IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 14, No. 6, December 2025, pp. 4923~4932
ISSN: 2252-8938, DOI: 10.11591/ijai.v14.i6.pp4923-4932

Enhancing cross-site scripting attack detection by using FastText as word embeddings and long-short term memory

Muhammad Alkhairi Mashuri[1], Nico Surantha[1,2]
[1]Master of Computer Science, BINUS Graduate Program, Bina Nusantara University, Jakarta, Indonesia
[2]Department of Electrical, Electronic and Communication Engineering, Faculty of Engineering, Tokyo City University, Tokyo, Japan

Article Info

Article history:

Received Aug 20, 2024
Revised Jul 7, 2025
Accepted Aug 6, 2025

Keywords:

Cross-site scripting
Cyber security
Deep learning
FastText
Long-short term memory
Word embedding

ABSTRACT

Cross-site scripting (XSS) is one of the dangerous cyber-attacks and the number of attacks continues to increase. This study takes a new approach to detect attacks by utilizing FastText as word embedding, and long-short term memory (LSTM), which aims to improve the performance of deep learning. This method is proposed to capture the broader meaning and context of the data used, leading to better feature extraction and model performance. This study not only improves the detection of XSS attacks, but also highlights the potential for better text processing techniques. The results obtained showing this method achieves higher results than other methods, with an accuracy of 99.89%.

Corresponding Author:

Muhammad Alkhairi Mashuri
Master of Computer Science, BINUS Graduate Program, Bina Nusantara University
Jakarta, Indonesia
Email: m.mashuri@binus.ac.id

1. INTRODUCTION

The growing use of web applications has resulted in a rise in security vulnerabilities, with cross-site scripting (XSS) attacks posing a major risk to data integrity and privacy. These attacks take advantage of client-side scripting weaknesses, making web applications a frequent target [1]. The consequences of a successful XSS attack can include session hijacking, sensitive data exposure, and impersonation of the victim and can even lead to code execution on the server, depending on the application and user account privileges [2]. XSS attacks can be generally divided into three groups: persistent, non-persistent, and document object model (DOM) based attacks [3]. Non-persistent attacks are also called reflected XSS, which are attacks carried out by inserting malicious scripts into the uniform resource locator (URL). While persistent attacks are also called stored XSS, which are attacks that insert malicious scripts into system files, databases, or other locations managed by the server and will be displayed to users. DOM based XSS is an attack that changes the DOM environment.

The open web application security project (OWASP) states that XSS is one of the top ten vulnerabilities on websites and XSS is ranked seventh, in addition it is noted that vulnerabilities to XSS are in about two-thirds of all web applications. In 2021, XSS is included in one of the attacks in the injection category which is ranked third [4]. In an XSS attack, attackers exploit vulnerabilities by injecting malicious scripts into hypertext markup language (HTML) webpages. When users access these compromised pages, the browser executes the malicious code, allowing attackers to hijack web sessions. Recently, the prevalence of XSS threats has grown significantly, drawing increasing attention from both industry and academia. This has made XSS vulnerabilities a key focus in cybersecurity research, with numerous studies dedicated to enhancing detection and mitigation strategies. Consequently, the rising severity of XSS attacks underscores the urgent need for robust defense mechanisms [5]. XSS can also be exploited in conjunction with other vulnerabilities, such as cross-site request forgery (CSRF), and remote code execution (RCE), potentially leading to more significant threats and harm to the victim's internal network [6].

In March 2024, the SecureList by Kaspersky website published a report with research data from 2021 to 2023 stating that XSS vulnerabilities were found on 61% of the analyzed websites [7]. XSS ranked 2nd out of the 25 most dangerous software vulnerabilities in 2023 [8], and has received the same and higher ranking in previous years by taking first place in 2020 with the highest overall score in prevalence and severity. According to Cisco's 2018 annual security report, it shows that all web applications that they analyzed is having at least one vulnerability. The report also showing the web attacks are becoming more frequent, more specialized, and more technically sophisticated. Additionally, 40% of all attack attempts involved lead to a technique known as XSS, making it one of the most widely used technique [9].

A variety of prevention and mitigation methods have been proposed to combat an existing XSS attacks, which were classified into the following categories: machine learning technique, client-size technique, proxy-side technique, server-size technique, and combined technique [10]. However, the proposed solutions are becoming insufficient due to the increasingly sophisticated form of XSS payloads over time [11], resulting in non-negligible false positive cases. XSS became the most widespread form of attack vector in 2019. Research by precise security [12] states that XSS accounted for almost 40% of all attacks recorded by security experts. Around 75% of companies across North America and Europe have also been targeted by this attack during 2019. In addition, the National Vulnerabilities Database also stated that the total number of XSS attacks in 2019 increased by 30.2% compared to 2018, and 79.2% compared to 2017. The threat of XSS attacks is widespread, resulting in several major incidents. Like in 2018, British Airways was compromised Magecart, a hacker group specializing in credit card theft. The attackers exploited a XSS vulnerability in the Javascript library that utilized on the British Airways website, it enabling the attacker to use the script to exfiltrate customer data to the attacker' remote server, this breach led to the theft orf credit cards information from approximately 380,000 booking transactions before it was detected. Similarly in 2019, Fortnite, a multiplayer game with more than 200 million users, was found to have an XSS vulnerability on an unsecured webpage. Furthermore, in early 2016, eBay identified a fatal XSS vulnerability caused by improperly validated URL parameter, which allowed attacker gain full access to the seller account, conduct unauthorized transactions, and stealing payment information details [13].

