and a fully connected layer are applied. An output layer that makes use of the sigmoid activation function rounds out the model. The LSTM model is made up of an output layer with the 123 sigmoid activation function after a single LSTM layer with 20 units and the relu activation function. This is part of our quest to obtain a streamlined model with optimal performance to allow quick detection. Similarly, the GRU models were composed of an output layer with the sigmoid activation function after a single GRU layer with 20 units and a relu activation function. Every model was trained with a batch size of 128 over 100 epochs. A uniform feature representation was created by concatenating the outputs of the CNN, GRU, and LSTM models. The strengths of each model were merged in this representation: CNN was used for local feature extraction, and LSTM and GRU were used for sequential dependency modeling. The final prediction was mapped from the combined features using a dense layer and a sigmoid activation function. Optimal feature fusion was ensured by training the architecture end to end. This approach can be considered an integrated ensemble where models are not independently trained but collaborate to form a unified network [41,42]. The following Algorithm 3 explains the ensemble classifier:

Algorithm 3 Integrated Ensemble Approach

```
1: Input: Text data as sequences of word indices (word_sequences).
2: Output: Trained ensemble model for classification.
3: Step 1: Shared Embedding Layer
4: Convert word_sequences into dense vector representations using a shared embedding
   layer (embedding_layer).
5: Step 2: Individual Neural Networks
6: Pass embedding_layer to:
     • LSTM: Extract sequential features (lstm_features).
     • GRU: Extract sequential features (gru_features).
     • CNN:
         – Apply 1D convolution with activation (ReLU).
         – Use Global Max Pooling to extract features (cnn_features).
7: Step 3: Feature-Level Fusion
8: Concatenate lstm_features, gru_features, and cnn_features into a single feature vector
   (combined_features).
9: Step 4: Classification Layer
10: Pass combined_features through:
     • A dense layer with activation (ReLU).
     • A final dense layer with sigmoid activation to generate predictions (output).
11: Step 5: Model Compilation and Training
12: Define loss function (e.g., binary_crossentropy).
13: Use an optimizer (e.g., Adam) to minimize the loss.
14: Train the model using word_sequences and corresponding labels (train_labels).
```

The parameters used for DL classifiers are shown in Table 4. Note that these parameters were used by Bakir et al. [22].

Table 4. DL Classifier Hyperparameter Settings.

Classifier	Hyperparameter Setting
GRU (Gated Recurrent Unit)	• Loss: binary crossentropy • Optimizer: adam • Activation function: Sigmoid • Learning rate: 0.001 • Epoch: 100 • Batch_size: 128
LSTM (Long Short-Term Memory)	• Loss: binary crossentropy • Optimizer: adam • Activation function: Sigmoid • Learning rate: 0.001 • Epoch: 100 • Batch_size: 128
CNN (Convolutional Neural Network)	• Filters in Convolution Layers: 128, 64, 32 • kernel_size: 4 • padding: same • Activation functions: Relu, Relu, Relu, Sigmoid • Learning rate: 0.001 • Epoch: 100 • Loss: binary crossentropy • Optimizer: adam • Batch_size: 128

4.5. Optmization

The models were trained using the Adam optimizer with a learning rate of 0.001, and binary crossentropy was used as the loss function for classification tasks. To reduce overfitting, dropout regularization and early halting were used. With 20% set aside for validation, the training dataset was divided into training and validation sets.

4.6. Experiment Evaluation

A set of thorough evaluation metrics were used to evaluate the efficacy and performance of our XSS attack detection models based on their accuracy, precision, recall, and F1-scores, giving us a comprehensive picture of their performance. To identify any biases or imbalances in performance, the confusion matrix was also used to display the distribution of TP, TF, FP, and FN predictions.

Some typical evaluation metrics are presented. Accuracy: this widely used indicator determines the percentage of accurate predictions made by the model, as shown in Equation (1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Precision (sensitivity): the true positive rate of prediction, which is the proportion of accurately recognized real positives, as shown in Equation (2).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall (specificity): the proportion of positive examples accurately expected to be positive is measured by the recall, as shown in Equation (3).

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F1 score: a harmonic-based mean of recall and precision, as shown in Equation (4).

