Journal of Computer Science and Cybernetics, V.42, N.x (2026), 1-
DOI: 10.15625/1813-9663/23407

ENHANCING WEB ATTACK DETECTION EFFICIENCY BASED ON NATURAL LANGUAGE PROCESSING TECHNIQUES

CHO DO XUAN[1,2], NGUYEN MANH SON[1,2], MAI THE TOAN[3]

[1]*Department of Information Security, Posts and Telecommunications Institute of Technology, 96A Tran Phu street, Ha Dong Ward, Ha Noi, Viet Nam*
[2]*Lab Blockchain, Department of Information Security, Posts and Telecommunications Institute of Technology, 96A Tran Phu street, Ha Dong Ward, Ha Noi, Viet Nam*
[3]*Faculty of Advanced Technology and Engineering, Vietnam Japan University, My Dinh 1 Urban Area, Cau Dien Ward, Ha Noi, Viet Nam*

Abstract. Web security continues to pose significant challenges for organizations and individuals worldwide. The rapid evolution of technology has made website protection increasingly complex against sophisticated cyber attacks. Delayed vulnerability remediation often results in severe consequences, including authentication bypass, data breaches, information theft, and complete system compromise. Despite extensive research and proposed attack prevention and mitigation solutions, existing approaches demonstrate limitations due to high false-positive rates. Recent advances in artificial intelligence have shown promising results in improving detection capabilities, especially for zero-day attacks. In this study, we propose the Lightweight Deep Web Learning (LDWL) model based on Natural Language Processing (NLP) for feature extraction and employ multilayer neural networks for classification. We evaluated the model on the HTTP Param dataset and achieved excellent results, with all metrics reaching 99.99% accuracy.

Keyword. Web security, attack detection, natural language processing, deep learning, neural networks, HTTP parameter analysis, web attack prevention.

1. INTRODUCTION

Web applications are now essential components in the digital world, providing information and online services. However, their popularity has also made them attractive targets for sophisticated cyber attacks [1]. Attackers actively exploit security weaknesses such as SQL injection (SQLi) [2], Cross-Site Scripting (XSS) [3], Command injection (CMDi), and other vulnerabilities, causing significant damage to organizations and individuals. Global reports [4, 5] show a steady increase in website attacks, especially those targeting web applications and Application Programming Interface (API). To protect against these threats, organizations have implemented various security measures, particularly Web Application Firewalls (WAFs), which act as security barriers between attackers and web applications [6, 7, 8].

*Corresponding author:
E-mail address: chodx@ptit.edu.vn (C.D. Xuan), sonnm@ptit.edu.vn (N.M. Son), maithetoan2005@gmail.com (M.T. Toan).

However, the Ponemon Institute reports that only 9% of WAF users believe their systems have never been breached (Vicente, 2019) [9]. To improve WAF effectiveness, researchers are turning to machine learning [10] and deep learning [11, 12] solutions. The use of behavior analysis techniques to detect web attacks has shown promising results [2, 3, 6, 7, 8, 13]. Machine learning models can automatically identify normal and suspicious activities in data, understand attack patterns, and reduce reliance on manual rule creation. Recent research has successfully used both manual analysis and Natural Language Processing (NLP) methods, such as Word2Vec [14], Doc2Vec [15], and n-gram [16, 17], to identify attack characteristics. The combination of large models like BERT [18] with deep learning [11, 12] has further improved detection accuracy. However, these methods still need improvement. Current NLP models like Word2Vec [14] and Doc2Vec [15] are limited by their fixed vocabulary, which restricts their ability to understand attack patterns. This suggests the need for more advanced NLP models. In machine learning-based attack detection, the most important factor affecting accuracy is how the system identifies attack characteristics [2, 3, 6, 7, 8, 13]. Current methods using simple approaches like n-gram [16, 17] and bag-of-words [19] often miss important information due to vocabulary limitations. This makes it harder to detect new types of attacks [20, 21]. To solve these problems, researchers are developing models that can learn from unlabeled data. Some recent studies have also used Large Language Models (LLMs) to analyze URLs directly [18]. While these studies [18, 22] show good results in identifying attack patterns, they require too much computing power and can't detect attacks in real-time, making them difficult to use in practice.

To address the issues presented above, we propose a new attack detection model based on the combination of MiniLM [23] and multilayer neural networks. Specifically, we use a pre-trained and fine-tuned MiniLM model to extract features from Uniform Resource Identifiers (URI). The model then processes these features through a Multilayer Perceptron (MLP) [24, 25] for classification. The use of MiniLM in this problem aims to solve the limitations in representing URI semantics, helping feature extraction to fully capture URI characteristics. It can recognize complex pattern relationships and represent diverse valuable information, even when patterns are not present in known attack data or pattern representations are not yet clear.