Several studies have been conducted using artificial intelligence (AI) to perform detection, such as the use of machine learning and deep learning. Deep learning can be applied as one of the efforts to prevent XSS attacks. Deep learning has higher capabilities and flexibility in processing a number of features in unstructured data. In algorithms in deep learning, data is processed through multiple layers, where each layer is able progressively extract and refines features before passing them on to the next layer. Various architectures can be used to implement deep learning, including unsupervised pre-trained networks, convolutional neural networks (CNN), recurrent neural networks (RNN), and recursive neural networks [14]. However, although AI can significantly solve this problem, there are still fundamental shortcomings such as the false negative ratio. False negative is a bigger problem than false positive, because many attacks try to avoid the detection system, which ultimately results in real threats and compromised security systems [15].

Many of studies using the Word2Vec word embedding method. Word2Vec is an algorithm that maps each word in a text into a vector form. This algorithm was introduced in 2013 with two main methods, namely skip-gram and continuous bag of words (CBOW) [16]. Until now, this word embedding model has been widely used in natural language processing (NLP) research. The Word2Vec method generates word embeddings using a dense representation. As a predictive model, it assigns probabilities to words, making it highly effective in word similarity tasks [17]. However, Word2Vec is unable to learn syntactic relationships, the structure of Word2Vec proves to be extremely dependent on the corpus, making semantic proximity only a side effect of its true objective function [18].

In addition to Word2Vec, there are other word embedding techniques, namely GloVe which was introduced in 2014 and FastText which was introduced in 2017. GloVe is different from Word2Vec which only relies on local information from words with local context windows (skip-gram and CBOW), GloVe also combines word co-occurrence information or global statistics to obtain semantic relationships between words [19]. While FastText is a development of Word2Vec, this method learns words by considering information from syllables. Each word is represented as a set of n-gram characters. This helps in the sense of shorter words and allows embedding to understand the suffix and prefix of the word [20].

FastText has the advantage of being able to provide a representation of words that do not appear in the training data, or being able to overcome the problem of out of vocabulary. If it does not find a word that is not in the training process, then the word will be broken down into n-grams in the form of a collection of syllable sequences to get its embedding vector. FastText has good performance, and can train models on large datasets quickly, and is able to provide a representation of words that do not appear in the training data.

There are several previous studies related to XSS detection. In general, the solutions offered are similar to each other. Research by Bakir and Bakir [21], conducted XSS attack detection using several machine learning and deep learning models, combined with Word2Vec word embedding and universal sentence encoder (USE). The results of deep learning and Word2Vec, long-short term memory (LSTM) obtained an accuracy of 93.94%, and an accuracy of 97.66% with CNN. The results of deep learning and USE and Word2Vec, obtained an accuracy of 98.47% with LSTM, and 99.16% with Vanilla NN 2.

Research conducted by Tadhani *et al.* [22], proposed a solution using CNN and LSTM to detect SQL injection and XSS. The process begins with preprocessing input data, which includes decoding, tokenization, and generalization techniques. Then, the processed data is fed into the CNN model for feature extraction, and the extracted features are used for training the LSTM model. Data is taken from various sources, such as the National Vulnerability Database, and also OWASP. The Word2Vec model is used for feature extraction from input data. The results of this study, the CNN model has an accuracy of 99.5%, and LSTM 98.69%, and CNN with LSTM gets an accuracy of 99.84% with its own testbed dataset. The results of this study also suggest that further studies can look at other architectures and methods to combine CNN and LSTM models to improve model precision and robustness.

In a study conducted by Younas *et al.* [13] trained a set of 13,686 records data from Kaggle, presenting an approach that use machine learning and deep learning that includes LSTM to create early detection of XSS attacks, by comparing and applying term frequency-inverse document frequency (TFIDF) to evaluate the results. LSTM-TFIDF for feature extraction, using both temporal features and TFIDF from the XSS dataset, resulting in a new set of features. There are also several algorithms used in this study, such as random forest (RF), logistic regression, Gaussian naive Bayes (GNB), decision tree (DT), gated recurrent unit. With feature extraction from LSTM-TFIDF, an accuracy of 74% was obtained with the LSTM model. To perform data extraction, the study conducted with the bag of words (BoW) technique for feature extraction. By applying these BoW features, the results of GNB, DT, and RF getting weak performance scores. BoW limitation is largely due to inability to capture high-level or complex features, resulting in low performance scores for many methods. For further improvement, alternative exploration of feature extraction techniques is needed.

