XSS Attack Detection with Deep Learning and AdaBoost

Yixian Liu,Lu Wang,Yupeng Dai

Abstract—**Security constitutes a foundational aspect of web application development, primarily due to the need to safeguard substantial volumes of sensitive data. Among the various security threats, Cross-Site Scripting (XSS) attacks continue to pose a significant risk to the integrity and security of contemporary web applications. Traditional XSS detection methods are hindered by limited detection coverage, difficulties in adapting to highly dynamic web environments, and high false positive rates. While deep learning-based approaches offer advantages such as robust feature extraction, they still face challenges related to dataset quality, class imbalance, and the need for further improvements in accuracy. To overcome these limitations, this paper proposes an XSS detection method that integrates Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM) networks, and a Transformer based multi-attention mechanism. Moreover, the AdaBoost algorithm is integrated for the final classification decision. Empirical evaluations conducted on two publicly available datasets demonstrate that the proposed model surpasses existing detection approaches, achieving accuracy rates of 99.73% and 99.52%, respectively.**

Index Terms—**Security, Cross-Site Scripting, Deep learning, Transformer, AdaBoost.**

I. INTRODUCTION

The current development of network technology has greatly changed people's lives and has become an indispensable and important tool. According to the Open Web Application Security Project (OWASP), Cross-Site Scripting (XSS) has consistently been classified within the top 10 security vulnerabilities affecting web applications worldwide, securing positions 4, 4, 1, 3, 7, and 3 in the years 2004, 2007, 2010, 2013, 2017, and 2021, respectively [1]. In recent years, artificial intelligence (AI) techniques have garnered significant attention, particularly in machine learning (e.g., Decision Trees (DTs), Random Forests (RF), and Support Vector Machines (SVM) [2]) and deep learning (e.g., Convolutional Neural Networks (CNN) [3] and Long Short-Term Memory Networks (LSTM) [4]), due to their notable advantages in detecting attacks and intrusions. Compared to traditional detection methods (e.g., input validation, static analysis, and dynamic analysis) [5,6], these methods address certain drawbacks of traditional techniques, enhancing both detection accuracy and adaptability. Despite progress, detecting XSS attacks remains a challenge due to factors such as reliance on single base classifiers, limited dataset size, class imbalance, and the lack of model interpretability. Ensemble learning techniques, which effectively address these challenges, have shown superior performance in detecting XSS attacks and other cybersecurity threats [7]. Despite the widespread application of deep learning models in XSS detection, many studies overlook the advantages of combining multiple models to harness their complementary strengths and improve classification accuracy.

To improve the detection of XSS attacks, this research introduces a framework that integrates deep learning techniques with the AdaBoost ensemble algorithm. This research makes the following key contributions: 1) We integrate features extracted by CNN-LSTM and Transformer models to construct an integrated model with a parallel structure, significantly improving the model's overall expressiveness and generalization capability. 2) Building on this parallel-structured integrated model, we integrate the AdaBoost classifier to enhance performance. Feeding the extracted features into AdaBoost significantly improves the model's classification accuracy.

II. RELATED WORK

XSS, a form of client-side exploitation, allows attackers to inject malicious JavaScript into web pages—thereby gaining unauthorized access to sensitive user data such as session tokens and login credentials. XSS vulnerabilities are generally categorized into client-side and server-side vulnerabilities [8].

Early studies primarily focused on analyzing web application source code for XSS vulnerabilities using static inspection techniques [9]. Static analysis methods rely on source code inspection, providing efficient detection but often resulting in false positives and false negatives [10]. In contrast, dynamic detection methods simulate attacker behavior to identify vulnerabilities [11].

Although these methods deliver high detection accuracy without needing access to the source code, their effectiveness is often constrained by specific attack vectors, which can notably affect detection performance [12].

With the advancements in machine learning, researchers have proposed XSS detection techniques based on these technologies to address the shortcomings of static and dynamic methods [13]. Some integrated methods employ supervised ensemble learning techniques, such as AdaBoost,

Manuscript received December 11, 2024; revised May 11 , 2025.

Y. Liu is a senior engineer at the School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710000, China (e-mail: liu-yi-xian@xupt.edu.cn).

L. Wang is a graduate student at the School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710000, China (corresponding author to provide phone: +086-158-3589-0996,e-mail: 1633171257@qq.com).

Y. Dai is a graduate student at the School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710000, China (e-mail: daiyupeng@stu.xupt.edu.cn).

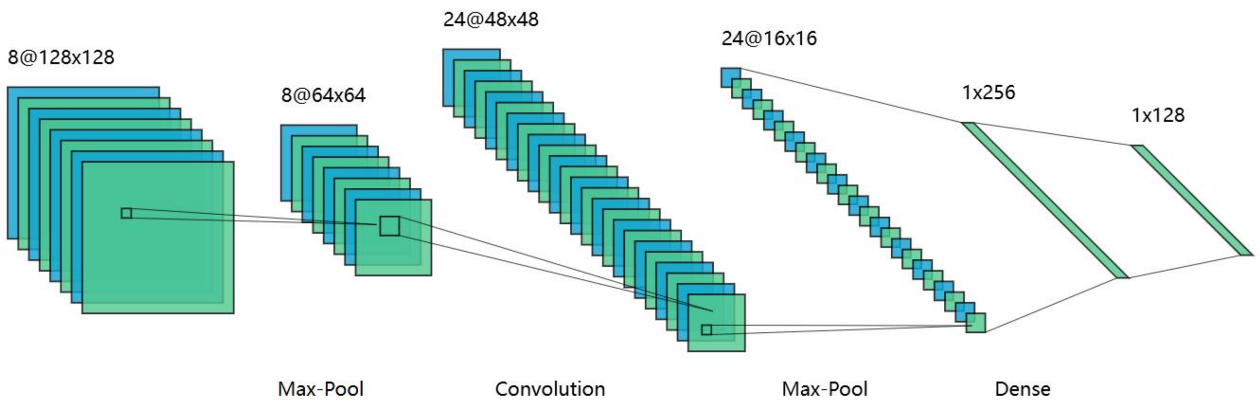

Fig. 1. The CNN architecture

achieving detection accuracy as high as 99.8% [14]. However, these methods typically rely on manual feature extraction, which can be both time-intensive and influenced by personal judgment [15]. Unlike traditional machine learning methods, deep learning autonomously extracts textual features, overcoming their inherent limitations [16]. Compared to traditional methods, XSS detection powered by deep learning offers greater automation, improved accuracy, and reduced false positive rates [17].

Several deep learning models have achieved outstanding accuracy in detecting XSS attacks: LSTM networks reached 99.25% [18], CNN-LSTM combinations attained 99.4% [19], and LSTM with an attention mechanism achieved 99.11% [20].However, as attackers continue to innovate their techniques to bypass detection, the primary challenge in current research is optimizing deep learning models to effectively detect the continuously evolving techniques used in XSS attacks.

III. Research background

A. CNN architecture

The design of CNN is influenced by biological neurons in the human and animal brain and shares similarities with traditional neural networks. CNNs offer three primary advantages: translation invariance, sparse interactions, and parameter sharing [21]. A CNN, by design, includes convolutional layers, pooling layers, and fully connected layers, employing weight sharing and local connectivity to enable automatic feature extraction, thereby enhancing the efficiency and accuracy of visual recognition and classification [22]. The structure is illustrated in Fig. 1.

For a one-dimensional Convolutional Neural Network (1D-CNN), its mathematical formulation is expressed as folows (where x denotes the input signal, n represents the length of the input sequence, k signifies the size of the convolutional kernel, h indicates the convolutional kernel, s denotes the stride length, and y represents the output feature map) [23].

$$y_n = \begin{cases} \sum_{i=0}^{k} x(n+i)h(i), n=0 \\ \sum_{i=0}^{k} x(n+i+(s-1))h(i), otherwise \end{cases} \tag{1}$$

B. LSTM architecture

LSTMs, as a specialized type of recurrent neural network, offer enhanced capabilities for handling sequential data. They incorporate multiple gating mechanisms and internal states within their memory cells, effectively addressing the gradient vanishing problem commonly encountered in traditional RNNs [24]. Each gate processes the input and modulates the flow of information from previous time steps while maintaining the sequential processing nature of RNNs. Fig. 2 depicts the architecture of LSTM.

