WADBERT: Dual-channel Web Attack Detection Based on BERT Models

Kangqiang Luo*, Yi Xie†, Shiqian Zhao‡, Jing Pan*✉
*Guangzhou Institute of Technology, Xidian University, China
†Tsinghua University, China
‡Nanyang Technological University, Singapore
luokangqiang2003@163.com, yi-xie@tsinghua.edu.cn, shiqian.zhao@ntu.edu.sg, jinglap@aliyun.com

***Abstract*—Web attack detection is the first line of defense for securing web applications, designed to preemptively identify malicious activities. Deep learning-based approaches are increasingly popular for their advantages: automatically learning complex patterns and extracting semantic features from HTTP requests to achieve superior detection performance. However, existing methods are less effective in embedding irregular HTTP requests, even failing to model unordered parameters and achieve attack traceability. In this paper, we propose an effective web attack detection model, named `WADBERT`. It achieves high detection accuracy while enabling the precise identification of malicious parameters. Towards this goal, we first employ Hybrid Granularity Embedding (HGE) to generate fine-grained embeddings for URL and payload parameters. Then, URLBERT and SecBERT are respectively utilized to extract their semantic features. Further, parameter-level features (extracted by SecBERT) are fused through a multi-head attention mechanism, resulting in a comprehensive payload feature. Finally, by feeding the concatenated URL and payload features into a linear classifier, a final detection result is obtained. The experimental results on CSIC2010 and SR-BH2020 datasets validate the efficacy of `WADBERT`, which respectively achieves F1-scores of 99.63% and 99.50%, and significantly outperforms state-of-the-art methods.**

1. Introduction

web attack comprises of a set of malicious behaviors targeting web applications (e.g., Amazon, Facebook and Gmail), wherein attackers exploit vulnerabilities to hijack accounts, steal data, spread malware, or disrupt services [1], [2]. Unlike traditional attacks that compromise personal devices through implanting viruses or malicious programs, web attacks invade publicly accessible applications [3] to access users' sensitive data without being authorized, compromising confidentiality, integrity and availability of systems [4].

Web attack detection is a reliable solution to prevent most web attacks. Its basic idea is to identify malicious patterns by analyzing HTTP requests that consist of three components—request line, request headers and request body (illustrated in Figure 1). Specifically, it first extracts features from these components, then applies rule-based, machine learning, or deep learning methods to identify malicious requests. Out of three, the rule-based methods utilize whitelists and blacklists to classify requests as benign or malicious through patterns matching [5], [6], [7]. However, they will fail to work when attacks fall outside predefined rules. In other words, for any attack, which employs encodings or obfuscations to conceal malicious payloads, the detections would be bypassed. Machine learning provides a flexible solution. It is able to learn complex attack patterns beyond predefined rules by training models on handcrafted features [8], [9], [10]. Unfortunately, handcrafted features cannot effectively represent the semantics and contextual relationships of HTTP requests. Deep learning is believed to be a more effective solution since it is capable of automatically learning these relationships through neural language models [11], [12], [13]. Several studies [14], [15], [16], [17] based on multi-layer neural networks have seen better performances for web attack detections. Indeed deep learning provides greater flexibility and accuracy. However, these methods still face several challenges:

(1) Inadaptability of embedding methods: Existing embedding methods (e.g., BPE [18] and WordPiece [19]) are primarily designed for natural language and unapplicable to URLs and payloads. The reason is that URLs and payloads often contain non-standard tokens (e.g., getUserName), symbol-dense strings (e.g., %27 or 1%2B=1%20) and strings with low tolerance for variation, thus these embedding methods fail to learn robust representations of HTTP requests.

(2) Limitations of ordered parameters: According to the design of HTTP, payload parameters in a HTTP request should be unordered. More precisely, HTTP requests containing a same group of parameters with different orders are functionally equivalent, e.g., "id=123&name=John" *vs* "name=John&id=123". However, previous studies [20], [21], [22] treat parameters as ordered sequences, failing to recognize this equivalence. Therefore, modeling should consider the combinatorial relationships among payload parameters rather than sequential relationships.

(3) Lack of attack traceability: Existing detection methods [16], [17] achieve relatively high accuracy of attack detections but lack the ability to trace attacks. They focus on detecting whether a request is malicious, but do not identify specific malicious parameters, thereby failing to pinpoint the

sources of attacks. This limitation reduces their practicality for security response.