Next, there is a study conducted by Alaoui and Nfaoui [23], this study proposes an approach based on Word2Vec as word embedding, and a stacked generational ensemble model for LSTM to detect malicious HTTP Requests. Word2Vec is a two-layer neural network that takes a text corpus as input and outputs a set of vectors. It includes two models, CBOW and skip-gram. This paper uses the CBOW model because it is usually faster and more accurate than the skip-gram model. This study modifies the training parameter values, namely window size and embedding size, to obtain different vector representations of words with the same word. With the stacking ensemble of LSTM models, the highest accuracy result is 78.95%.

There is also a study conducted by Gogoi *et al.* [2] that proposes XSS attack detection using the support vector machine (SVM) algorithm. This study uses TFIDF to perform feature extraction from tokenized text. With linear and non-linear SVM, the classification results get a precision of 0.97, recall 0.99, and F1 score 0.97.

Then there is another study conducted by Lente *et al.* [24], this study combines CNN with LSTM, and achieves 99.36% accuracy when predicting whether a new URL corresponds to an XSS attack. The dataset consists of samples of 33,426 XSS attacks extracted from the XSSed database at http://www.xssed.com/ and 31,407 regular URLs extracted from the DMOZ database at http://dmoztools.net/. This study consists of four main steps: processing input data, Word2Vec transform, convolution, and LSTM. Based on previous research, this study proposes a XSS attack detector that uses deep learning with the LSTM method which is a type of RNN. LSTM stores information about data patterns and can also learn which data should be kept or discarded, because each LSTM neuron has several gates that regulate the memory of each neuron itself [25]. The feature extraction method used is FastText which aims to provide innovation in script features. FastText extends Word2Vec by including subword information. This feature represents words as a collection of n-gram characters, allowing it to produce embeddings that capture morphological variations and handle rare or misspelled words more effectively. This feature is very useful for XSS detection, because attackers often use creative obfuscation techniques in scripts to bypass security filters. So that it can increase the accuracy in detecting XSS attacks. With this method, it is expected to increase the accuracy in detecting XSS attacks in this study.

2. METHOD

The research stage begins with data collection. XSS url data is taken from Kaggle, which consists of 13,686 data that have been labeled as XSS attacks and non-XSS attacks. The data is taken from PortSwigger, and OWASP Cheat Sheets for XSS attacks, which consists of 7,373 XSS attacks, and 6,313 non-XSS attacks.

Then continued with the preprocessing stage, the dataset is cleaned from noise or things that have no effect. The processes that will be carried out are decode, generalization, and tokenization. Furthermore, the results of the preprocessing will be word embedding using FastText. The model that will be used for classification is LSTM, and then all the results will be analyzed to measure the performance and accuracy level of the text classification model using LSTM.

2.1. Dataset

The dataset used in this study uses a dataset for deep learning published on the Kaggle repository at https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning. The dataset consists of 13,686 entries collected from PortSwigger and OWASP Cheat Sheets for XSS attacks. This dataset already includes a variety of XSS attack vectors that allow this study to assess the effectiveness of the model used in detecting various attack scenarios. There are 2 labels in the dataset, namely XSS and not XSS. Of the 13,686 entries, there are 7,373 entries that are XSS attacks, and 6,313 that are not XSS attacks, or 53.8% XSS attacks and 46.2% not XSS attacks. Samples of the data can be seen in Table 1.

Table 1. Sample dataset

Sentence	XSS
<div draggable="true" contenteditable>drag me</div><header ondrop=alert(1) contenteditable>drop here</header>	1
<ins oncut="alert(1)" contenteditable>test</ins>	1
</div> </td> </tr><tr><th scope="row" class="navbox-group" style="width:1%"><a href="/wiki/Algorithm" title="Algorithm">Algorithms </a> </th><td class="navbox-list navbox-even hlist" style="text-align:left;border-left-width:2px;border-left-style:solid;width:100%;padding:0px">	0
</span><link rel="mw-deduplicated-inline-style" href="mw-data:TemplateStyles:r935243608"/>	0
<summary ondblclick="alert(1)">test</summary>	1

2.2. Preprocessing

Preprocessing is the first stage carried out after obtaining data, and it plays a critical role in preparing the raw input for analysis. This stage consists of three key steps: decoding, generalization, and tokenization, which are explained in the following subsections. Each step serves to clean, transform, and structure the data, ensuring it is in a suitable format for the model to process effectively. Proper preprocessing helps minimize noise, reduce irrelevant information, and maintain consistency across the dataset. This not only enhances the performance of machine learning models but also ensures that important features related to the detection of XSS attacks are preserved and highlighted during the training phase.