The paper is structured as follows. First, we review the related work and summarize existing studies that are closely connected to our research topic. Next, we introduce the proposed LDWL model in detail, including its main design principles and methodological framework. Finally, we present the experimental setup and evaluation results to demonstrate the effectiveness and performance of the proposed approach.

2. RELATED WORKS

2.1. Input data filtering-based approaches

Solutions in these approaches utilize rules and signatures to detect and prevent web attacks. Notable tools include Libinjection [26], XSStrike [27], Detectify [28], and Probely [29]. Libinjection relies on syntax analysis from attack patterns collected from millions of real requests to detect SQL injection and XSS. However, it requires manual configuration and system integration. XSStrike employs fuzzing techniques and tests multiple payload types to detect complex XSS attack forms, including polymorphic XSS, but requires users to

have in-depth knowledge for effective setup and usage. Detectify is an automated tool that checks for over 2000 vulnerabilities, including SQL injection and XSS, through a signature database. However, it may still encounter false predictions in complex applications. Probely is a signature-based security vulnerability detection tool that provides detailed remediation guidance, but its scanning capabilities remain limited on complex applications.

2.2. Anomaly-based approaches

Solutions in these approaches focus on detecting anomalous behaviors based on application behavior profiles. SQIRL [30] uses 'grey-box' techniques and dynamic application feedback, applying reinforcement learning to optimize SQLi detection by minimizing unnecessary requests. TPSQLi [31] focuses on high-risk elements in applications and prioritizes testing of vulnerable areas based on previous test results. SQLIML [32] analyzes SQL patterns in real-time to detect anomalous behaviors in SQL queries. PHPIDS [33] creates normal behavior profiles of PHP web applications and monitors requests to detect abnormal differences in traffic. Torrano-Gimenez Anomaly Detection [34] builds normal behavior profiles of HTTP requests and monitors traffic to detect anomalies in SQL queries and XSS. Sec2vec [35] is a model for detecting anomalies in HTTP traffic and malicious URLs by using natural language processing methods to convert HTTP requests into vectors, thereby detecting anomalous patterns.

2.3. Machine learning-based approaches

Guo, Yuan, and Wu [36] tested LogBERT on three different datasets, HDFS, BGL, and Thunderbird, achieving high detection rates of 82.32%, 90.83%, and 96.64%, respectively, for anomalous patterns. Additionally, Yunus et al. [37] proposed a new model to distinguish between normal and abnormal HTTP requests, using Natural Language Processing (NLP) and a BERT encoder [18] combined with deep learning techniques. This model was tested on the CSIC 2010 dataset [38], achieving an over 99% success rate with lower attack detection times compared to other models.

Betarte et al. [39] proposed a method using one-class classification and n-gram analysis to improve ModSecurity's attack detection performance. One-class classification is particularly useful when only normal data is available for training. This method builds a model based on normal patterns and treats any deviation from trained patterns as anomalous. The n-gram technique [16, 17] analyzes character or word sequences in data and compares them with known patterns. When both normal and attack data are available, n-gram analysis improves attack detection through sequence pattern analysis.

Udi Aharon et al. [40] proposed an API anomaly detection method called FT-ANN, designed to work effectively with limited training data (few-shot learning). This method combines FastText embedding and Approximate Nearest Neighbor (ANN) search to classify abnormal API requests. The model uses a Classification-by-Retrieval structure, enabling high-accuracy API attack detection, resource optimization, and easy adaptation to various API types like REST and GraphQL. Tests on CSIC 2010 [38] and ATRDF 2023 [41] datasets show that FT-ANN reduces false positives and improves accuracy, making it a promising tool for real-time API attack detection.