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4}$$

We employed a Repeated Stratified K-Fold Cross-Validation technique to strengthen the validation even more. In order to achieve this, the data had to be divided into 10 folds, each of which was utilized as a training set, with one fold used as a validation set. To avoid biases related to the ordering of data points, this process was performed three times with a random data shuffle in between each cycle [21].

5. Results

Here, the performance features of various models used to detect XSS attacks are reviewed, with a focus on certain statistical metrics, including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of each model's ability to identify and classify XSS attacks, which is crucial for robust cybersecurity measures. The analysis is based on performance data from the deployed models, including LR, SVM, RF, DT, MLP, KNN, XGBoost, LR, and a Deep Neural Network (CNN, LSTM, and GRU models), as well as the integrated ensemble technique.

5.1. Evaluation of Different Models

In the process of determining the most effective ML model for detecting XSS vulnerabilities, we conducted a comparative analysis using several well-known algorithms. Table 5 summarizes the performance of each model across multiple metrics to assess their effectiveness in real-world scenarios. The purpose of this step was to identify the most effective model for our proposed futureset. Among the ML and DL models, the ensemble learner had the highest accuracy and F-score (99.9% for both). The LSTM model also worked effectively with an accuracy of 99.2% and F-score of 98.7%. However, coordination between the CNN, LSTM, and GRU (ensemble) models produced a better XSS detecton rate. For some models, such as the RF and XGBoost model, the evaluation metrics for the proposed feature dataset showcased a robust performance, with an accuracy of 97.62%, precision of 98.3%, recall of 69.7%, and an F1-score of 97.5%. However, the ensemble classifiers showed an improved detection rate compared to using the ML and DL models alone. Figure 5 shows the differences between the individual models and the ensemble classifier. These results are discussed in the Discussion section (Section 6).

Table 5. Performance Metrics for Different Models (Presented as Ratios).

Model	**Proposed Feature (Raw + Bipartite)**			
	Accuracy	**Precision**	**Recall**	**F-Score**
SVM	0.946	0.927	0.942	0.957
RF	0.967	0.967	0.984	0.965
DT	0.943	0.92	0.958	0.939
MLP	0.96	0.947	0.966	0.956
KNN	0.90	0.90	0.90	0.90
XGBoost	0.966	0.952	0.975	0.963
LR	0.935	0.916	0.944	0.93
CNN	0.982	0.975	0.985	0.98
LSTM	0.992	0.987	0.993	0.988
GRU	0.972	0.967	0.973	0.97
CNN-GRU	0.975	0.979	0.971	0.975
CNN-LSTM	0.984	0.989	0.981	0.984
LSTM-GRU	0.979	0.983	0.977	0.979
Ensemble (CNN, LSTM, GRU)	**0.9997**	**0.9995**	**0.9998**	**0.9997**

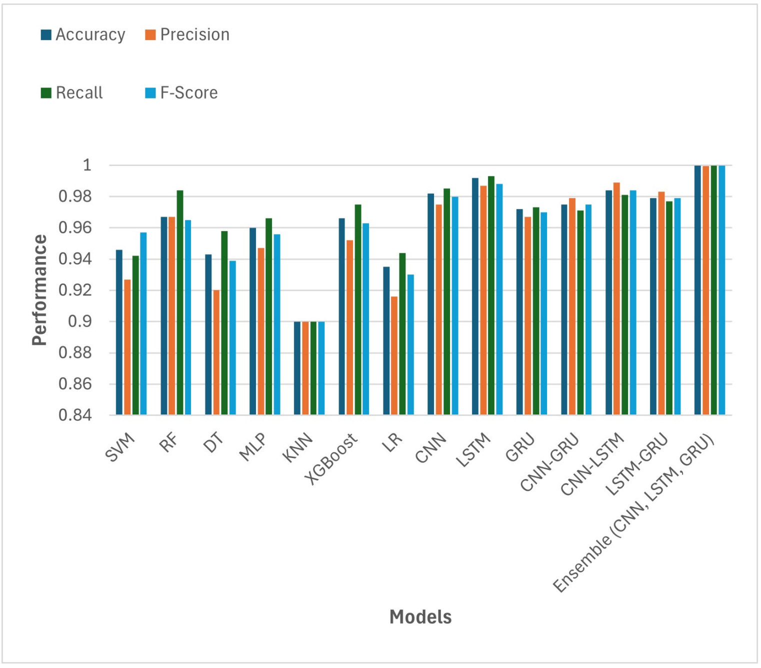

Figure 5. Evaluation metrics for different models with our proposed dataset.