2.2.1. Decoding

Attackers can avoid filters or validation based on regular expressions by using encoding techniques such as Hex encoding, URL encoding, unicode encoding, and HTML entity encoding. So decoding is needed to restore the original value. The Table 2 is an example of a comparison of data before and after decoding from the dataset used. The data uses encoding in inserting script tags, and is successfully returned to the actual script tag form after decoding.

Table 2. The comparison of decoded results

Original text	Decoded text
<A HREF="http://%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D"></A>	<A HREF="http://www.google.com"></A>

2.2.2. Generalization

This process is carried out after the decoding process, with the aim of reducing interference from redundancy and irrelevant information. All parts such as domains, IP addresses, and extensions contained in the URL are replaced to ensure a uniform and standardized format. Generalization helps to strip away unnecessary details that could introduce noise into the analysis, allowing the model to focus on the core structure and behavior of the URL or script. This step is particularly important in detecting patterns common in malicious URLs, as it reduces variability and highlights suspicious patterns that might otherwise be overlooked. By doing so, generalization enhances the model's ability to detect XSS attacks more accurately during subsequent stages of preprocessing and feature extraction. The results of generalization on one of the data that has gone through the decoding process can be seen in Table 3. Domain is one of the parts that needs to be generalized from the data from the results of the decoding process. And after the generalization process is carried out, "http://www.google.com" is changed to "domain".

Table 3. The comparison of generalization results

Original text	Decoded text
<A HREF="http://www.google.com"></A>	<A HREF="domain"></A>

2.2.3. Tokenization

This stage helps standardize text by replacing multiple forms of the same tag with consistent tokens. Tokenization reduces the complexity of text by converting it into a form that is easier to analyze and process. It ensures that similar patterns, such as variations in syntax or case, are treated uniformly, enabling the model to focus on content rather than formatting differences. Additionally, tokenization plays a crucial role in handling complex data like URLs or scripts, breaking them down into manageable components. After this stage, the tokenized data will be used for the word embedding process using FastText, which captures contextual relationships between tokens for downstream tasks such as classification or prediction.

Table 4 explains the comparison of data before and after the tokenization process. The generalized text shows that the opening and closing tags are still in one unit, and after the tokenization process, the opening and closing tags are separated. At this stage, the attributes on the tag are maintained because attributes can contain XSS attacks, such as alert, Javascript, and eval.

Table 4. The comparison of tokenization results

Original text	Decoded text
<A HREF="domain"></A>	<A HREF="domain" > </A>

2.3. Word embedding

Word embedding is a set of NLP models that represent words in the form of low-dimensional vectors called embeddings, with the aim of improving the performance of machine learning networks in textual analysis of data sets. This technique transforms text data into numeric representations that can be used by machine learning models. In FastText, each word is represented as the average of the n-gram character vector representations and the word itself. For example, in the word “equal” and n=3, the word will be represented by the n-gram characters: <eq, equ, qua, ual, al> and <equal>. So, the word embedding for the word ‘equal’ can be given as the sum of all the vector representations of all the n-gram characters and the word itself. FastText can also apply hyperparameter tuning. In this study, the hyperparameter values described in Table 5.

Table 5. Hyperparameter configurations for word embedding

Parameter	Value
Model	CBOW, skip-gram
Dimension	100, 150
Learning rate	0.05, 0.1
Epoch	50, 100

2.4. Long-short term memory

The LSTM works by using units consisting of several gates, namely forget gate, input gate, cell gate, and output gate. The detection model used includes LSTM layer, dropout layer, and softmax output layer. LSTM layer is the core layer of this model detection, dropout layer is chosen to reduce overfitting problems, and softmax output layer for XSS attack prediction. The LSTM model will use a hidden layer size of 128, dropout 0.2, and the final output layer is activated using sigmoid. For the loss function, it will use binary cross-entropy. Details of the hyperparameters used can be seen in Table 6.

Table 6. Hyperparameter configurations for LSTM

Parameter	Value
Epochs	5, 7, 9, 10
Batch size	32, 64
Validation split	0.2
Hidden layer size	128
Dropout rate	0.2
Dense unit	32
Learning rate	0.001

2.5. Evaluation

In this study, the evaluation of the XSS attack detection model using deep learning is based on three key performance metrics: accuracy, recall, and F1 score. These metrics are chosen to comprehensively assess the model's ability to correctly identify malicious inputs while minimizing misclassification. The evaluation is supported by a confusion matrix that comprises four classes: true positive class representing the number of XSS samples correctly identified as XSS, false positive class indicating the number of non-XSS samples that are mistakenly classified as XSS, true negative class which includes the number of non-XSS samples correctly classified, and False Negative class representing XSS samples that are incorrectly classified as non-XSS. A high true positive rate indicates strong detection capability, while a low false negative rate is essential to avoid missing actual attacks. At the same time, reducing False Positives is important to prevent disrupting normal user inputs. By analyzing these metrics, we can determine how well the model can classify the XSS attacks.