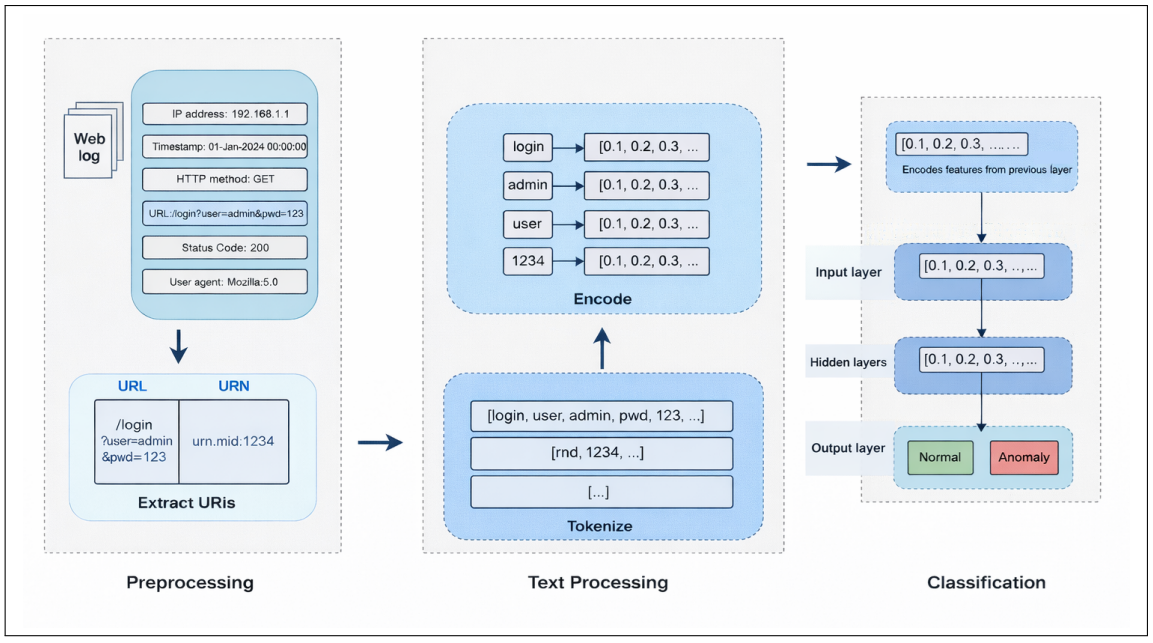

Figure 1: The architecture of the LDWL model

3. PROPOSED LDWL MODEL

3.1. Overview of the LDWL model

Based on the introduction presented above and Figure 1, the operating principle of the proposed model can be described as follows: First, web logs are input for analysis and extraction of URIs (Uniform Resource Identifier) patterns. These URI patterns undergo preprocessing to convert the data into a format compatible with the dataset. The data then passes through the Tokenization block to convert from text to numerical form. The Encode block performs encoding and synthesis to create feature vectors for each pattern. These feature vectors are transmitted through neural network layers to extract higher-level features. Finally, the deep learning algorithms in the Classification block perform prediction and classify requests into two classes: Normal or Anomaly.

3.2. Data pre-processing

In the data pre-processing phase, converting URLs to URIs is crucial for standardizing input data. URIs provide a more generalized structure than URLs, allowing a comprehensive representation of components such as scheme, authority, path, query, and fragment. This conversion helps standardize web address representations, avoiding differences caused by URL variants such as relative URLs, encoded URLs, or URLs containing special characters. Particularly in web attack detection, URIs enable more detailed analysis of address structure and semantics, thereby facilitating the identification of suspicious or malicious patterns in different address components. SQL injection attacks typically appear in the query portion of URIs, while path traversal attacks focus on the path component. Therefore, using URIs instead of plain URLs enhances the capability to detect and classify web attacks more effectively.

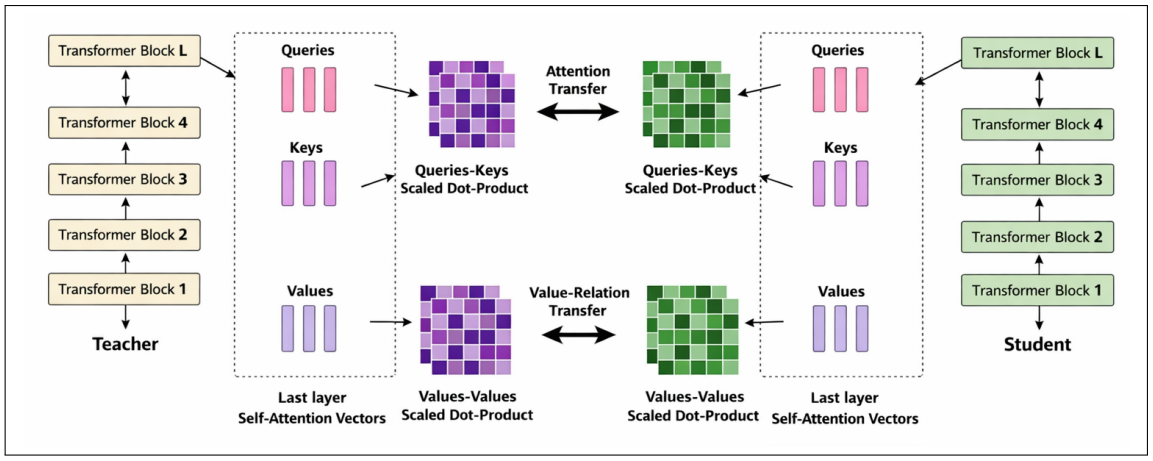

Figure 2: The architecture of the MiniLM model

3.3. Encode layer architecture model

3.3.1. Overview of the MiniLM model

a) Architecture of the MiniLM model

MiniLM [23] (Miniature language model) is a lightweight language model designed to inherit the power of large Transformer models like BERT [18] and RoBERTa [42], while requiring lower computational resources. Through the use of knowledge distillation techniques [43, 44, 45] and optimization of self-attention mechanisms [44, 46], MiniLM maintains high performance on Natural Language Processing (NLP) tasks without requiring the large number of parameters of the original models.