5.2. Comparison with State-of-the-Art Models

Table 6 provides a detailed comparison of the efficacy of several (XSS) detection models. At the forefront is the proposed model, representing an innovative approach that combines a bipartite and raw text as features. The results of this study's experiments and those of previous studies are listed in Table 6.

Table 6. Comparison with state-of-the-art methods (metrics presented as ratios).

Reference	Model	Accuracy	Precision	Recall	F-Score
Mokbal et al. [27]	XGboost	0.995	0.995	0.99	0.995
Wu et al. [43]	TextCNN	0.997	0.997	0.997	0.997
Chaudhary et al. [44]	Self-organizing-map (SOM)	0.9904	0.993	0.991	0.9938
Pan et al. [45]	FSXSS	0.9	NA	NA	0.79
Bach et al. [21]	Stacking ensemble learning	0.9987	0.998	0.997	0.9987
Bakir et al. [22]	Vanilla NN 2	0.9916	0.9922	0.9899	0.9946
Luu et al. [46]	XSShield	0.9927	0.9965	0.9561	0.9759
Odeh et al. [47]	Hybrid RNN-CNN	0.967	0.977	0.956	0.967
Proposed model	Bipartite feature + Ensemble classifier	0.9997	0.9995	0.9998	0.9997

This novel proposal (bipartite feature set and ensemble learner) was shown to have a remarkable accuracy of 99.97%, signifying its ability to make correct predictions. Moreover, the proposed model exhibited an impressive precision score of 99.95%, underscoring its proficiency in correctly identifying positive instances with minimal FPs. By achieving a substantial recall score of 99.98%, the model demonstrated a commendable capacity to capture a significant portion of the positive cases. Consequently, its F1-score of 99.97% indicates its balanced precision and recall. This result outperforms those shown in the studies listed in Table 6. Note that the raw text in the used dataset was recently examined by Bkir et al. [22], who demonstrated an accuracy level of 99.16% and an F-score of 99.46%. Our proposed feature set with the ensemble DL classifier improved the detection accuracy by about 0.8%, which is a substantial improvement in the XSS detection field.

When assessing the robustness of the model and ruling out overfitting, a standard deviation of 0.01% and an average F1 score of 99.97% were shown over 10 folds. Thus, the model demonstrated minimal variance in its predictions and consistent performance across data splits.

6. Discussion

This section discusses the results and presents the advantages associated with using bipartite features. Table 6 shows that the proposed model achieved a 99.9% accuracy rate, a higher accuracy value than those shown in previously reported studies. A novel representation of the payloads and the words within them is introduced by the bipartite feature set. This method records both local and global interactions, which are essential for comprehending the composition of possible XSS payloads. The use of a bipartite feature set and the combination of CNNs, GRUs, and LSTMs is also innovative. By combining these elements, the model may learn sequential and structural patterns in a manner that his impossible for conventional methods. Bipartite representation takes advantage of the malicious payloads' inherent structures rather than relying solely on raw-text features. Consequently, the bipartite network captures the contextual relevance and dependencies between payload components. Bipartite graphs a more comprehensive representation of text data by capturing relationships between sentences and words. This can assist with the identification of patterns and anomalies that conventional approaches might overlook [48]. For instance, the model can precisely detect even complicated or ambiguous scenarios, since the network can recognize several co-occurring phrases or sequences that define an XSS attack. In addition, by projecting the bipartite network onto a single mode, the dimensionality is reduced while meaningful relationships between features are preserved, making the data display more compact and less noisy. This is especially important when dealing with high-dimensional text data. Furthermore, embeddings from bilateral networks are expressive, because they encode the payload's structural arrangement as well as the semantic connections among its elements. These embeddings increase the model's capacity to identify micropatterns suggestive of XSS payloads by acting as different inputs to the corresponding models, such as LSTM and GRU.