3. RESULTS AND DISCUSSION

3.1. Dataset setting

The dataset is divided into three parts, consisting of training data, test data, and validation data. The separation into these three parts is done using a 60:20:20 portion, where 60% of the data is allocated for training, 20% for testing, and 20% for validation. This division is essential for ensuring that the model is properly trained, evaluated, and fine-tuned. The training data is used to teach the model, allowing it to learn patterns and relationships within the data. The test data provides an initial assessment of the model's performance, helping to identify any overfitting or underfitting issues. Finally, the validation data serves as an unbiased dataset to fine-tune the model's parameters, ensuring that the model generalizes well to unseen data. By maintaining this separation, the model's accuracy and robustness can be evaluated more effectively, leading to more reliable and trustworthy predictions in real-world applications.

3.2. Hyperparameter

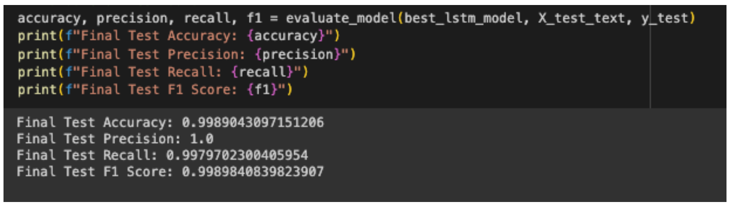
The results of the experiments highlight the importance of carefully selecting hyperparameters for both FastText and LSTM models in the context of XSS attack detection. Among the 128 hyperparameter combinations tested, the configuration with FastText using the CBOW model, a dimension of 150, a learning rate of 0.05, and 100 epochs, paired with an LSTM setup of 64 batch size and 7 epochs, yielded the highest accuracy of 99.89%. This combination consistently outperformed other configurations, demonstrating a balance between the richness of the word embeddings generated by FastText and the sequential learning capabilities of LSTM. The high accuracy, coupled with strong precision, recall, and F1 scores, reflects the robustness of the model in classifying XSS attacks. The results also emphasize that increasing the dimensionality of embeddings and adjusting learning rates can significantly impact model performance, particularly in cases where complex patterns such as malicious scripts need to be identified. Additionally, the validation and test results confirmed that the model generalizes well to unseen data, making it a reliable approach for practical XSS detection scenarios. This section also discusses the role of preprocessing steps, such as tokenization and generalization, which contributed to enhancing model performance by reducing noise and preserving key features in the data.

3.3. Evaluation

The evaluation results can be seen in the Figure 1, based on the prediction results, the FastText and LSTM models achieved an accuracy of 99.89%. This accuracy indicates that the model is able to correctly classify approximately 99% of the total testing data, demonstrating its high reliability in detecting XSS attacks. Furthermore, the model achieved a perfect precision score of 100%, meaning that all the samples predicted as XSS were indeed true XSS samples, with no false positives. The recall score reached 99.79%, indicating that nearly all actual XSS samples were successfully identified, with only a minimal number of false negatives. The F1 score, which represents the harmonic mean of precision and recall, stood at 99.89%, reflecting an optimal balance between the model's ability to detect XSS attacks and minimize misclassification. These results highlight the effectiveness of combining FastText word embeddings with an LSTM in capturing the patterns of XSS payloads.

From the confusion matrix results in Figure 2, it can be seen that there are 1,260 data that are not XSS attacks and are classified correctly or true positive. There is no data that is not an XSS attack and is classified as an XSS attack or false negative. Then there are 3 XSS attack data that are classified as not an XSS attack or false positive; and 1,475 XSS attack data that are classified as XSS, or true negative. These results indicate a high level of accuracy in detecting both XSS and non-XSS attacks, with minimal misclassification. The low number of False Positives and absence of false negatives demonstrates the model's effectiveness in minimizing classification errors, which is crucial for real-world XSS detection applications.

The strong performance in correctly classifying malicious attacks highlights the reliability of the model for security tasks, ensuring that true threats are identified without generating unnecessary alerts.



```
accuracy, precision, recall, f1 = evaluate_model(best_lstm_model, X_test_text, y_test)
print(f"Final Test Accuracy: {accuracy}")
print(f"Final Test Precision: {precision}")
print(f"Final Test Recall: {recall}")
print(f"Final Test F1 Score: {f1}")

Final Test Accuracy: 0.9989043097151206
Final Test Precision: 1.0
Final Test Recall: 0.9979702300405954
Final Test F1 Score: 0.9989840839823907
```