Below is a detailed description of MiniLM's architecture and its operating mechanism. Figure 2 illustrates the architecture of the MiniLM model.

From Figure 2, we can see that MiniLM is based on the Transformer architecture [18, 36, 42, 47], consisting of Encoder layers responsible for learning semantic representations from input sentences. Each Encoder layer in MiniLM includes two main components:

- Self-Attention Layer [44, 46]: This component helps identify relationships between words in a sentence, independent of their positions.
- Feed-Forward Layer: After calculating attention between words, the results are passed through a simple neural network [48] (feed-forward network) to adjust and represent semantic features.

The Transformer model operates sequentially, with stacked Encoder layers enabling the model to learn different levels of semantics from natural language data. The self-attention processing can be represented by the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{1}$$

where Q (Query), K (Key), and V (Value) are matrices constructed from the input sequence, d_k is the dimension of the Key vector space, and softmax is the function that normalizes the attention scores between the current word and the other words in the sequence.

b) Self-attention mechanism optimization

MiniLM [23] achieves optimization by reducing the number of multi-head attention [45, 49] in the self-attention mechanism. In a traditional Transformer [44], each head learns a different relationship between words in a sentence. MiniLM retains only the most important heads, significantly reducing computational requirements while ensuring that the most critical attention relationships are maintained. The calculation formula in multi-head attention after optimization can be rewritten as formula (2)

$$\text{Multi-HeadAttention}(Q, K, V) = \sum_{i=1}^{H_{\text{pruned}}} \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i, \tag{2}$$

where, H_{pruned} is the number of heads retained after pruning (removing less important heads), Q_i, K_i, and V_i are the corresponding vector at the i^{th} head.

Reducing the number of heads not only speeds up computation but also retains essential information for language tasks. This is particularly beneficial for deploying MiniLM on resource-constrained systems and enables real-time usage.

c) Knowledge distillation

MiniLM [23] is built using knowledge distillation techniques [50], where a larger model, typically BERT [18] or RoBERTa [42], serves as the teacher model. MiniLM, acting as the student model, learns to recreate hidden representations and classification results from the larger model, but with a simpler and more compact structure. The knowledge distillation process is described through formula (3)

$$L_{distill} = \alpha \cdot L_{hard}\left(y, y_{true}\right) + (1 - \alpha) \cdot L_{soft}\left(y, y_{teacher}\right). \tag{3}$$

By learning representations and relationships from the teacher model, MiniLM can maintain learning quality without using the full number of parameters as in the larger model. This helps MiniLM minimize model size and increase computational speed.

d) Representation compression via pruning techniques

MiniLM also optimizes through representation compression [51, 52] by learning only the summary representations from the most important attention layers of the teacher model. Pruning techniques [53, 54] help eliminate unimportant attention relationships, thereby reducing the number of parameters required for the model. The self-attention compression function can be expressed as follows

$$L_{pruned\ attention} = \sum_{i=1}^{H_{pruned}} ||Attention_{MiniLM} - Attention_{teacher}||^2. \tag{4}$$

This process ensures that MiniLM learns attention relationships from the teacher model without retaining all unnecessary information.

3.3.2. Using MiniLM for web attack feature extraction

In this research, we apply MiniLM to extract features from URIs through a structured processing pipeline. First, URIs are preprocessed by normalizing to lowercase and splitting

into main components, including path and parameters. Then, MiniLM's tokenizer is used to segment URIs into meaningful tokens, while adding special tokens [CLS] and [SEP] at the beginning and end of the sequence. This token sequence is normalized in length through padding or truncating to ensure uniform size before being passed through MiniLM's layers to generate vector representations. The vector at the [CLS] token position is used as a feature vector representing the entire URI with 384 dimensions, containing the URI's semantic information. This feature vector is then normalized to ensure stability during learning while preserving important semantic relationships and structures in the URI. This method offers several advantages, such as the ability to capture context and complex relationships between URI components, not being limited by a fixed dictionary, good generalization capability with new attack patterns, and particularly fast processing speed thanks to MiniLM's optimized architecture. The obtained feature vector will be used as input for the classification model, helping to increase efficiency in detecting web attacks, especially with previously unknown attack patterns.

3.4. Classification

To ensure good classification capability of the model with transformed and processed data, in this research, we use MLP [24, 25] to help the model effectively classify complex unstructured data.

4. EXPERIMENT AND EVALUATION

4.1. Experiment datasets and data collection

The HTTP Param dataset [55] includes 31,067 URI payloads from web requests, containing payload length and labels as shown in Table 1. There are two payload labels: Normal and Anomal (abnormal). The Anomal payloads are divided into four types: SQLi, XSS, CMDi, and Path-traversal.