To address these challenges, this paper proposes a novel web attack detection model, called WADBERT, designed to identify malicious requests in complex HTTP traffic. It consists of three primary phases. In the first phase, WADBERT generates embeddings for URL and payload parameters using Hybrid Granularity Embedding (HGE), which integrates the subword-level semantic information with fine-grained character features. In the second phase, WADBERT utilizes URLBERT [23] and SecBERT [24] to extract semantic features from these embeddings. In the third phase, WADBERT employs a multi-head attention mechanism to fuse parameter-level features, generating a comprehensive payload feature. The concatenated URL and payload features are then fed into a linear classifier to produce the final detection result. With these techniques, WADBERT can effectively generate embeddings for irregular HTTP requests, model unordered payload parameters, and identify malicious parameters, thereby improving the detection performance.

We evaluate WADBERT on two publicly available datasets: CSIC2010 [25] and SR-BH2020 [26]. The experimental results show that WADBERT achieves accuracy of 99.70% and 99.32%, respectively, outperforming existing methods in terms of accuracy, recall, precision, and F1-score. The ablation studies show that the performance would degrade if we remove key components (e.g., HGE and the multi-head attention mechanism), which demonstrates their importance. Moreover, our attention weight visualization shows that WADBERT effectively identifies malicious parameters. This improves interpretability of its predictions. All these findings demonstrate that WADBERT is highly useful in detecting web attacks with high accuracy and strong interpretability.

In summary, this paper makes the following contributions.

- We propose WADBERT, an effective model for web attack detection. It is able to accurately identify malicious HTTP requests and effectively locate their corresponding attack parameters.
- To embed the symbol-dense HTTP requests effectively, we introduce HGE as the embedding layers of WADBERT. This significantly improves detection performance.
- To model the unordered relationships of parameters, WADBERT employs a multi-head attention mechanism. Then, it takes an attention weight analysis to identify malicious parameters. This enhances robustness to parameter permutations and achieves attack traceability.
- We compare WADBERT with existing deep learning methods. The results show its effectiveness in web attack detection, improving F1-score by 1.23% on SR-BH2020 and 0.64% on CSIC2010 over baselines.

The remaining sections of this paper are organized as follows. §2 discusses the related work. WADBERT is presented in §3. In §4, we provide the experimental results and discussions. §5 concludes this paper.

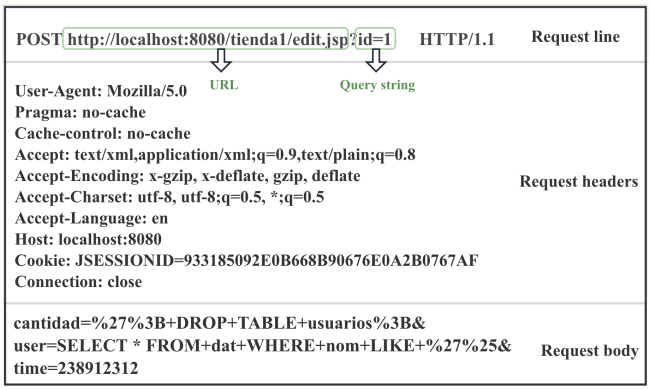


Figure 1: Components of an HTTP request. The URL is extracted from the request line, excluding the query string. The request body, query string and request header form a set of key-value pairs, known as the payload parameters.

2. Related Work

This section surveys the existing studies on web attack detection. We categorize them into machine learning, traditional deep learning, and Transformer-based methods. For each category, we discuss their main processes and respective limitations.

Machine learning methods mainly rely on manually designed statistical and lexical features for HTTP requests. These features are then fed into classifiers for training [8], [9], [10]. Typical models include Support Vector Machines, Naive Bayes, k-Nearest Neighbors, Decision Trees and ensemble models (e.g., Random Forest, Gradient Boosting and Xgboost). These models are lightweight so that training and inference are efficient, and the handcrafted features also enhance model interpretability.

However, their detection performance is dependent on the quality of the manually engineered features. Furthermore, HTTP requests are complex text sequences, so these methods often fail to capture their contextual and semantic relationships. Therefore, deep learning methods are gaining popularity in web attack detection due to their powerful ability to learn text representations.

Traditional deep learning methods leverage multi-layer neural networks to automatically learn semantic representations from raw HTTP requests. Early works targeting specific attack types (e.g., SQL injection and XSS) employ neural networks to learn textual patterns and contextual relationships from URLs and payloads. These models include feed-forward neural networks (FNN) [27], [28], convolutional neural networks (CNN) [29] and long short-term memory (LSTM) [30]. These single-attack detection models achieve high accuracy for particular attack patterns but fail to generalize across diverse web attack types.