Figure 1. Classification result

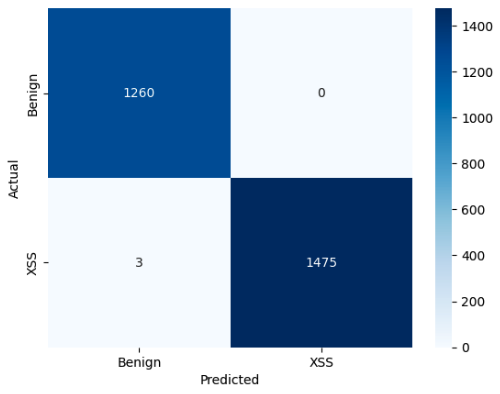


Figure 2. Confusion matrix result

Figure 3 shows the model accuracy across training and validation. The training accuracy increases steadily from around 0.985 to 0.998, indicating the model is learning effectively from the training data. The validation accuracy then shows a similar upward trend but with some fluctuations. It starts at around 0.996 and peaks around 0.998 before dropping slightly. This indicates the model is generalizing well, and likely experiencing a bit of overfitting towards the end of the validation process.

Figure 4 shows the loss model. The significantly decreasing training line indicates that the model makes fewer errors on the training data as the number of epochs increases, suggesting effective learning and convergence during training. Similarly, the validation loss line also shows a decreasing trend, which reflects that the model is generalizing well. However, there are still some fluctuations in the validation line, which may reflect sensitivity to specific validation samples or slight overfitting during certain epochs. Despite this, the overall downward trend in both training and validation loss suggests that the model remains stable and performs well. This behavior further supports the robustness of the FastText and LSTM architecture used in the experiment.

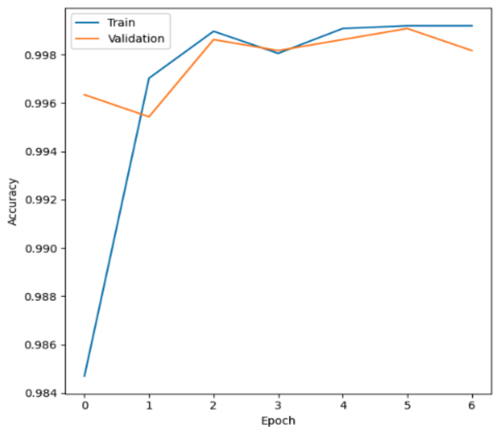


Figure 3. Model accuracy

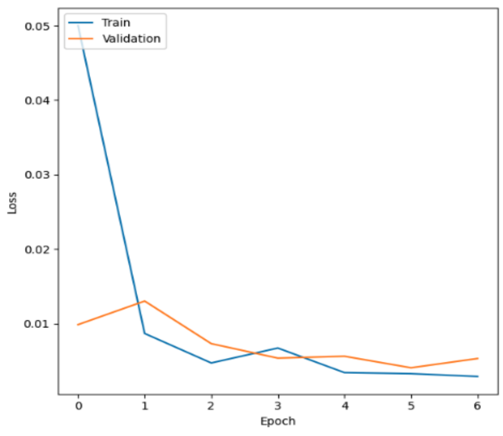


Figure 4. Model loss

3.4. Comparison with other methods

Referring to previous research [21] that also conducted research on machine learning, deep learning and word embedding using Word2Vec and USE, and also using the same dataset. The accuracy obtained in this study is quite far increased when compared to detection using other methods in previous studies. Detailed results can be seen in Table 7.

Table 7. The comparison with other models

Metode	Word embedding	Accuracy (%)	F1 score	Recall	Precision
Vanilla NN 1	Word2Vec	96.64	0.9683	0.9880	0.9493
Vanilla NN 2	Word2Vec	97.26	0.9751	0.99558	0.9953
GRU	Word2Vec	93.61	0.9437	0.8999	0.9919
CNN	Word2Vec	97.66	0.9788	0.9603	0.9980
LSTM	Word2Vec	93.94	0.9435	0.9499	0.9371
Vanilla NN 1	Word2Vec and USE	99.12	0.9919	0.9939	0.9899
Vanilla NN 2	Word2Vec and USE	99.16	0.9922	0.9899	0.9946
GRU	Word2Vec and USE	94.49	0.9497	0.9838	0.9684
CNN	Word2Vec and USE	99.05	0.9912	0.9351	0.9878
LSTM	Word2Vec and USE	98.47	0.9858	0.9946	0.9878
LSTM (proposed)	FastText (proposed)	99.89	0.9989	0.9979	1

LSTM combined with Word2Vec achieved an accuracy of 93.94%, while the combination of LSTM, Word2Vec, and the USE reached 98.47% accuracy. In this study, LSTM paired with FastText achieved a notably higher accuracy of 99.89%, outperforming both previous methods. This improvement highlights the superiority of FastText in capturing word relationships, especially for tasks like XSS attack detection, where understanding subtle variations in input patterns is crucial. FastText's ability to model subword information gives it a significant advantage over Word2Vec, which only considers whole words, and even the USE, which operates at a sentence level but might miss intricate details at the token level.