Table 1: Number of payloads in the HTTP param dataset

Type of payloads	Number of payloads	Percentage (%)
Normal	19304	62.2
SQL Injection	10852	35.0
XSS	532	1.7
CDMi	89	0.3
Path traversal	290	0.9
Total	31067	100

In our experiments, we utilized a single file containing the entire dataset to train and validate the proposed web attack detection model. The data splitting method was employed with 80% of the dataset allocated for training, 10% for validation, and the remaining 10% for testing.

4.2. Evaluation metrics

In this research, we employ four main metrics. These are common metrics widely used in research studies within this field.

Table 2: Comparison of models on the HTTP Param dataset

Model	Parameter			Performance			
	Class weight	Max length	Batch size	Acc	Pre	Rec	F1
Our model	None	64	64	99.75%	99.97%	99.30%	99.63%
	Balanced	128	128	**99.99%**	**99.95%**	**99.85%**	**99.90%**
Dau et al. [56]	None	X	X	99.68%	99.70%	99.60%	99.65%
	Balanced	X	X	99.60%	99.50%	99.75%	99.62%
Liang et al. [57]	None	64	X	98.42%	98.50%	98.30%	98.40%
	Balanced	128	X	98.20%	99.00%	97.30%	98.14%
Pan et al. [58]	None	64	64	91.40%	91.50%	91.20%	91.35%
	Balanced	128	128	90.90%	90.20%	92.50%	91.34%

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{7}$$

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{8}$$

where TP-True Positive: the number of abnormal samples correctly classified, FN-False negative: the number of abnormal samples misclassified as normal, TN-True negative: the number of normal samples correctly classified, and FP-False Positive: the number of normal samples misclassified as abnormal.

4.3. Evaluation scenarios

To evaluate the validity and effectiveness of our model, we conduct assessments based on two research questions (RQs):

- RQ1: Does the proposed LDWL model perform better than current state-of-the-art models in web attack detection?
- RQ2: How effective is the LDWL model when applied in real-world environments?

4.4. Experimental results

4.4.1. Experimental results for RQ1

The experimental results from Table 2 demonstrate the superior performance of the LDWL model in detecting web attacks on the HTTP Param dataset [55]. The model achieved 99.99% accuracy across all evaluation metrics, including Accuracy, Precision, Recall, and F1-score. This effectiveness is attributed to the model's optimized processing pipeline architecture. From web logs, URI samples are extracted and processed through pre-processing steps

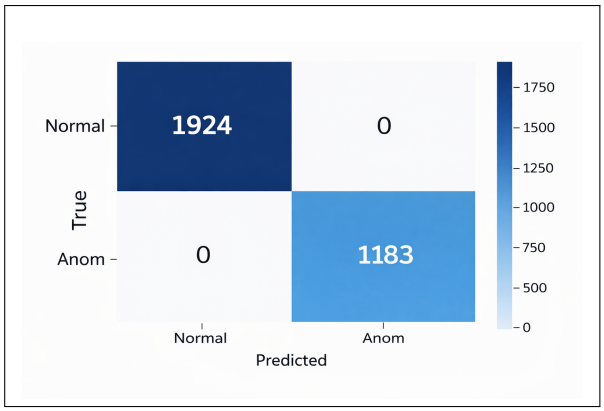

Figure 3: Confusion matrix of the LDWL model

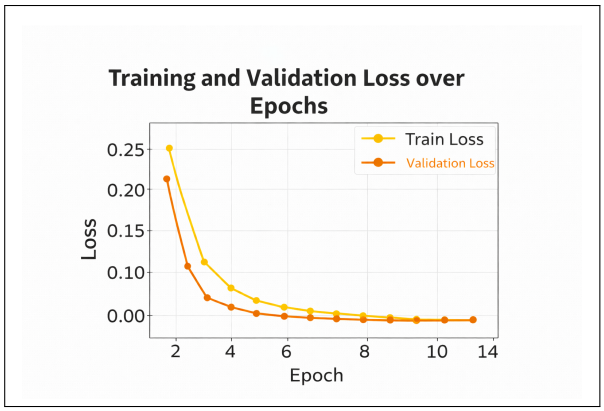

Figure 4: Training and validation loss curves

to convert data into a standardized format. The tokenization process helps transform data from text to numerical form, followed by encoding and feature vector compilation. These vectors are passed through representation learning layers to learn higher-level features before being classified by deep learning algorithms. Compared to previous studies by Dau et al. [56] using a decision tree (99.68%), Liang et al. [57] with RNN (98.42%), and Pan et al. [58] using Word2vec-SVM (91.40%), LDWL's sequential processing pipeline approach demonstrates more effective attack feature capture capabilities. The Confusion Matrix in Figure 3 illustrates the model's accurate classification performance on the experimental dataset.