In addition to the binary feature set, the ensemble model also combines CNN, GRU, and LSTM to promote the detection capability. Each cluster element plays a specific role: CNN extracts local patterns and n-gram-like features, GRU records short-term dependencies, and LSTM manages long-term relationships within the input. When these models are combined, the clusters leverage each other's strengths, building a classifier where both spatial and sequential feature learning are used to achieve excellence. Combining raw text features with bilateral embeddings improves this coordination further by providing a holistic display of information.

In summary, the bipartite feature set's enhanced value is illustrated by the improvement in accuracy over that shown in previous studies (99.7%). The bipartite representation adds a structural perspective, which lessens the reliance on noisy characteristics and improves the generalization, whereas older models might mostly rely on sequential dependencies in the raw text. Additionally, a key component of this method's effectiveness is the bipartite network's ability to recognize complex relationships within payloads, which helps to locate patterns specific to malicious inputs.

7. Conclusions

This work introduces a novel approach for detecting XSS threats by merging bipartite graph-based feature extraction with an ensemble DL classifier composed of CNN, LSTM, and GRU models. The approach creatively uses bipartite graphs, which are represented as two distinct sets, to reflect the intricate affiliation ties between payloads and their constituent words. By overcoming the limitations of traditional methods, this unique representation enhances the detection of complex and obfuscated XSS payloads. The results demonstrate the effectiveness of the proposed approach, which surpasses state-of-the-art models with a remarkable accuracy of 99.97%. The significant improvements in the precision, recall, and F1-score significantly support the model's adaptability and resilience. By exploiting the contextual relationships and co-occurrence patterns found in the bipartite graph, the proposed method provides a scalable and precise XSS detection method. It is concluded that the combination of ensemble DL classifiers and advanced feature engineering through bipartite graphs could enhance online application security.

Practical Implications of the Bipartite Network for XSS Detection

One of the most significant practical implications of this approach is its potential for real-time web application security. In contemporary web applications, XSS attacks frequently target user inputs. Using the relationships between the payloads and the words they include, the bipartite network model can be incorporated into real-time security systems to evaluate incoming data and identify dangerous payloads. This enables online applications to stop assaults before they can be carried out, avoiding exploitation, defacement, and data theft.

In addition to securing web applications, the bipartite network model can be applied to automated threat detection in APIs. Since many businesses use APIs to communicate online, they are often the focus of XSS attacks. To ensure that fraudulent requests are identified and stopped before they reach crucial backend services, the bipartite approach may be easily integrated into API gateways to monitor and filter payloads based on lexical and structural patterns.

Another significant advantage of this model is its ability to reduce FPs and FNs, which are common challenges in traditional XSS detection methods. Conventional XSS detection techniques sometimes have trouble with FPs and FNs, either overlooking complex attack attempts or blocking valid user inputs. By analyzing the interaction between payloads and their constituent words, the bipartite network improves the detection precision and lowers the FP and FN rates. This guarantees that web apps can continue to be both safe and easy to use.

The model also has notable applications in user-generated content platforms. XSS attacks frequently target platforms such as blogs and social media that permit user-generated material. These platforms can automatically identify and stop dangerous payloads in user inputs before they are published by using the bipartite network concept, protecting users and the application itself.

In addition, by highlighting odd or unique combinations that can point to new attack types, the model makes anomaly identification easier by detecting new payloads with components that mimic established XSS payloads.

Finally, the bipartite network model's adaptability and scalability are essential for managing the increasing intricacy of online applications and changing security risks. The bipartite network-based model can be retrained to identify novel patterns and tactics as attack methods change. Its scalability ensures that it can adjust to the increasing complexity of XSS attacks and makes it appropriate for a broad range of applications, from small-scale web apps to huge enterprise-level systems.

Although the bipartite network-based method for XSS detection has shown encouraging results, a number of issues must be resolved before its full potential can be realized in practical applications. The complexity of model training is one of the main obstacles. Large volumes of labeled data are needed to train the bipartite network. However, obtaining these can be challenging, especially for more complex or unique XSS attacks. Significant computational expenses could also be incurred during the training process, particularly when working with big datasets or high-dimensional feature spaces. This may prevent the concept from being widely adopted, especially by smaller or less well-funded groups. However, the computational overhead could be greatly decreased by employing techniques such as parallel processing, effective data partitioning, and approximate graph methods. The scalability may also be improved by investigating alternate representations that strike a balance between the detection accuracy and computing economy, such as hybrid approaches that combine lightweight sequence-based models with bipartite graphs.