Additionally, the performance gain using FastText demonstrates its ability to handle out-of-vocabulary words more effectively, which is particularly important in the dynamic and evolving nature of web security, where new forms of XSS attacks continuously emerge. By leveraging this embedding technique, the model in this study not only achieved higher accuracy but also demonstrated better robustness and generalization. This suggests that FastText, when combined with LSTM, offers a more reliable solution for real-time XSS detection compared to traditional embedding methods, positioning it as a more suitable choice for future research and practical implementations in cybersecurity.

4. CONCLUSION

From the results of the research conducted and comparisons with several other methods and word embeddings in classifying XSS attacks, the findings revealed that the LSTM and FastText combination achieved the highest performance in detecting attacks. After conducting several experiments with different combinations of hyperparameters, it was proven that the LSTM and FastText methods significantly strengthened the detection capabilities, achieving an impressive accuracy of 99.89%, recall of 99.79%, precision of 100%, and an F1 score of 99.89%. These results demonstrate not only the robustness and precision of the model but also its ability to effectively minimize false positives and false negatives, which is critical for practical security applications. The superior performance compared to other embedding techniques such as Word2Vec and USE further emphasizes the importance of selecting the right model and hyperparameter configurations. This study highlights that the combination of LSTM with FastText can be considered a highly effective approach for real-time XSS detection and offers a promising solution for improving web security frameworks. Future work could explore scaling this approach to handle more diverse attack patterns or integrate it into comprehensive threat detection systems for even broader applications.

FUNDING INFORMATION

This work was supported by BINUS Graduate Program – Master of Computer Science, Department of Computer Science, Bina Nusantara University.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Muhammad Alkhairi Mashuri	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			
Nico Surantha					✓					✓		✓	✓	✓

C	: **C**onceptualization	I	: **I**nvestigation	Vi	: **Vi**sualization
M	: **M**ethodology	R	: **R**esources	Su	: **Su**pervision
So	: **So**ftware	D	: **D**ata Curation	P	: **P**roject administration
Va	: **Va**lidation	O	: Writing - **O**riginal Draft	Fu	: **Fu**nding acquisition
Fo	: **Fo**rmal analysis	E	: Writing - Review & **E**diting		

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

This study does not involve individual personal data or identifiable human subjects. Therefore, informed consent was not required.

ETHICAL APPROVAL

This study did not involve human participants or animals, and therefore did not require ethical approval. All procedures were conducted in accordance with relevant institutional and national guidelines.

DATA AVAILABILITY

The data that support the findings of this study are openly available in Kaggle Repository at https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning.

REFERENCES

[1] A. A. Salama, E.-S. F. Aboelfotoh, H. E. Khalid, A. K. Essa, H. M. El-Bakry, and D. S. El-Morshedy, "Integrating neutrosophic logic with ASP.NET to prevent XSS attacks," *Neutrosophic Optimization and Intelligent Systems*, vol. 5, pp. 14–28, 2025, doi: 10.61356/j.nois.2025.5455.

[2] B. Gogoi, T. Ahmed, and H. K. Saikia, "Detection of XSS attacks in web applications: a machine learning approach," *International Journal of Innovative Research in Computer Science & Technology*, vol. 9, no. 1, pp. 1–10, 2021, doi: 10.21276/ijircst.2021.9.1.1.

[3] A. Alanda, D. Satria, and H. A. Mooduto, "Cross-site scripting (XSS) vulnerabilities in modern web applications," *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 270–276, 2024, doi: 10.1109/EECSI63442.2024.10776461.

[4] OWASP, "OWASP TOP Ten," *The Open Web Application Security Project*, 2021. (Accessed: Mar. 01, 2025). [Online]. Available: https://owasp.org/www-project-top-ten/

[5] G. H. Luu *et al.*, "XSShield: a novel dataset and lightweight hybrid deep learning model for XSS attack detection," *Results in Engineering*, vol. 24, 2024, doi: 10.1016/j.rineng.2024.103363.

[6] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: cross site scripting detection based on deep learning," *ACM International Conference Proceeding Series*, pp. 47–51, 2018, doi: 10.1145/3194452.3194469.

[7] SecureList by Kaspersky, "Top 10 web application vulnerabilities in 2021–2023," *Securelist*, 2023. (Accessed: Mar. 12, 2024). [Online]. Available: https://securelist.com/top-10-web-app-vulnerabilities/112144/.

[8] CWE, "2020 CWE Top 25," *Common Weakness Enumeration*, 2020. (Accessed: Mar. 1, 2025). [Online]. Available: https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html.

[9] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides, "Cross-site scripting (XSS) attacks and mitigation: A survey," *Computer Networks*, vol. 166, 2020, doi: 10.1016/j.comnet.2019.106960.

[10] S. J. Y. Weamie, "Cross-site scripting attacks and defensive techniques: A comprehensive survey," *International Journal of Communications, Network and System Sciences*, vol. 15, no. 8, pp. 126–148, 2022, doi: 10.4236/ijcns.2022.158010.