According to Table 2, the prediction performance on samples further demonstrates the validity of LDWL.

Figure 4 shows a decreasing trend in both Train Loss and Validation Loss over 15 epochs, reflecting the model's effective training process. In the initial phase (epochs 1-5), both losses decrease rapidly, then gradually stabilize from epoch 6 onwards, indicating good convergence of the model. The similarity between the two loss curves suggests that the model does not suffer from overfitting [20, 21], ensuring high generalization on the validation set.

Table 3: Model processing time per request

Method	Time (ms)
Web attack detecting via autoencoder [58]	5.1
Text analysis based SQLi attack detection [59]	0.89
LogBERT [36]	0.4
3-gram [56]	0.068
Our model	0.12

4.4.2. Model performance in real-world applications

In this section, we present experimental results on real web logs collected from our web servers. Typically, popular web servers such as Apache HTTP Server, Microsoft Internet Information Services (IIS), and Nginx record logs as text files containing information about requests sent to the server. The table above shows the average processing time per request of our proposed model compared to other methods.

Based on the processing time comparison in Table 3, although slower than n-gram [58] with 0.06s, our model, with a processing time of 0.12ms, is still sufficiently fast for many practical applications and offers higher accuracy. Our model demonstrates significant superiority in detecting attack patterns compared to existing methods such as Autoencoder [58], Text Analysis [59], and LogBert [36, 47]. In the field of cybersecurity, instantaneous response capability is crucial for preventing attacks as they occur. Our LDWL approach not only improves performance but also minimizes latency to the lowest level, enabling the system to quickly detect and respond to web threats.

5. DISCUSSION

The experimental results presented in Section 4 have demonstrated the effectiveness of the proposed model. In this section, we will discuss the functionality of each component that helps the model achieve these results.

5.1. Role of the encoder model

The use of NLP models for embedding significantly enhances the semantic learning capability of URIs compared to Word2vec, GloVe, or models like LSTM or GRU. The implementation of MiniLM [23] for generating embeddings, rather than using other NLP models such as RoBERTa [42] or BERT [18], aims to improve the speed of URI embedding for feature extraction, enabling optimal performance when applying the model in real-time systems.

Table 4: Parameters and Speed Comparison with BERT

Model	Parameters	Speed (compare to BERT)
DistilBERT [60]	66M	60% faster
MiniLM [23]	12M	Faster than DistilBERT
ALBERT [61]	Very few	Faster than BERT
Electra [62]	14M	Faster than BERT
RoBERTa [42]	110M	Comparable to BERT
FastText [63]	Very few	Fastest

As shown in Table 4, MiniLM and FastText provide the highest processing speeds for extracting URI features. However, FastText's limited number of parameters may significantly constrain its feature extraction capabilities, especially as attack techniques become increasingly sophisticated and harder to detect. The use of MiniLM not only ensures effective feature extraction but also maintains robust performance in NLP tasks through knowledge transfer learning [50] and optimized self-attention mechanisms [44, 46], while significantly reducing the number of parameters and computational costs.

5.2. Limitation

Although the model has demonstrated good effectiveness on the HTTP Param dataset [55], there remain numerous opportunities for future development and research expansion. Testing on a single dataset is only the initial step in evaluating the model's effectiveness. In the context of increasingly diverse and unpredictable cyber attacks [1, 4, 5], testing across various data types would provide a more comprehensive assessment of the model's adaptability, particularly with complex web attacks such as SQL Injection, Cross-Site Scripting, or Command Injection.

Furthermore, the model's performance needs additional optimization, especially regarding processing time and system resource utilization. While MiniLM significantly reduces computational costs compared to other NLP models, the use of deep learning still requires substantial hardware resources when deployed in real-world environments.

Table 5 below shows some of the evaluation results in terms of training and testing time of the proposed model with the simplest natural language processing model.

Table 5: Details of processing time assessment results

Method	Time requirements (s)	Time per Sample (ms)	MLP Train Time (s)
Our	3.65	0.117	4.6
2-gram [56]	1.70	0.055	4.10
3-gram [56]	2.10	0.068	4.80
4-gram [56]	2.60	0.084	5.50

From the results in the table, it takes longer to process the proposed model than the approaches using n-grams. This leads to the need to find ways to improve and refine the proposed model in the next study, which not only brings better efficiency but also reduces the calculation time.

Additionally, the model may face challenges with encoded attacks or those specifically designed to evade detection. The lack of an automatic mechanism for updating and learning from new attack patterns is also a limitation that needs to be addressed.