Given the input at time step t, along with the previous hidden state h_{t-1} and cell state c_{t-1}, the calculations performed within an LSTM cell are specified by the following equations:

- The forget gate:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{2}$$

- The input gate:

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{3}$$

$$\tilde{C}_t = \tanh(W_c [h_{t-1}, x_t] + b_c) \tag{4}$$

- The update cell state:

$$C_t = C_{t-1} \odot f_t + i_t \odot \tilde{C}_t \tag{5}$$

- The output gate:

$$o_t = sigmoid(W_o \odot [h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = o_t \odot \tanh(C_t) \tag{7}$$

W_f, W_i, W_c and W_o represent the weight matrices for the forget gate, input gate, candidate cell state, and output gate, respectively, while b_f, b_i, b_c and b_o are their respective bias terms.

C. Self-attention mechanism

The self-attention mechanism captures long-range dependencies by enabling direct interactions between all positions within a sequence. It primarily consists of scaled dot-product attention and multi-head attention. In scaled dot-product attention, attention weights are computed by taking the dot product of the query and key vectors, dividing by the square root of the key dimension to reduce gradient instability, and applying a softmax function to obtain norma-

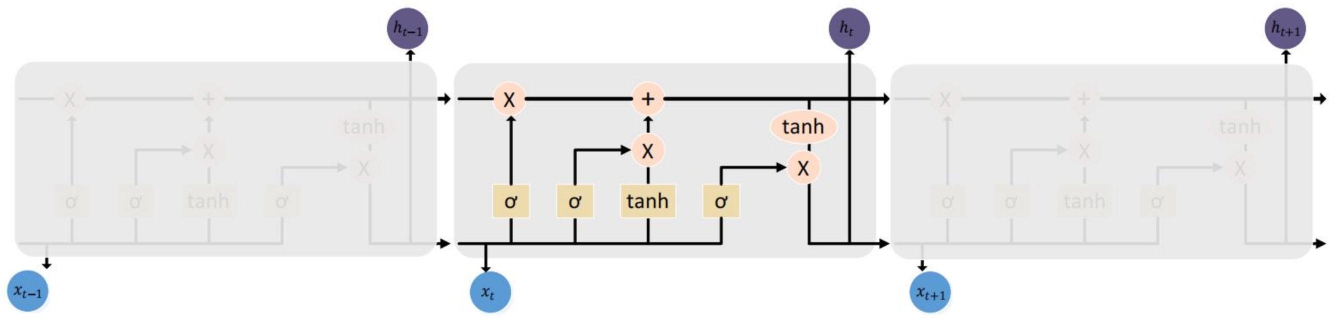

Fig. 2. The LSTM structure diagram

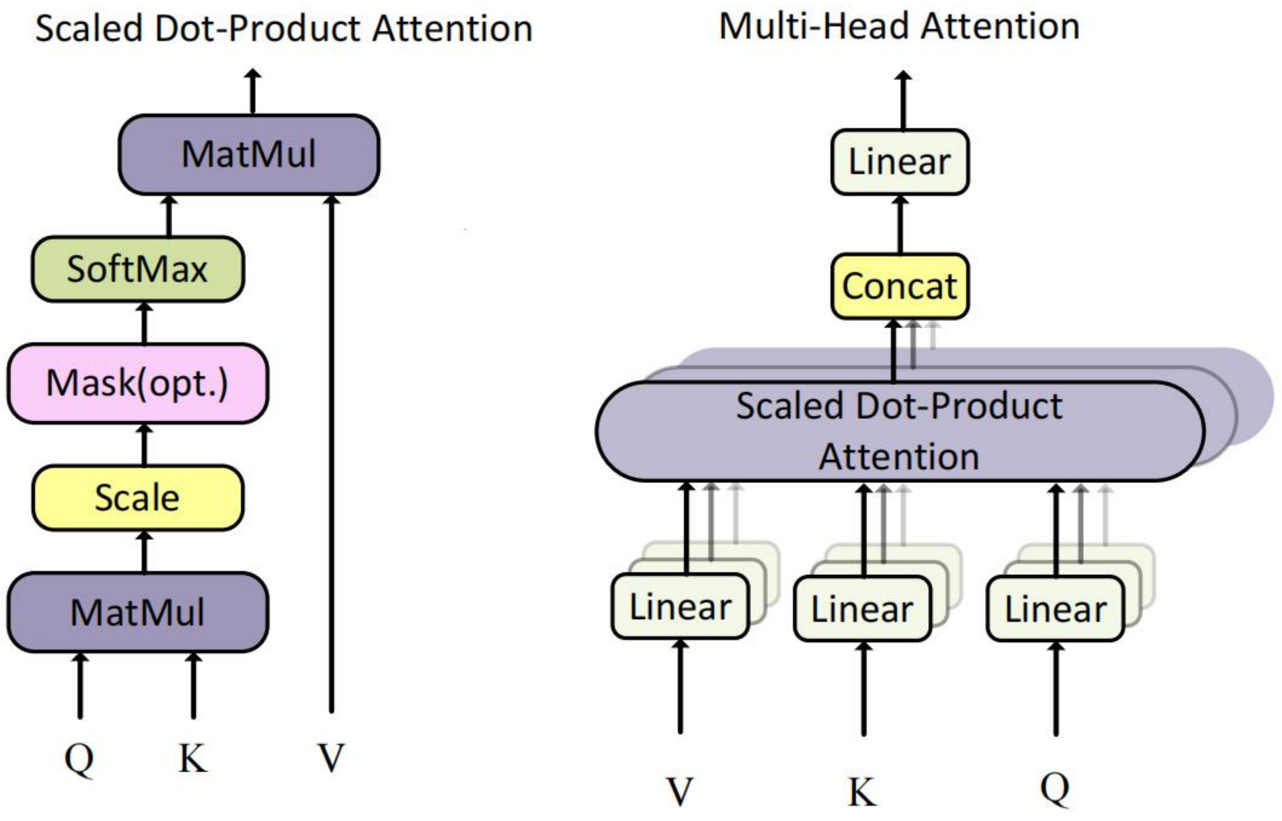

Fig. 3. The self-attention mechanism

lized attention scores. Multi-head attention divides the query, key, and value matrices into multiple heads using separate linear projections, performs attention in parallel across these heads, and then concatenates the resulting outputs to produce the final representation. By enabling the model to focus on information from different representation subspaces across various positions at the same time, this structure enhances its ability to represent complex data.

Additionally, positional encoding is integrated to encode word order information, addressing the sequence modeling limitations of the self-attention mechanism. These attributes together enhance the Transformer model's effectiveness and scalability in processing sequential data [25]. Fig. 3 depicts the self-attention mechanism.

D. *AdaBoost classifier*

The AdaBoost algorithm, introduced by Freund and Schapire in 1995, is a widely adopted ensemble learning technique based on the Boosting framework. It improves predictive performance by sequentially training a series of weak learners, typically decision trees, and aggregating them into a strong classifier. During each iteration, AdaBoost dynamically adjusts the sample weights based on the misclassifications made by the preceding model, assigning higher weights to incorrectly predicted instances. This approach ensures that subsequent learners focus more on difficult cases. The contribution of each weak learner is determined by its classification error; a lower error results in a higher contribution. Finally, the algorithm integrates the outputs of all weak learners using weighted voting to produce the final prediction [27].