[11] Y. Wang *et al.*, "A comparison of word embeddings for the biomedical natural language processing," *Journal of Biomedical Informatics*, vol. 87, pp. 12–20, 2018, doi: 10.1016/j.jbi.2018.09.008.

[12] J. Ilic, "Cross-site scripting (XSS) makes nearly 40% of all cyber attacks in 2019," *Precise Security*, 2019. [Online]. Available: https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/.

[13] F. Younas, A. Raza, N. Thalji, L. Abualigah, R. A. Zitar, and H. Jia, "An efficient artificial intelligence approach for early detection of cross-site scripting attacks," *Decision Analytics Journal*, vol. 11, 2024, doi: 10.1016/j.dajour.2024.100466.

[14] A. Mathew, P. Amudha, and S. Sivakumari, "Deep learning techniques: an overview," *Advances in Intelligent Systems and Computing*, vol. 1141, pp. 599–608, 2021, doi: 10.1007/978-981-15-3383-9_54.

[15] F. M. M. Mokbal, W. Dan, W. Xiaoxi, Z. Wenbin, and F. Lihua, "XGBXSS: an extreme gradient boosting detection framework for cross-site scripting attacks based on hybrid feature selection approach and parameters optimization," *Journal of Information Security and Applications*, vol. 58, 2021, doi: 10.1016/j.jisa.2021.102813.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Neural information processing systems*, vol. 1, pp. 1–9, 2006.

[17] S. S. Birunda and R. Kanniga Devi, "A review on word embedding techniques for text classification," *Innovative Data Communication Technologies and Application*, vol. 59, pp. 267–281, 2021, doi: 10.1007/978-981-15-9651-3_23.

[18] G. Di Gennaro, A. Buonanno, and F. A. N. Palmieri, "Considerations about learning Word2Vec," *Journal of Supercomputing*, vol. 77, no. 11, pp. 12320–12335, 2021, doi: 10.1007/s11227-021-03743-2.

[19] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014, pp. 1532–1543, doi: 10.3115/v1/d14-1162.

[20] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017, doi: 10.1162/tacl_a_00051.

[21] R. Bakır and H. Bakır, "Swift detection of XSS attacks: enhancing XSS attack detection by leveraging hybrid semantic embeddings and AI techniques," *Arabian Journal for Science and Engineering*, vol. 50, no. 2, pp. 1191–1207, 2025, doi: 10.1007/s13369-024-09140-0.

[22] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," *Scientific Reports*, vol. 14, no. 1, 2024, doi: 10.1038/s41598-023-48845-4.

[23] R. L. Alaoui and E. H. Nfaoui, "Web attacks detection using stacked generalization ensemble for LSTMs and word embedding," *Procedia Computer Science*, vol. 215, pp. 687–696, 2022, doi: 10.1016/j.procs.2022.12.070.

[24] C. Lente, R. Hirata Jr., and D. M. Batista, "An improved tool for detection of XSS attacks by combining CNN with LSTM," *Anais Estendidos do Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais,* pp. 1–8, 2021, doi: 10.5753/sbseg_estendido.2021.17333.

[25] W. Zhang, Z. Lin, and X. Liu, "Short-term offshore wind power forecasting-a hybrid model based on discrete wavelet transform (DWT), seasonal autoregressive integrated moving average (SARIMA), and deep-learning-based long short-term memory (LSTM)," *Renewable Energy*, vol. 185, pp. 611–628, 2022, doi: 10.1016/j.renene.2021.12.100.

BIOGRAPHIES OF AUTHORS

Muhammad Alkhairi Mashuri holds a Bachelor of Applied Science from Politeknik Caltex Riau, Indonesia in 2018. He is currently pursuing master's degree at Bina Nusantara University. He is also working as an Assistant Vice President (AVP)-Full Stack Software Engineer at OCBC Sdn Bhd, Malaysia. His research interest is in software engineering and cyber security. He can be contacted at email: m.mashuri@binus.ac.id.

Nico Surantha received the B.Eng. and M.Eng. degrees from the Bandung Institute of Technology, Indonesia and the Ph.D. degree from the Kyushu Institute of Technology, Japan. He is a lecturer with the Department of Electrical, Electronics, and Communication Engineering, Tokyo City University, Japan. He is also an Associate Professor with the Master of Computer Science Department, BINUS Graduate Program, Bina Nusantara University, Indonesia. He was with the Autonomous University of Barcelona (Spain) and the University of Technology Sydney (Australia) in 2024 as a visiting professor and visiting researcher, respectively. His research interests include ubiquitous computing, intelligent systems, the internet of things, and digital health. He is currently a research expert committee member at the IEICE Smart Info Media System Society and the Development of Digital Human Resources Organization in Japan (DDHR) association. He is a senior member of IEEE and a member of IEICE Japan. He can be contacted at email: nico.surantha@binus.ac.id.