Data imbalance in training datasets is another problem. Datasets used to train XSS detection models in real-world applications are frequently extremely unbalanced, containing considerably more benign inputs than dangerous ones. Similar to many ML models, the bipartite network may have trouble detecting these imbalances, which could result in biases when identifying uncommon or unique XSS payloads. To rectify this imbalance, specific methods such as cost-sensitive learning, undersampling, or oversampling need to be used, complicating the training procedure. It is also necessary to consider the model's scalability and performance in high-traffic settings. Large volumes of incoming payloads can be resource-intensive to process and analyze in real time, even if the bipartite network can be extended to handle larger datasets. Enormous-scale web applications, e-commerce websites, or social media platforms that need to process enormous amounts of traffic quickly may find this especially problematic. To guarantee the model's survival under these kinds of settings, performance optimization without sacrificing detection accuracy is crucial.

Although the method successfully captures structural links in payloads, addressing heavily obfuscated payloads containing rare or unseen tokens is difficult due to its reliance on observed co-occurrence patterns. These payloads, which are frequently made to avoid detection, can interfere with the graph's capacity to generalize to attack vectors that have not been seen before.

Future research should concentrate on improving the model's generalizability and robustness in order to overcome these constraints. To capture semantic similarities between uncommon or novel tokens and well-known patterns, this may entail the use of contextual embeddings such as BERT or token embeddings such as Word2Vec and GloVe. Furthermore, the model may detect abnormal structures in disguised payloads by diversifying the co-occurrence patterns through the use of synthetic or adversarial samples during the training phase. Additionally, future researchers could broaden this approach to find additional web-based vulnerabilities, optimize the computational efficiency, and adapt the model for real-time detection scenarios. In future work, we intend to assess our model's resilience by injecting noise into the test payloads. Through this evaluation, we will determine how well the model responds to input differences, such as modest payload perturbations or obfuscation strategies. In order to increase the model's generalization and resistance to adversarial attacks, we will also investigate adversarial training techniques. Through these tests, we can ensure that our model works well under a variety of uncertain settings, which will increase its suitability for XSS detection duties in the real world.

Funding: The author extend their appreciation to Shaqra University, Saudi Arabia, for supporting this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: This study makes use of the publicly available XSS dataset, which is located in [22].

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Tan, X.; Xu, Y.; Wu, T.; Li, B. Detection of reflected XSS vulnerabilities based on paths-attention method. *Appl. Sci.* **2023**, *13*, 7895.
2. Liu, Z.; Fang, Y.; Huang, C.; Han, J. GraphXSS: An efficient XSS payload detection approach based on graph convolutional network. *Comput. Secur.* **2022**, *114*, 102597.
3. Liu, Z.; Fang, Y.; Huang, C.; Xu, Y. MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Comput. Secur.* **2023**, *124*, 103015.
4. van de Bijl, E.P. Towards Graph-Based Intrusion Detection in Cybersecurity. Master's Thesis, Vrije Universiteit Amsterdam, Amsterdam, Netherlands, 2020.
5. Thajeel, I.K.; Samsudin, K.; Hashim, S.J.; Hashim, F. Machine and deep learning-based xss detection approaches: A systematic literature review. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 101628.
6. Liu, M.; Zhang, B.; Chen, W.; Zhang, X. A survey of exploitation and detection methods of XSS vulnerabilities. *IEEE Access* **2019**, *7*, 182004–182016.
7. Kaur, J.; Garg, U.; Bathla, G. Detection of cross-site scripting (XSS) attacks using machine learning techniques: A review. *Artif. Intell. Rev.* **2023**, *56*, 12725–12769.
8. Sharma, S.; Zavarsky, P.; Butakov, S. Machine learning based intrusion detection system for web-based attacks. In Proceedings of the 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance

and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Baltimore, MD, USA, 25–27 May 2020; pp. 227–230.