IV. Research Methods

A. *Preprocessing*

Data Collection: Given that Xssed.com is currently one of the most comprehensive platforms for aggregating XSS vulnerabilities, this study utilizes web crawling techniques

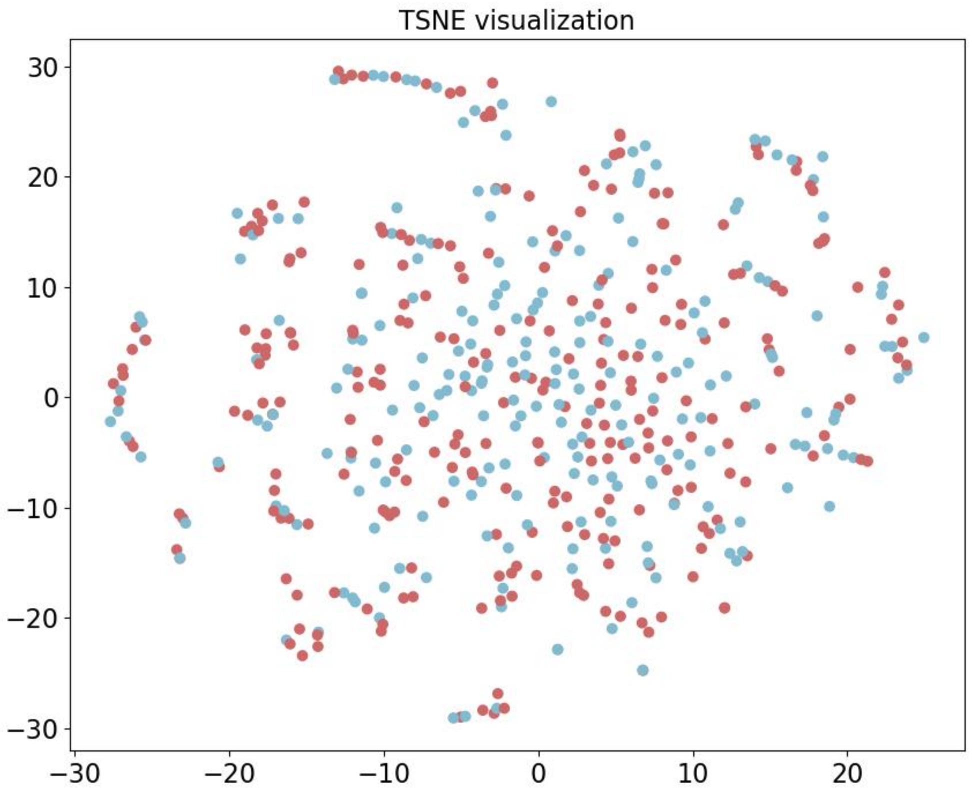

Fig. 4. Visualization of word vector dimensionality reduction

to extract malicious samples from the site, which serve as positive instances. These samples include a wide range of c-ommonly observed XSS attack types, featuring both malfo-rmed and obfuscated payloads. In contrast, benign samples are obtained from the DMOZ database and used as negativeinstances.

Data Preprocessing: In this study, the Word2Vec model is utilized to train a vocabulary and produce the corresponding word embeddings. To visualize the resulting embeddings, t-distributed Stochastic Neighbor Embedding (t-SNE) is used for dimensionality reduction. Specifically, the first 100 dimensions of the original 128-dimensional embeddings arereduced to two dimensions for visualization, as shown in Fig. 4.

Data Cleaning: To ensure data consistency and improve model performance, all positive and negative samples undergo a data cleaning process. The host and path components are removed from URLs, leaving only the malicious payload intact. Furthermore, all hyperlinks are normalized by substituting them with a standardized placeholder to ensure uniformity across the dataset.The cleaned URLs are then encoded and stored in a CSV file for subsequent processing.

Segmentation: Tokenization is applied to all samples. Unlike typical natural language text, the experimental data contains HTML tags, function bodies, parameter names, and other programming-related structures. Therefore, custom tokenization rules are established to handle these elements appropriately. Specifically, the following components are treated as separate tokens and enclosed in single quotation marks:

1) http/https links
2) html tags < script>
3) html tag beginning < h1
4) Parameter name topic=
5) function body alert(
6) Mixed strings of characters and numbers

An effective word embedding model enables textual content to be encoded as vectors within a continuous vector space. The data preprocessing workflow is depicted in Fig. 5.

B. Model Analysis and Architecture

XSS attack detection primarily involves the analysis of textual data such as web requests, URLs, and form inputs, which may contain malicious scripts exhibiting specific paterns, including distinct tags, attributes, or embedded JavaScript code. CNNs are effective in identifying localized patterns within sequences, making them suitable for detecting typical XSS attack signatures. However, more advanced XSS attacks often consist of longer, obfuscated payloads that exploit complex application logic. LSTM, with their capacity to model long-range dependencies and contextual semantics, are better suited for capturing such complex patterns. Nonetheless, their inherent sequential computation limits their ability to model distant dependencies efficiently.

On the other hand, Transformer architectures utilize self-attention mechanisms to facilitate global interactions among sequence elements, enabling the direct modeling of long-range dependencies without relying on stepwise state propagation.This global perspective strengthens the model's

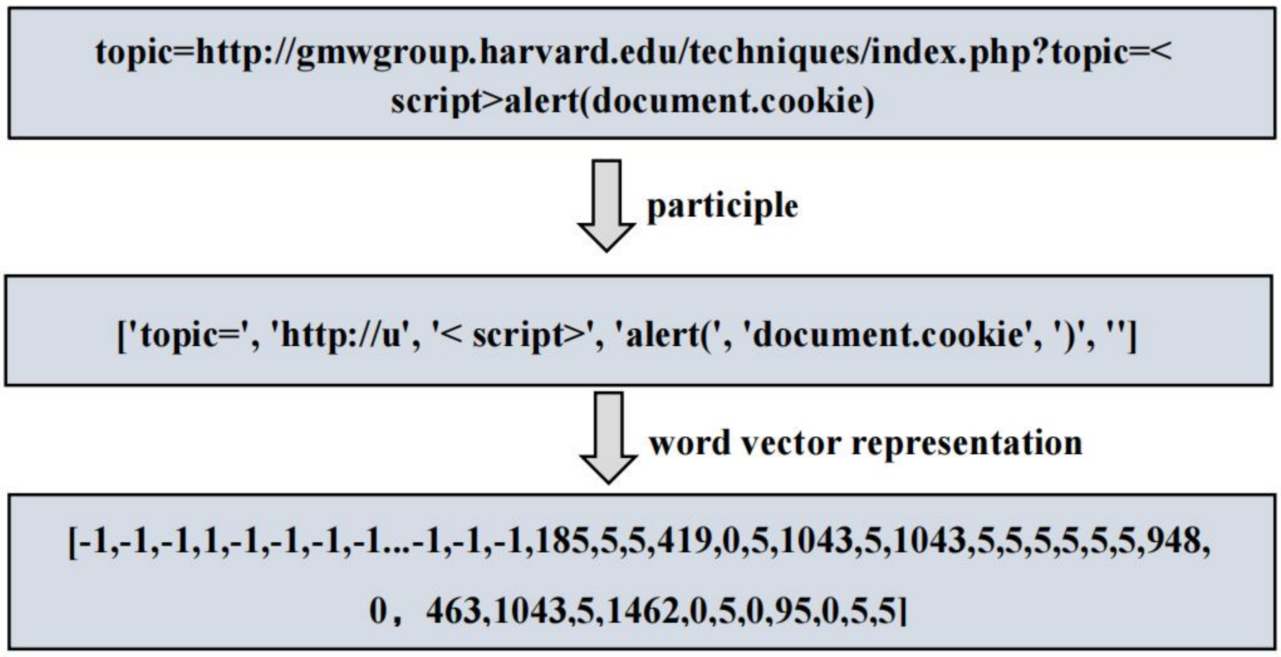
topic=http://gmwgroup.harvard.edu/techniques/index.php?topic=< script>alert(document.cookie)
participle
['topic=', 'http://u', '< script>', 'alert(', 'document.cookie', ')', '']
word vector representation
[-1,-1,-1,1,-1,-1,-1,-1...-1,-1,-1,185,5,5,419,0,5,1043,5,1043,5,5,5,5,5,5,948, 0， 463,1043,5,1462,0,5,0,95,0,5,5]