Later works extend the dataset to include multiple attack types and propose models based on FNNs and autoencoders [31], [32]. These models rely on statistical and lexical features extracted from HTTP requests rather than embedding representations. Manually engineered features limit the

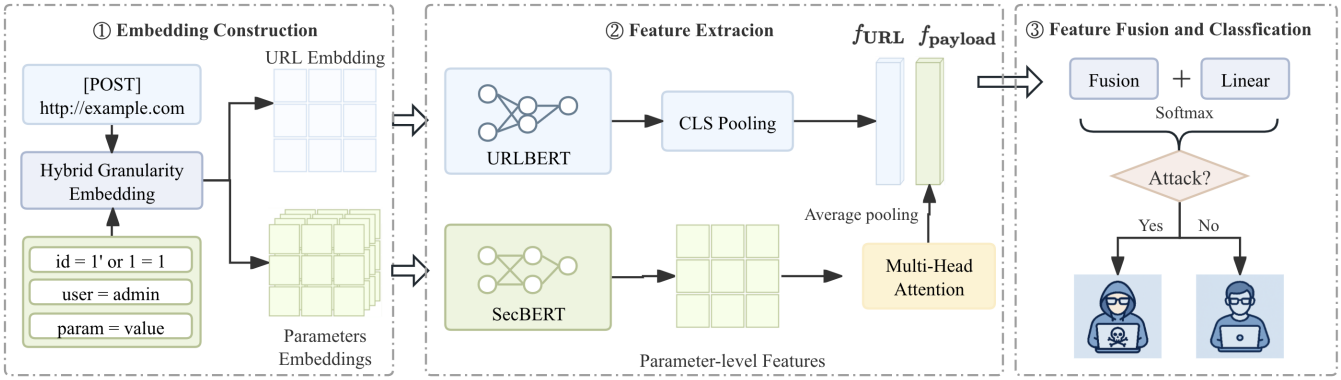


Figure 2: Framework of the proposed model.

ability of model to capture contextual and semantic relationships in request data. In contrast, embedding-based methods preserve token dependencies and latent semantics. Building on this, subsequent works propose CNN-based models [33], [34], [35], [36], which employ convolutional filters over char or word embeddings to extract n-gram features within HTTP requests. They effectively identify malicious requests by capturing local patterns in the data. Meanwhile, several works propose Recurrent Neural Network (RNN)-based [37] and LSTM-based models [38] that capture the contextual dependencies by processing HTTP requests token by token. This sequential modeling enables them to extract the semantic information of requests, improving accuracy in detecting web attacks.

However, CNNs excel at capturing local patterns but struggle to model long-range dependencies. Conversely, RNNs effectively handle sequential dependencies yet often overlook fine-grained local features. Consequently, hybrid models combining CNN and RNN (e.g., EDL [20] and CNN-BiLSTM [22]) are explored to enhance robustness. This complementary strengths helps mitigate the limitations of individual architectures, enabling more effective identification of sophisticated web attacks. Nonetheless, the limitations of RNNs in capturing long-range dependencies and their sequential computation bottlenecks derive the adoption of Transformer-based models [39].

Transformer-based methods employ the multi-head attention mechanism to capture global relationships among all tokens in a sequence. This mechanism generate a contextualized representation for each token to by aggregating semantic information from other tokens. Subsequently, these contextualized token representations are then combined to form a global representation of the sequence. Unlike RNNs and LSTMs, which process sequences step by step, Transformers process the entire sequence simultaneously. This allows them to capture dependencies between distant tokens, effectively modeling both local and global relationships. Experimental results from Transformer-based methods [14], [15], [16], [17] demonstrate their superior performance compared to traditional approaches.

Nonetheless, these approaches rely on embedding methods that are designed for natural language data. This makes them less effective in handling non-standard, symbol-dense strings commonly found in HTTP requests. Secondly, these approaches take the concatenated URL and payload sequence as input, which causes them to ignore the unordered nature of payload parameters. Moreover, they cannot effectively identify specific malicious parameters.

Therefore, we propose `WADBERT`, a dual-channel BERT-based model for web attack detection. It generates embeddings for the URL and payload parameters using HGE, and employs a multi-head attention mechanism to capture unordered relationships among payload parameters. Morever, `WADBERT` can identify specific malicious parameters through attention weight analysis. Overall, This approach provides both precision and interpretability.

3. Methodology

In this section, we provide the design details of the proposed `WADBERT`. Figure 2 illustrates the architecture of `WADBERT`, which consists of three key stages:

- *Embedding Construction (§3.1).* This stage uses HGE to generate fine-grained embeddings for the URL and each payload parameter.
- *Feature Extraction (§3.2 and §3.3).* At this stage, the URLBERT and SecBERT with a multi-head attention mechanism are respectively utilized to extract the URL and payload features.
- *Feature Fusion and Classification (§3.4).* This stage concatenates the URL and payload features, and feeds them into the classifier to produce detection results.