6. CONCLUSIONS

This research has made significant contributions to the field of web attack detection through the proposed LDWL model. Technically, the model has demonstrated superior performance with 99.99% accuracy across all evaluation metrics when tested on the HTTP Param dataset. Notably, the integration of MiniLM into the proposed architecture not only improves feature extraction capabilities but also ensures fast processing speeds, meeting the real-time requirements of security systems.

In the field of Information Security, the research has made two important contributions. The effective combination of natural language processing and machine learning techniques has created a novel approach to detecting web attacks, outperforming traditional methods such as Decision Tree, RNN, or Word2vec. Moreover, with a processing time of only 0.12ms, the model has proven its high practical applicability, meeting the requirement for an instantaneous response in detecting web attack patterns.

Based on these achievements, the research proposes three main directions for future development. The first approach is to expand the testing scope across diverse datasets, focusing on highly complex attacks such as SQL Injection, Cross-Site Scripting, and Command Injection. In parallel, the research will concentrate on optimizing processing performance and system resource management to enhance practicality when deployed in production environments. Notably, the key development direction is to build a continuous learning mechanism, allowing the system to automatically update and adapt to new attack patterns. These improvements will contribute significantly to developing an intelligent security system with high adaptability to increasingly sophisticated cyber threats.

REFERENCES

[1] OWASP Foundation, "Owasp top 10: The ten most critical web application security risks," https://owasp.org/Top10/, 2021, online.

[2] U. Farooq, "Ensemble machine learning approaches for detection of sql injection attack," *Tehnički glasnik*, 2021, retrieved from Semantic Scholar CorpusID:233804865.

[3] J. Kaur, U. Garg, and G. Bathla, "Detection of cross-site scripting (xss) attacks using machine learning techniques: A review," *Artificial Intelligence Review*, vol. 56, pp. 12 725–12 769, 2023.

[4] M. Tremante, D. Belson, and S. Zejnilovic, "Application security report: Q2 2023," https://blog.cloudflare.com/application-security-report-q2-2023/, Aug. 2023, cloudflare.

[5] Verizon, "2024 data breach investigations report," https://www.verizon.com/business/resources/reports/dbir/, 2024, online.

[6] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of SQL attacks," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2021, pp. 123–140.

[7] C. I. Pinzón, J. F. De Paz, J. Bajo, Á. Herrero, and E. Corchado, "AIIDA-SQL: An adaptive intelligent intrusion detector agent for detecting SQL injection attacks," in *International Symposium on Hybrid Artificial Intelligence Systems (HAIS)*, 2020, pp. 453–462.

[8] M. Amouei, M. Rezvani, and M. Fateh, "Rat: Reinforcement-learning-driven and adaptive testing for vulnerability discovery in web application firewalls," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3371–3386, 2022.

[9] G. Vicente, "The top 5 reasons why waf users are dissatisfied," https://hdivsecurity.com/bornsecure/the-top-5-reasons-why-waf-users-are-dissatisfied/, 2019, retrieved March 22, 2020.

[10] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[11] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021.

[12] Y. Li, K. Xu, Q. Yan, Y. Li, and R. H. Deng, "Detecting web attacks with end-to-end deep learning," *Journal of Internet Services and Applications*, vol. 10, no. 1, pp. 1–22, 2019.

[13] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2020.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations (ICLR)*, 2013, pp. 1–12.

[15] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning (ICML)*, vol. 32, no. 2, 2014, pp. 1188–1196.

[16] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2012, pp. 90–94.

[17] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186.

[19] I. Sheikh, I. Illina, D. Fohr, and G. Linarès, "Learning word importance with the neural bag-of-words model," in *1st Workshop on Representation Learning for NLP*. Berlin, Germany: Association for Computational Linguistics, 2016, pp. 222–229.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[21] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022022, 2019.

[22] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *IEEE Access*, vol. 8, pp. 193 907–193 934, 2020.

[23] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," in *Neural Information Processing Systems (NeurIPS)*, 2021, pp. 5776–5788.

[24] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Computing Surveys*, vol. 55, no. 12, p. Article 259, 2023.

[25] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-mixer: An all-MLP architecture for vision," in *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021, pp. 24 261–24 272.

[26] fzpi and GitHub Contributors, "libinjection," https://github.com/libinjection/libinjection, accessed: Dec. 30, 2025.

[27] s0md3v and GitHub Contributors, "Xsstrike," https://github.com/s0md3v/XSStrike, accessed: Nov. 15, 2025.

[28] Detectify, "Detectify github repository," https://github.com/Detectify, accessed: Nov. 16, 2025.

[29] Probely, "Probely github repository," https://github.com/probely, accessed: Nov. 16, 2025.

[30] S. Al Wahaibi, M. Foley, and S. Maffeis, "SQIRL: Grey-box detection of SQL injection vulnerabilities using reinforcement learning," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023, pp. 6097–6114.