Fig. 5. Data preprocessing process

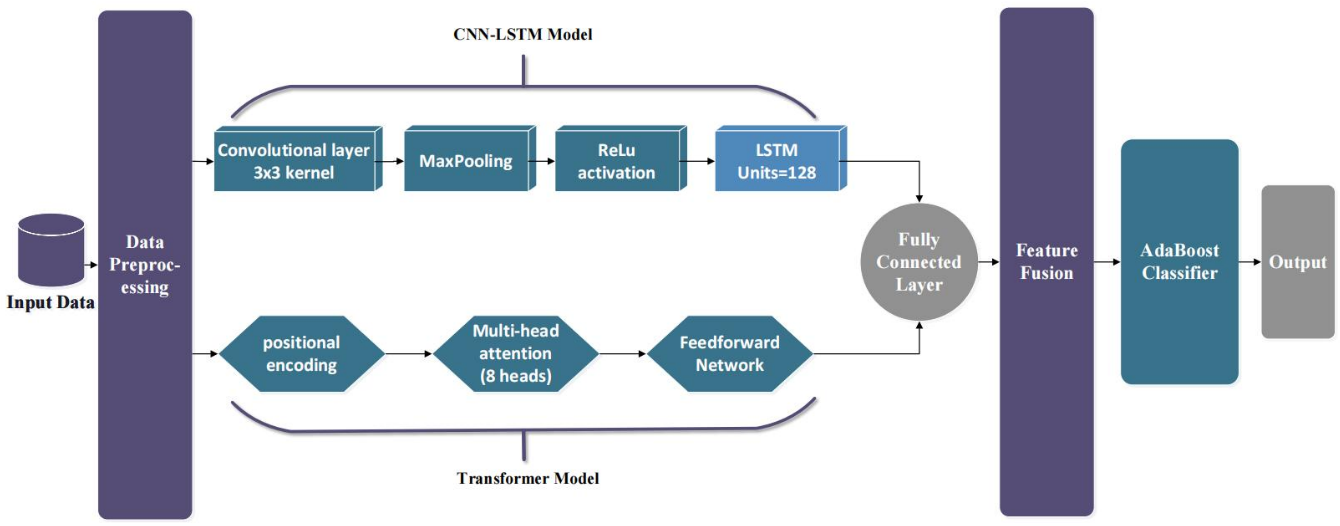

Fig. 6. Framework Diagram of the Proposed Model

capacity to capture intricate semantic structures, leading to improved detection accuracy for advanced XSS variants.

Moreover, XSS detection is typically framed as an imbalanced classification problem, where benign samples far outnumber attack samples. Class imbalance presents a challenge in further improving performance beyond a certain accuracy threshold. AdaBoost, an ensemble learning method, effectively addresses this issue by iteratively training multiple weak classifiers (typically decision trees) and adjusting the weights of misclassified instances, thereby enhancing the identification of minority-class samples.

Leveraging this advantage, the proposed framework adopts a multi-branch architecture that seamlessly integrates both sequential and attention-based neural models with AdaBoost, thereby significantly improving classification accuracy, robustness, and overall performance.

The overall architecture of the proposed model, as illustrated in Fig. 6, consists of three main components. First, preprocessed input data is fed into two parallel feature extractors: a CNN-LSTM hybrid and a Transformer-based encoder. The CNN-LSTM model adopts a sequential design, where CNN layers capture local features followed by LSTM layers that learn global sequential dependencies. In parallel, the Transformer encoder includes positional encoding to preserve order information and a multi-head self-attention mechanism, implemented with eight attention heads and two encoding layers. Both models independently produce classification outputs. The second component fuses the features from both models and conducts an intermediate classification on the merged representations.

Finally, AdaBoost is applied in the third component to perform the final classification, harnessing the combined deep feature representations to enhance both robustness and accuracy. This effectively boosts the model's ability to generalize on unseen data, ensuring stable and reliable performance across diverse scenarios.

TABLE I
EXAMPLES OF SAMPLE FEATURES FOR THE TWO DATASETS

Dataset1	Dataset2
<tt onmouseover="alert(1)">test</tt>	symbol%3D%3Ch1%3E%3Cscript%3Ealert%28/hacked/%29%3C/script%3E%3C/h1%3E%26id%3D58873%22%27
<caption draggable="true" ondragleave="alert(1)">test</caption>	RequestCookies%3D%26Requestdate%3D%26refer%3D%22%3E%3Cscript%3Ealert%28%27/xssed/%27%29%3C/script%3E
<button onmouseover="alert(1)">test</button>	page%3Dsearch%26q%3D%22%3E%3Cscript%3Ealert%281%29%3C/script%3E%26qmatch%3D0%26cntr%3D0%26domain%3D0%26find%3Dall%26from%3D%26to%3D
<caption onpointerdown=alert(1)>XSS</caption>	phrase%3D%27%3E1%3C/a%3E%3Cscript%3Ealert%28document.cookie%29%3C/script%3E
<figcaption onpointerleave=alert(1)>XSS</figcaption>	siteSect%3D880%26searchString%3D%22%3E%3Cscript%3Ealert%28%22Fugitif%22%29%3C/script%3E

V. Experiments and Results

A. Dataset

This study employs two publicly available datasets. The first dataset (Dataset 1) is an XSS corpus specifically curated for deep learning applications, as provided by [28]. It includes 6,313 benign samples and 7,373 malicious samples, collected from sources such as PortSwigger, OWASP, and the XSS Cheat Sheet. The second dataset (Dataset 2) is constructed using data from the DMOZ directory and the XSSed database, containing 34,500 benign samples and 30,110 malicious samples [29]. To ensure a balanced evaluation, each dataset was randomly partitioned, with 70% allocated for training and 30% for testing. Table I presents representative feature examples from each dataset.

It is significant to mention that the first dataset employed in this study consists of HTML-encoded XSS samples featuring complete front-end tag structures, which are intended to simulate executable attack code within realistic web environments. In contrast, the second dataset comprises URL encoded XSS payloads, designed to model attack vectors as they occur in actual network communications. The selection of these two distinct types of datasets is motivated by the need to comprehensively capture the diverse manifestations of XSS attacks: the first dataset facilitates the modeling of client-side attack behaviors within rendered web pages, whereas the second captures the obfuscation characteristics of payloads transmitted through backend systems. This complementary dataset design enhances the generalization capability and robustness of the detection model across a wide range of real-world application scenarios.

B. Experimental environment and parameter settings

The experimental platform and hyper parameter settings used in this study are provided in Table II.

TABLE II
EXPERIMENTAL PLATFORM ENVIRONMENT AND HYPERPARAMETER SETTINGS

Designation	Versions/parameters
Operating System	Windows 11
GPU	NVIDIA RTX3080
CPU	i9-13900
RAM	32G
framework	Pytorch
CUDA version	12.6
Python	3.8.19
Batch size	64
Convolution filters	64
The kernel size of the filter	3
LSTM hidden units	64
Attention head	8
Feedforward network dimension	128
Fully connected layer	64
Activation function	ReLu
Classification function	Sigmoid
Optimizer	Adam
Learning rate	0.01
Estimators	50
Epochs	10

C. Experimental design

In this study, the proposed model is compared with several leading approaches, all evaluated using the same dataset, including the LSTM model from [18], the CNN-LSTM model from [19], and the LSTM-Attention model from [20]. Furthermore, an experiment was conducted to empirically assess the performance improvements gained by integrating AdaBoost into the proposed model.

The selection of these models is motivated by their status as widely recognized baselines in the field of XSS attack detection. Comparing the proposed method against these well-established models enables a more rigorous and meaningful assessment of its effectiveness and potential improvements.

D. Experimental results and Analysis

In this study, the evaluation of the proposed model is based on four primary metrics: accuracy, precision, recall, and F1-score. Accuracy (Acc) refers to the proportion of correctly classified samples out of the total. Precision (P), on the other hand, measures the fraction of predicted malicious instances that are truly positive. Similarly, recall (R) quantifies the model's ability to correctly identify actual malicious samples. The F1-score, which is the harmonic mean of precision and recall, provides a more balanced view of model performance.In terms of classification results, True Positive (TP) indicates the number of correctly classified positive instances, while True Negative (TN) refers to correctly identified negative instances. On the contrary, False Positive (FP) represents negative instances misclassified as positive, and False Negative (FN) refers to positive samples incorrectly classified as negative. The evaluation metrics are computed as follows, with accuracy calculated by the formula:

$$ACC = \frac{TP}{TP + FN + TN + FP} \tag{8}$$

Precision is defined as the following equation:

$$P = \frac{TP}{TP + FP} \tag{9}$$

Recall is defined as the following equation:

$$R = \frac{TP}{TP + FN} \tag{10}$$

$F1$-score is defined as the following formula:

$$F1 - score = \frac{2TP}{2TP + FN + FP} \tag{11}$$

Table III presents the confusion matrix [30] for each evaluation parameter mentioned above.