9. Alam, F.; Pachauri, S. Comparative study of J48, Naive Bayes and One-R classification technique for credit card fraud detection using WEKA. *Adv. Comput. Sci. Technol* **2017**, *10*, 1731–1743.
10. Yang, W.; Zuo, W.; Cui, B. Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network. *IEEE Access* **2019**, *7*, 29891–29900.
11. Munonye, K.; Péter, M. Machine learning approach to vulnerability detection in OAuth 2.0 authentication and authorization flow. *Int. J. Inf. Secur.* **2022**, *21*, 223–237.
12. Wang, R.; Jia, X.; Li, Q.; Zhang, S. Machine learning based cross-site scripting detection in online social network. In Proceedings of the 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), Paris, France, 20–22 August 2014; pp. 823–826.
13. Kascheev, S.; Olenchikova, T. The detecting cross-site scripting (XSS) using machine learning methods. In Proceedings of the 2020 Global Smart Industry Conference (GloSIC), Chelyabinsk, Russia, 17–19 November 2020; pp. 265–270.
14. Banerjee, R.; Baksi, A.; Singh, N.; Bishnu, S.K. Detection of XSS in web applications using Machine Learning Classifiers. In Proceedings of the 2020 4th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), Kolkata, India, 2–4 October 2020; pp. 1–5.
15. Rathore, S.; Sharma, P.K.; Park, J.H. XSSClassifier: An efficient XSS attack detection approach based on machine learning classifier on SNSs. *J. Inf. Process. Syst.* **2017**, *13*, 1014–1028.
16. Khan, N.; Abdullah, J.; Khan, A.S. Defending malicious script attacks using machine learning classifiers. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 5360472.
17. Alhamyani, R.; Alshammari, M. Machine Learning-Driven Detection of Cross-Site Scripting Attacks. *Information* **2024**, *15*, 420.
18. Wang, Q.; Li, C.; Wang, D.; Yuan, L.; Pan, G.; Cheng, Y.; Hu, M.; Ren, Y. IGXSS: XSS payload detection model based on inductive GCN. *Int. J. Netw. Manag.* **2024**, *34*, e2264.
19. Karim, A.; Shahroz, M.; Mustofa, K.; Belhaouari, S.B.; Joga, S.R.K. Phishing detection system through hybrid machine learning based on URL. *IEEE Access* **2023**, *11*, 36805–36822.
20. Shukla, S.; Misra, M.; Varshney, G. HTTP header based phishing attack detection using machine learning. *Trans. Emerg. Telecommun. Technol.* **2024**, *35*, e4872.
21. Bacha, N.U.; Lu, S.; Ur Rehman, A.; Idrees, M.; Ghadi, Y.Y.; Alahmadi, T.J. Deploying Hybrid Ensemble Machine Learning Techniques for Effective Cross-Site Scripting (XSS) Attack Detection. *Comput. Mater. Contin.* **2024**, *81*, 707.
22. Bakır, R.; Bakır, H. Swift Detection of XSS Attacks: Enhancing XSS Attack Detection by Leveraging Hybrid Semantic Embeddings and AI Techniques. *Arab. J. Sci. Eng.* **2024**, *50*, 1191–1207.
23. Ade, M. A Review of Modern Techniques for Detecting Cross-Site Scripting (XSS) in Web Applications. 2024.
24. Hannousse, A.; Yahiouche, S.; Nait-Hamoud, M.C. Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey. *Comput. Sci. Rev.* **2024**, *52*, 100634.
25. Rodríguez-Galán, G.; Torres, J. Personal data filtering: A systematic literature review comparing the effectiveness of XSS attacks in web applications vs cookie stealing. *Ann. Telecommun.* **2024**, *79*, 763–802.
26. Ramoa, L.; Campos, P. Recommendation Systems in E-commerce: Link Prediction in Multilayer Bipartite Networks. In *Digital Transformation and Enterprise Information Systems*; CRC Press: Boca Raton, FL, USA, 2024; pp. 55–78.
27. Mokbal, F.M.M.; Dan, W.; Xiaoxi, W.; Wenbin, Z.; Lihua, F. 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.* **2021**, *58*, 102813.
28. Gniewkowski, M.; Maciejewski, H.; Surmacz, T.; Walentynowicz, W. Section 2vec: Anomaly detection in HTTP traffic and malicious URLs. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, 27–31 March 2023; pp. 1154–1162.
29. Gniewkowski, M.; Maciejewski, H.; Surmacz, T.R.; Walentynowicz, W. Http2vec: Embedding of http requests for detection of anomalous traffic. *arXiv* **2021**, arXiv:2108.01763.
30. Abu, T.N.A.; Doh, K.G. An Analysis of Machine-Learning Feature-Extraction Techniques using Syntactic Tagging for Cross-site Scripting Detection. *J. Softw. Assess. Valuat. (한국소프트웨어감정평가학회 논문지)* **2022**, *18*, 107–118.
31. de Albuquerque Oliveira, M.C. A Hybrid Machine Learning System for Vulnerability Detection in Web Applications. Master's Thesis, Universidade de Lisboa, Lisbon, Portugal, 2023.
32. Rozi, M.F.; Ban, T.; Ozawa, S.; Yamada, A.; Takahashi, T.; Inoue, D. Securing Code with Context: Enhancing Vulnerability Detection through Contextualized Graph Representations. *IEEE Access* **2024**, *12*, 142101–142126.

33. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53.
34. Dong, Y.; Wang, R.; He, J. Real-time network intrusion detection system based on deep learning. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019; pp. 1–4.
35. Chaudhary, P.; Gupta, B.B.; Chang, X.; Nedjah, N.; Chui, K.T. Enhancing big data security through integrating XSS scanner into fog nodes for SMEs gain. *Technol. Forecast. Soc. Chang.* **2021**, *168*, 120754.
36. Kuppa, K.; Dayal, A.; Gupta, S.; Dua, A.; Chaudhary, P.; Rathore, S. ConvXSS: A deep learning-based smart ICT framework against code injection attacks for HTML5 web applications in sustainable smart city infrastructure. *Sustain. Cities Soc.* **2022**, *80*, 103765.
37. Maurel, H.; Vidal, S.; Rezk, T. Statically identifying XSS using deep learning. *Sci. Comput. Program.* **2022**, *219*, 102810.
38. Shahid, W.B.; Aslam, B.; Abbas, H.; Khalid, S.B.; Afzal, H. An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. *J. Netw. Comput. Appl.* **2022**, *198*, 103270.
39. Hochreiter, S. Long Short-term Memory. In *Neural Computation*; MIT-Press: Cambridge, MA, USA, 1997.
40. Gao, C.; Yan, J.; Zhou, S.; Varshney, P.K.; Liu, H. Long short-term memory-based deep recurrent neural networks for target tracking. *Inf. Sci.* **2019**, *502*, 279–296.
41. Guan, D.; Yuan, W.; Lee, Y.K.; Najeebullah, K.; Rasel, M.K. A review of ensemble learning based feature selection. *IETE Tech. Rev.* **2014**, *31*, 190–198.
42. Ganaie, M.A.; Hu, M.; Malik, A.K.; Tanveer, M.; Suganthan, P.N. Ensemble deep learning: A review. *Eng. Appl. Artif. Intell.* **2022**, *115*, 105151.
43. Wu, A.; Feng, Z.; Li, X.; Xiao, J. ZTWeb: Cross site scripting detection based on zero trust. *Comput. Secur.* **2023**, *134*, 103434.
44. Chaudhary, P.; Gupta, B.; Singh, A.K. Adaptive cross-site scripting attack detection framework for smart devices security using intelligent filters and attack ontology. *Soft Comput.* **2023**, *27*, 4593–4608.
45. Pan, H.; Fang, Y.; Guo, W.; Xu, Y.; Wang, C. Few-shot graph classification on cross-site scripting attacks detection. *Comput. Secur.* **2024**, *140*, 103749.
46. Luu, G.H.; Duong, M.K.; Pham-Ngo, T.P.; Ngo, T.S.; Nguyen, D.T.; Nguyen, X.H.; Le, K.H. XSShield: A Novel Dataset and Lightweight Hybrid Deep Learning Model for XSS Attack Detection. *Results Eng.* **2024**, *24*, 103363.
47. Odeh, A.; Taleb, A.A. XSSer: Hybrid deep learning for enhanced cross-site scripting detection. *Bull. Electr. Eng. Inform.* **2024**, *13*, 3317–3325.
48. Reka, R.; Karthick, R.; Ram, R.S.; Singh, G. Multi head self-attention gated graph convolutional network based multi-attack intrusion detection in MANET. *Comput. Secur.* **2024**, *136*, 103526.