[31] G.-Y. Yang, F. Wang, Y.-Z. Gu, Y.-W. Teng, K.-H. Yeh, P.-H. Ho, and W.-L. Wen, "TPSQLi: Test prioritization for SQL injection vulnerability detection in web applications," *Applied Sciences*, vol. 14, p. 8365, 2024.

[32] D. Mehta, H. Suhagiya, H. Gandhi, M. Jha, P. Kanani, and A. Kore, "SQLIML: A comprehensive analysis for SQL injection detection using multiple supervised and unsupervised learning schemes," *SN Computer Science*, vol. 4, no. 281, 2023.

[33] PHPIDS Contributors, "PHPIDS: PHP intrusion detection system," https://github.com/PHPIDS/PHPIDS, accessed: Nov. 16, 2025.

[34] C. Torrano-Giménez, A. Pérez-Villegas, and G. Á. Marañón, "An anomaly-based approach for intrusion detection in web traffic," 2010, corpus ID: 18778194.

[35] M. Gniewkowski, H. Maciejewski, T. Surmacz, and W. Walentynowicz, "Sec2vec: Anomaly detection in http traffic and malicious urls," in *38th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2023, pp. 1154–1162.

[36] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via bert," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[37] Y. E. Seyyar, A. G. Yavuz, and H. M. Ünver, "An attack detection framework based on BERT and deep learning," *IEEE Access*, vol. 10, pp. 68 633–68 644, 2022.

[38] "Http-csic-2010 dataset," https://github.com/Kiinitix/HTTP-CSIC-2010, dataset, Accessed: Dec. 30, 2025.

[39] N. Montes, G. Betarte, R. Martínez, and A. Pardo, "Web application attacks detection using deep learning," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (CIARP)*, ser. Lecture Notes in Computer Science. Springer, 2021, vol. 12702.

[40] U. Aharon, R. Dubin, A. Dvir, and C. Hajaj, "A classification-by-retrieval framework for few-shot anomaly detection to detect API injection attacks," *Computers and Security*, 2024.

[41] S. Lavian, R. Dubin, and A. Dvir, "API traffic research dataset framework (ATRDF)," https://github.com/ArielCyber/Cisco_Ariel_Uni_API_security_challenge, 2023, dataset.

[42] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized bert pretraining approach," 2019, arXiv preprint.

[43] M. Ji, B. Heo, and S. Park, "Show, attend and distill: Knowledge distillation via attention-based feature matching," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14 059–14 068.

[44] K. Wang, F. Yang, and J. van de Weijer, "Attention distillation: Self-supervised vision transformer students need more guidance," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 12 775–12 785.

[45] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," in *International Conference on Learning Representations (ICLR)*, 2020.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[47] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1285–1298.

[48] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 5484–5495.

[49] I. Rish, "An empirical study of the naive bayes classifier," in *Workshop on Empirical Methods in Artificial Intelligence (IJCAI)*, 2001, pp. 41–46.

[50] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[51] Z. Pei, A. Zhang, S. Wang, X. Ji, and Q. Huang, "Data-free neural representation compression with riemannian neural dynamics," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, vol. 235, 2024, pp. 40 129–40 144.

[52] A. Koratana, D. Kang, P. Bailis, and M. Zaharia, "LIT: Block-wise intermediate representation training for model compression," in *Proceedings of the 2019 International Conference on Machine Learning (ICML)*, 2019, pp. 3819–3828.

[53] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *IEEE Access*, vol. 8, pp. 187 577–187 593, 2020.

[54] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[55] "Http param dataset," https://github.com/Morzeux/HttpParamsDataset, dataset, Retrieved June 20, 2020.

[56] X. D. Hoang, "Detecting common web attacks based on machine learning using web log," in *Advances in Engineering Research and Application (ICERA)*, ser. Lecture Notes in Networks and Systems. Springer, 2021, vol. 178.

[57] J. Liang, W. Zhao, and W. Ye, "Anomaly-based web attack detection: A deep learning approach," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing (ICNCC)*. ACM, 2017, pp. 80–85.

[58] L. Yu, S. Luo, and L. Pan, "Detecting SQL injection attacks based on text analysis," in *Proceedings of the 3rd International Conference on Computer Engineering, Information Science and Application Technology ICCIA*. Atlantis Press, 2019, pp. 95–101.

[59] H. Mac, D. Truong, L. Nguyen, H. Nguyen, H. A. Tran, and D. Tran, "Detecting attacks on web applications using autoencoder," in *International Symposium on Information and Communication Technology*. ACM, 2018, pp. 416–421.

[60] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of bert: Smaller, faster, cheaper and lighter," 2019, arXiv:1910.01108.

[61] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2019, arXiv preprint.

[62] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[63] 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.

Received on September 07, 2025
Accepted on December 13, 2025