TABLE III
CONFUSION MATRIX

Determination of maliciousness	Predicted to be malicious	Predicted as normal
Actual malicious	TP	FN
Actual Normal	FP	TN

In the training process, a learning rate of 0.01 is used for the proposed model, with the stochastic gradient descent (SGD) algorithm and a binary cross-entropy loss function applied.

Fig. 7 illustrates the accuracy trend throughout the training process on the first dataset, while Fig. 8 displays the corresponding loss trend. After 10 training epochs, the model achieves an accuracy of 99.76% on the test set, with the loss value reduced to 0.0069, demonstrating the remarkable capability of the proposed model in detecting XSS attacks based on HTML structure and JavaScript event triggers. For URL encoding and complex XSS attacks, the proposed model achieves an accuracy of 99.52% on the test set, with the loss value reduced to 0.0187, as shown in Fig. 9 and Fig. 10. The experimental results indicate that the proposed model effectively simulates security detection in real world scenarios.

Fig. 11 and Fig. 12 present the performance evaluation results of the model on two different datasets. The results of the experiments confirm that the proposed model surpasses the comparison models in performance on both datasets.

The first dataset consists of common HTML statements, which are relatively simple in structure and easy to parse. On this dataset, the proposed model achieves a precision of 99.73%, a recall of 99.77%, and an F1-score of 0.9975. These results, consequently, highlight the model's capability to efficiently extract key information from HTML statements while achieving a strong balance between prediction accuracy and coverage.

Compared to the simpler structure of the first dataset, the second dataset comprises more challenging, URL-encoded complex statements. These inputs are not only structurally intricate but also often contain substantial extraneous content, posing greater demands on the model's parsing and recognition capabilities. On this dataset, the proposed model achieves an accuracy of 99.79%, reflecting a very low false positive rate and effectively preventing the misclassification of benign traffic. With a recall of 99.18%, the model successfully identifies the vast majority of malicious or anomalous instances. The F1-score of 0.9948 underscores its ability to maintain an optimal equilibrium between precision and recall, demonstrating robust adaptability in handling complex XSS scenarios.

TABLE IV
PERFORMANCE COMPARISON ON DATASET 1

model	Accuracy %	Precision %	Recall %	F1-score
[18]	99.61	99.63	99.63	0.9963
[19]	99.41	99.23	99.68	0.9946
[20]	99.41	99.95	98.96	0.9945
Proposed model	99.73	99.73	99.77	0.9975

TABLE V
PERFORMANCE COMPARISON ON DATASET 2

model	Accuracy %	Precision %	Recall %	F1-score
[18]	99.36	99.74	98.90	0.9932
[19]	99.31	99.40	99.13	0.9926
[20]	99.34	99.80	98.78	0.9929
Proposed model	99.52	99.79	99.18	0.9948

Tables IV and V provide a comparative summary of model performance on Dataset 1 and Dataset 2.

On Dataset 1, the model from [18] achieves an accuracy of 99.61% and an F1-score of 0.9963; the model from [19] achieves 99.41% accuracy and an F1-score of 0.9946; and the model from [20] achieves 99.41% accuracy with an F1-score of 0.9945. In comparison, the proposed model outperforms others, with an accuracy of 99.73%, precision of 99.73

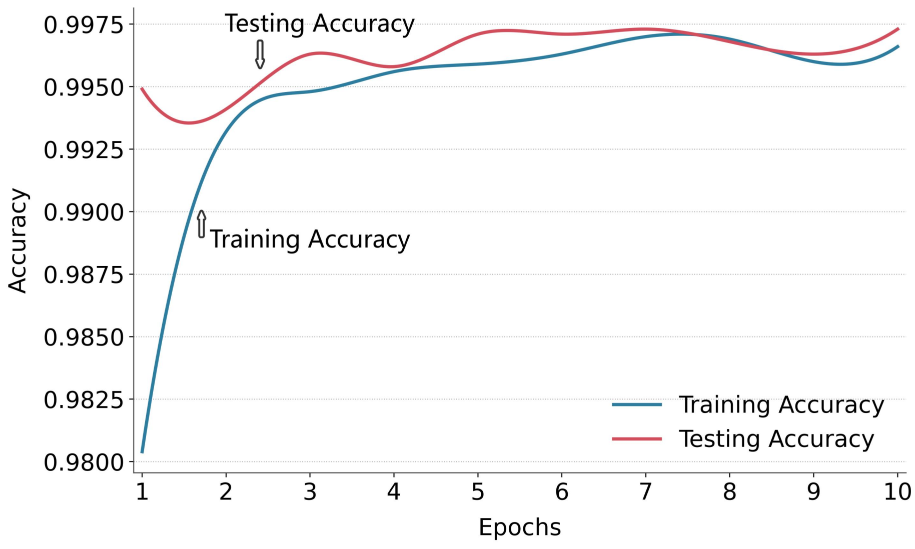

Fig. 7. Accuracy of the proposed model on Dataset 1

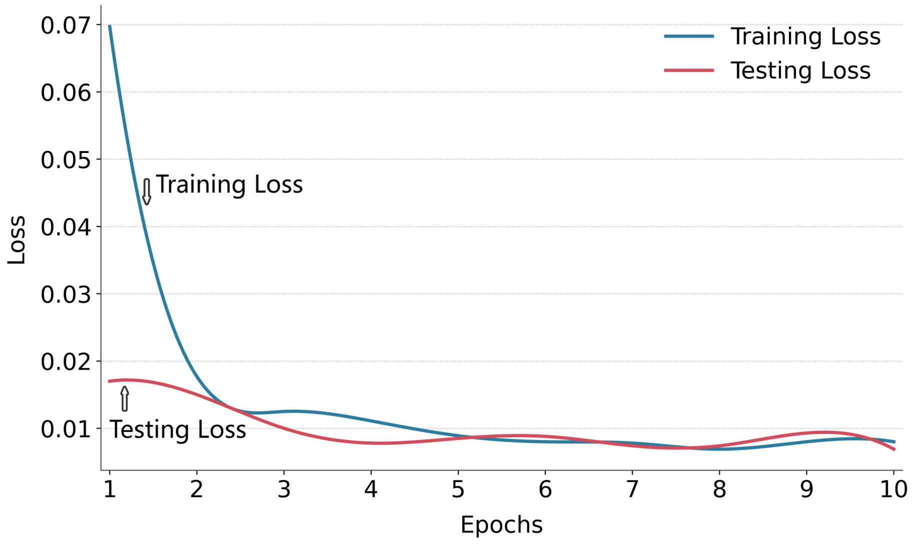

Fig. 8. Loss of the Proposed Model on Dataset 1

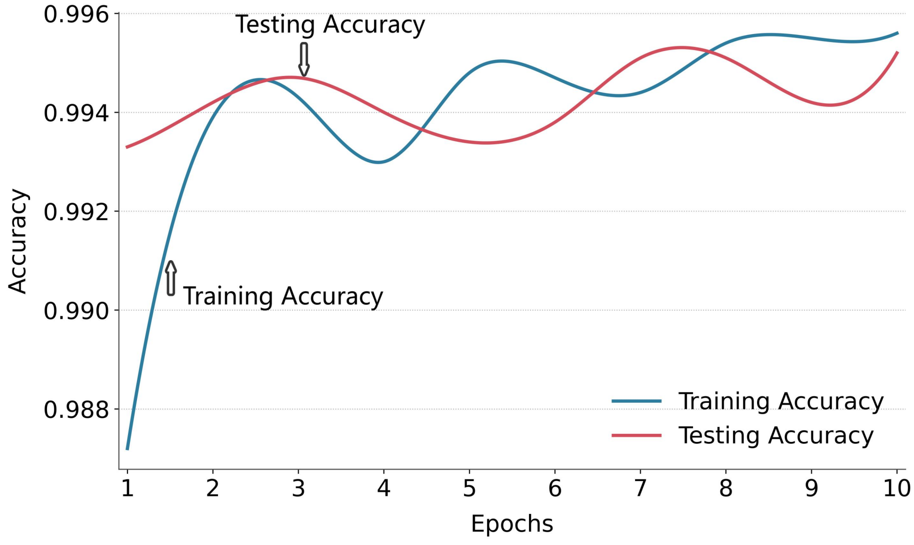

Fig. 9. Accuracy of the proposed model on Dataset 2

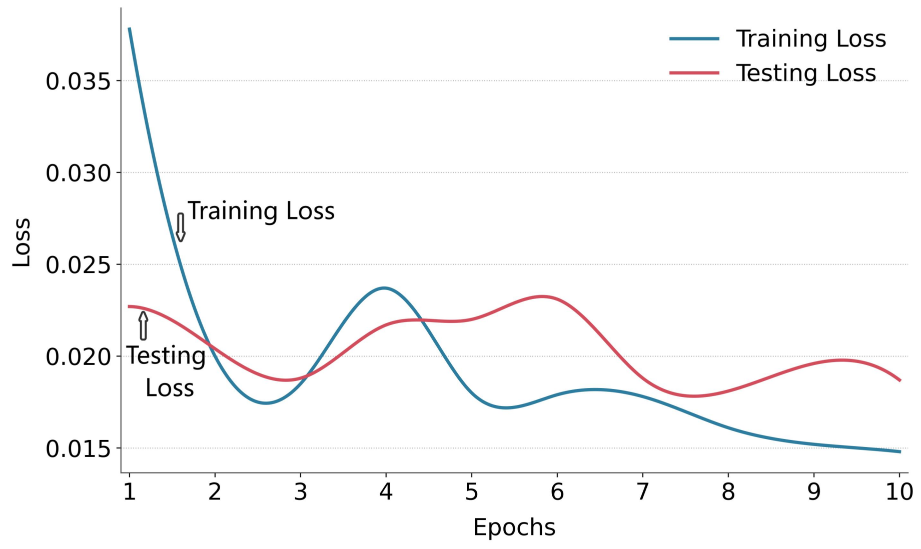

Fig. 10. Loss of the Proposed Model on Dataset 2

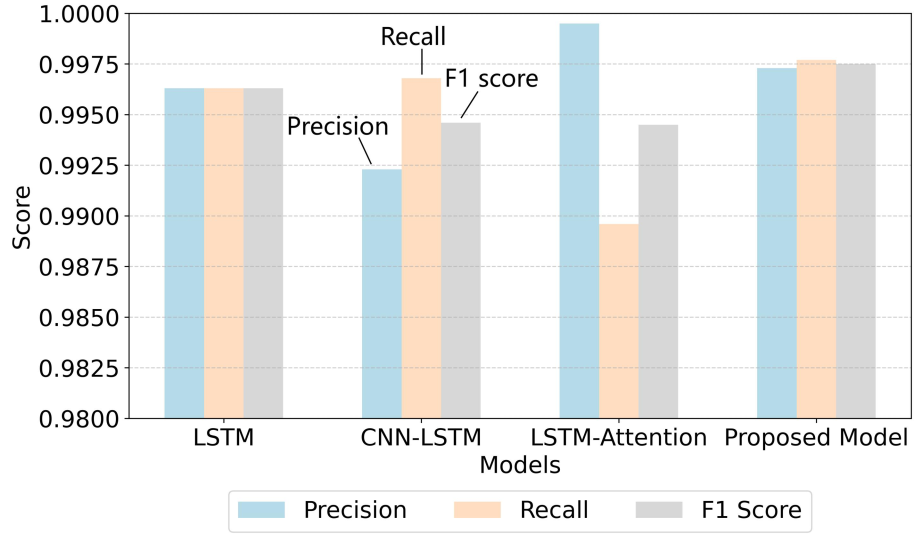

Fig. 11. Comparison of the proposed model with other models on Dataset 1

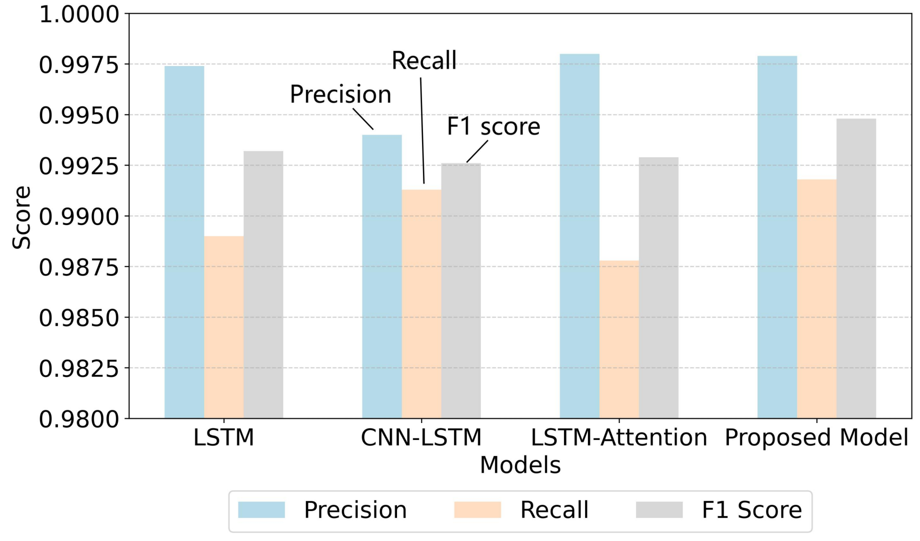

Fig. 12. Comparison of the proposed model with other models on Dataset 2

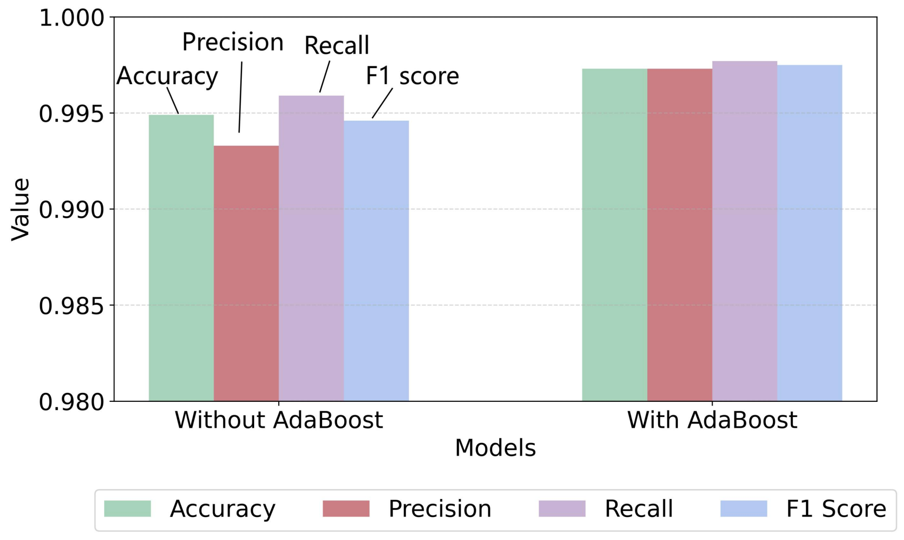

Fig. 13. Comparison of model results without and with AdaBoost on Dataset 1

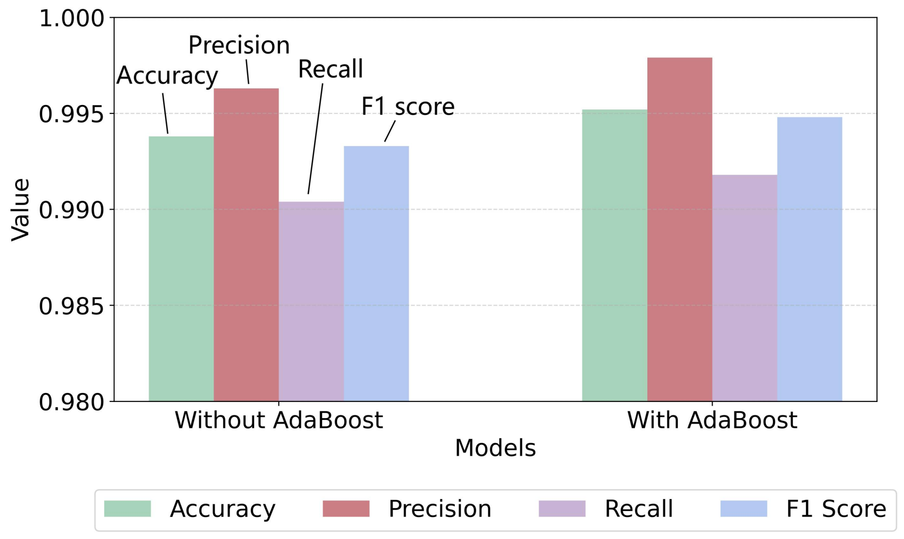

Fig. 14. Comparison of model results without and with AdaBoost on Dataset 2

%, recall of 99.77%, and an F1-score of 0.9975 on Dataset 1.

On Dataset 2, the model from [18] attains an accuracy of 99.36% and an F1-score of 0.9932; the model from [19] reaches an accuracy of 99.31% with an F1-score of 0.9926; and the model from [20] achieves an accuracy of 99.34% with an F1-score of 0.9929. In contrast, the proposed model achieves an accuracy of 99.52%, precision of 99.79%, recall of 99.18%, and an F1-score of 0.9948 on Dataset 2, again outperforming the other models.

The proposed model's outstanding performance in precision and recall across both datasets highlights its robustness and adaptability, making it highly effective for detecting XSS attacks in different contexts.

The superiority of the proposed model was demonstrated across various metrics in the experiments above.To further evaluate the contribution of each module, an ablation experiment was conducted, with a particular focus on the AdaBoost module. In this experiment, the AdaBoost module was removed while keeping all other components intact. This setup enabled the quantification of AdaBoost's impact on model performance and provided a more detailed analysis of its contribution to the evaluation metrics.

Fig. 13 and Fig. 14 show the performance of AdaBoost with two different datasets, respectively. The figures illustrate that the AdaBoost algorithm significantly enhances the model's performance across various datasets, improving accuracy, precision, recall, and F1-score. It shows that the integration of AdaBoost algorithm significantly enhances the ability of the model to identify positive class samples.

Figure. 15 presents the confusion matrix of the model evaluated on Dataset 1 without the integration of AdaBoost, showing 2,154 TP, 8 FN, 13 FP, and 1,921 TN.

Figure. 16 illustrates the results following the integration of AdaBoost, where TP increases to 2,206, FN decrease to 6, FP drop to 5, and TN exhibit only slight variation.

Figure. 17 displays the confusion matrix for Dataset 2 in the absence of AdaBoost, yielding 10,270 TP, 33 FN, 87 FP, and 8,938 TN.

Fig. 18 demonstrates the outcome following AdaBoost integration on Dataset 2, with TP increasing to 10,291, FN dropping to 19, FP declining to 74, and TN improving to 8, 945.

Taken together, the results indicate that AdaBoost enhances the model's classification performance across both datasets. Through adaptive weight adjustment, AdaBoost allows the model to prioritize boundary and misclassified instances, reducing false positives and missed detections, while improving overall accuracy. As a result, AdaBoost shows exceptional robustness and broad applicability in various attack detection scenarios, particularly when handling complex and diverse attack patterns. Its adaptive focus on misclassified instances enhances the model's effectiveness, making it an essential tool for improving detection accuracy in real-world applications.

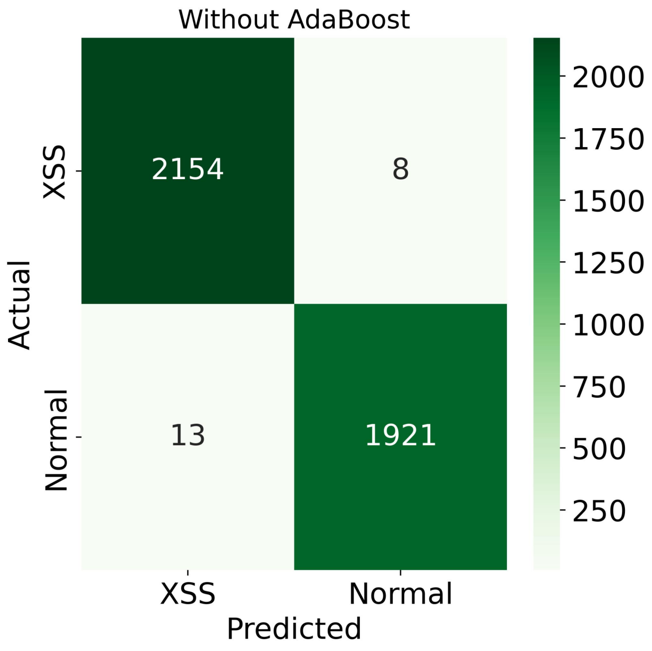

Fig. 15. The confusion matrix of the model without AdaBoost on Dataset1

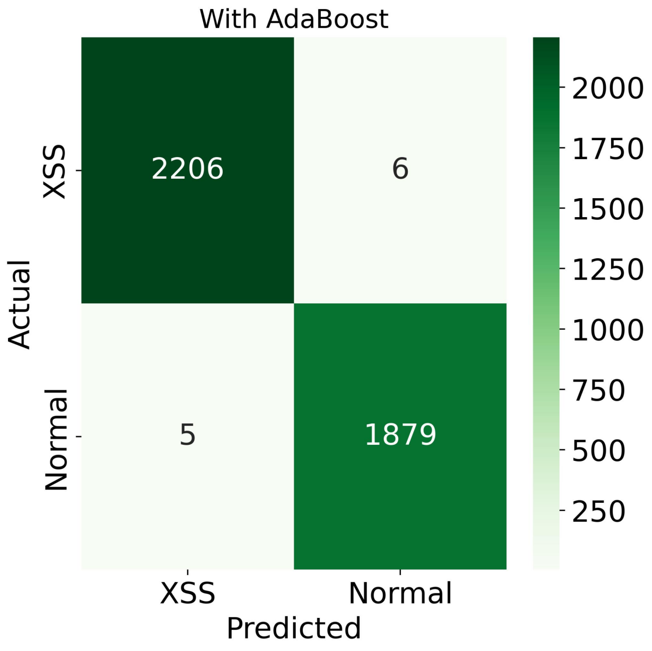

Fig. 16. The confusion matrix of the model with AdaBoost on Dataset1

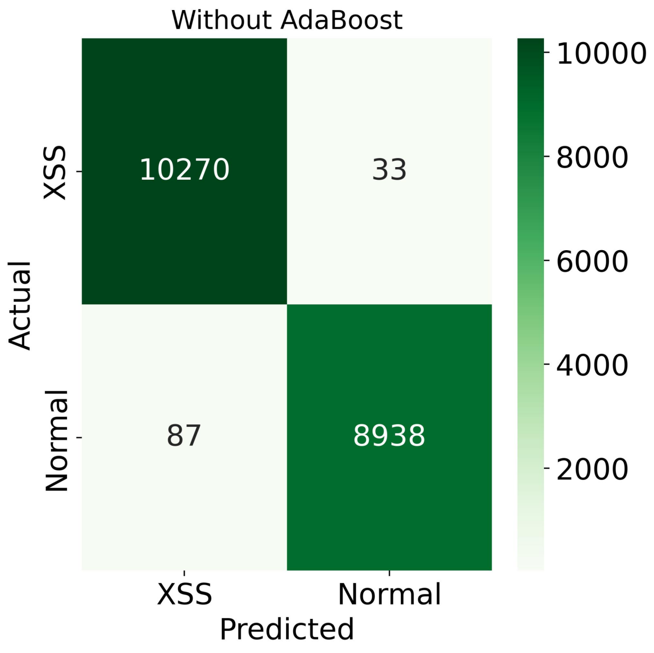

Fig. 17. The confusion matrix of the model without AdaBoost on Dataset2

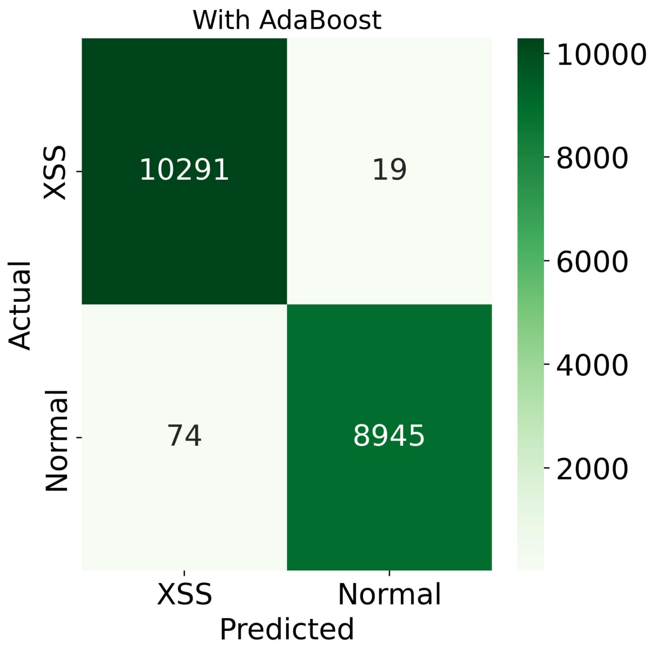

Fig. 18. The confusion matrix of the model with AdaBoost on Dataset2

VI. Conclusion

An XSS attack detection model that integrates deep learning with the AdaBoost ensemble learning algorithm is proposed in this paper. Initially, a Word2Vec model is utilized to generate word embeddings from the text data, thereby providing a robust and high-quality representation of the word vectors. A preprocessing step is conducted to verify the data's validity and ensure its consistency. The trained word vectors are then input into the deep learning model for feature extraction and classification. The output of the deep learning model is subsequently optimized using the AdaBoost classifier to enhance classification performance. This approach leverages the strengths of both deep learning and ensemble learning, resulting in improved detection accuracy. Extensive experiments demonstrate that the proposed model achieves accuracy rates of 99.73% and 99.52% on two distinct datasets, highlighting its superior efficacy in detecting XSS attacks.

REFERENCES

[1] OWASP Top Ten Web Application Security Risks | OWASP. (n.d.). Retrieved February 23,2021.

[2] Wathiq Laftah AI-Yaseen, ‘‘Improving Intrusion Detection System by Developing Feature Selection Model Based on Firefly Algorithm and Support Vector Machine,” IAENG International Journal of Computer Science, vol.46, no.4, pp534-540, 2019.

[3] Wei-Zhong Sun, Jie-Sheng Wang, Bo-Wen Zheng, and Zhong-Feng Li,“A Novel Convolutional Neural Network Voiceprint Recognition Method Based on Improved Pooling Method and Dropout Idea ” IAENG International Journal of Computer Science, vol.48, no.1, pp202-212, 2021.

[4] J. Kaur, U. Garg, and G. Bathla, "Detection of crosssite scripting (XSS) attacks using machine learning techniques: a review," Artificial Intelligence, 2023

[5] M. Liu, B. Y. Zhang, W. B. Chen, and X.L. Zhang, "A survey of exploitation and detection methods of XSS vulnerabilities," IEEE Access, vol. 7, pp. 182004–182016, 2019.

[6] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TTXSS: A novel taint tracking based dynamic detection framework for DOM CrossSite Scripting," Journal of Parallel and Dis tributed Computing, vol. 11-8, pp. 100–106, 2018.

[7] P. M. D. Nagarjun and S. S. Ahamad, "Ensemble methods to detect XSS attacks," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 11, no. 5, pp. 695–703, 2020.

[8] Gupta, Shashank, and B. B. Gupta. "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art." International Journal of System Assurance Engineering and Management, vol. 8, suppl. 1, 2017, pp. S512–S530.

[9] D. Bates, A. Barth, and C. Jackson, “Regular expressions considered harmful in client-side xss filters,” in Proceedings of the 19th international conference on World wide web, 2010, pp. 91–100.

[10] G. Usha, S. Kannimuthu, P. Mahendiran, A. K. Shanker, and D. Venugopal, “Static analysis method for detecting cross site scripting vulnerabilities,” International Journal of Information and Computer Security,vol. 13, no.1, pp. 32–47, 2020.

[11] F. Duchene, R. Groz, S. Rawat, and J.L. Richier, “Xss vulnerability detection using model inference assisted evolutionary fuzzing,” in 2012 IEEE Fifth International Conference on Software Testing, Verificationand Validati on. IEEE, 2012, pp. 815–817.

[12] S. Bensalim, D. Klein, T. Barber, and M. Johns, “Talking about my generation: Targetd dom-based xss exploit generation using dynamicdata flow analysis,” in Proceedings of the 14th European Workshop on Systems Security, 2021, pp. 27–33.

[13] B. Vishnu and K. Jevitha, “Prediction of cross-site scripting attack using machine learning algorithms,” in Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing, 2014, pp. 1–5.

[14] Nagarjun, PMD, and Shaik Shakeel Ahamad. "Ensemble Methods toDetect XSS Attacks." International Journal of Advanced Computer Science and Applications, vol. 11, no. 5, 2020, pp. 695.

[15] G. Habibi and N. Surantha, “Xss attack detection with machine learning and n-gram methods,” in 2020 International Conference on Information Management and Technology (ICIMTech). IEEE, 2020, pp.516–520.

[16] W. Melicher, C. Fung, L. Bauer, and L. Ji a, “Towards a lightweight,hybrid approach for detecting dom xss vulnerabilities with machine learning,” in Proceedings of the Web Conference 2021, 2021, pp. 26-84–265.

[17] S. Saleem, M. Sheeraz, M. Hanif, and U.Farooq, “Web server attack detection using machine learning,” in 2020 International Conference on Cyber Warfare and Security (ICCWS).IEEE, 2020, pp. 1–7.

[18] Hakim, Naufal Ahmad Nur, Vera Suryani, and Muhamad Irsan. "Detection of Cross-Site Scripting Attacks on Web Applications Using the LSTM Method." 2024 12th International Conference on Information and Communication Technology (ICoICT), IEEE, 2024, pp. 432-437.

[19] Kadhim, Raed Waheed, and Methaq Talib Gaata. "A hybrid of CNN and LSTM methods for securing web application against cross-site scripting attack." Indonesian Journal of Electrical Engineering and Computer Science, vol. 21, no. 2, 2021, pp. 1022-1029.

[20] Et-tolba, Maryam, Charifa Hanin, and Abdelhamid Belmekki. "DL - based XSS Attack Detection Approach using LSTM Neural Network with Word Embeddings." 2024 IEEE, 2024, pp. 979-8-3503-7786-6.

[21] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning, vol. 1. Cambridge: MIT press; 2016.

[22] Alzubaidi, Laith, et al. "Review of Deep Learning: Concepts,CNN Architectures, Challenges, Applications, Future Directions." Journal of Big Data, vol. 8, no. 1, 2021, pp. 53.

[23] R. S. Srinivasamurthy, “Understanding 1D convolutional neural networks using multiclass time-varying signalss,” PhD Thesis, Clemson University,2018.

[24] Hochreiter, Sepp, and Jürgen Schmidhuber." Long short-term memory."Neural computation 9.8 (1997): 1735-1780.

[25] A. Vaswani et al., “Attention is all you need.” arxiv, Dec. 05, 2017. [Online].Available: http://arxiv.org/abs/1706.03762.

[26] Y. Freund and R. E. Schapire, ‘‘A short introduction to boosting,’’ inProc.16th Int. Joint Conf. Artif. Intell., 1999, pp. 1401–1406.

[27] Mienye, Ibomoi D., and Yanxia Sun. "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects." IEEE Access, vol. 10, 2022, pp. 99129-99149.

[28] "Dataset."[Onlne].Available:https://www.kaggle.com/code/syedsaqlainhussain/cross-site-scripting-attack-detection-using-cnn/input. Accessed: Mar. 15, 2024.

[29] SparkSharly. "DL_for_xss Dataset." GitHub, GitHub, 2025, https://github.com/SparkSharly/DL_for_xss/tree/master/data.

[30] J. Liang, " Confusion matrix: machine learning, " POGIL Activity Clearinghouse, vol. 3, no. 4, 2022.