3.1. Hybrid Granularity Embedding (HGE).

HGE is capable of enhancing the representational capacity of traditional WordPiece embedding by integrating character-level features. Its process consists of three stages: tokenization and char embedding, character-level representation construction and hybrid embedding fusion, as depicted in Figure 3.

Step I: Tokenization and Char Embedding. For a given

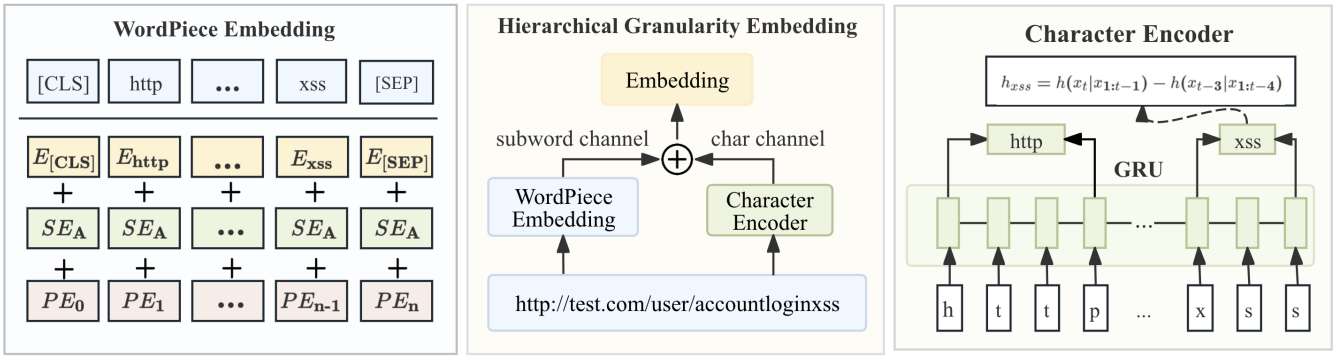


Figure 3: Framework of the HGE.

input text T, we first apply the WordPiece algorithm to tokenize it into a token sequence $S = [s_1, s_2, \ldots, s_m]$. Then, we map the corresponding character sequence $C = [c_1, c_2, \ldots, c_L]$ into char embeddings $X = [x_1, x_2, \ldots, x_L]$ where $x_j = CE[c_j]$ for each $j = 1, ..., L$. Here, $CE \in \mathbb{R}^{v \times d}$ is a char embedding matrix with character vocabulary size v and embedding dimension d. Lastly, HGE employs a bidirectional GRU network [40] to process these embeddings, producing forward and backward hidden states at each position, as follows:

$$\overrightarrow{h}_j = \text{GRU}_{\text{fwd}}(x_j, \overrightarrow{h}_{j-1}), \quad (1)$$

$$\overleftarrow{h}_j = \text{GRU}_{\text{bwd}}(x_j, \overleftarrow{h}_{j+1}), \quad (2)$$

where $\overrightarrow{h}_j$ and $\overleftarrow{h}_j$ are the forward and backward hidden states of the j-th position, respectively.

Step II: Character-level Representation Construction. For each token, HGE first computes forward and backward differential representations as in Equations (3) and (4). Then, by concatenating these differential representations, we obtain the character-level representations of all tokens h_{char}. Therefore, we effectively represent token variants by preserving fine-grained character information.

$$h_{\text{fw}} = \begin{cases} \overrightarrow{h}_e - \overrightarrow{h}_{s-1}, & s > 0, \\ \overrightarrow{h}_e, & s = 0, \end{cases} \quad (3)$$

$$h_{\text{bw}} = \begin{cases} \overleftarrow{h}_s - \overleftarrow{h}_{e+1}, & e < L-1, \\ \overleftarrow{h}_s, & e = L-1, \end{cases} \quad (4)$$

where h_{fw} and h_{bw} are respectively the forward and backward differential representations of the token; L represents the length of sequence T, meanwhile s and e are the start and end positions of the token.

Step III: Hybrid Embedding Fusion. Firstly, h_{char} are linearly projected to obtain the character-level embeddings of all tokens E_{char}, making them align with original WordPiece embedding space. Then, by summing token, position and segment embeddings, we obtain the WordPiece embeddings $E_{\text{WordPiece}}$. Finally, summing both WordPiece and character-level embeddings of all tokens, we have the hybrid embeddings

$$E = E_{\text{WordPiece}} + E_{\text{char}}. \quad (5)$$

In `WADBERT`, HGE replaces the original WordPiece embedding layers in both URLBERT and SecBERT. This enables `WADBERT` to integrate subword-level semantic information with fine-grained character-level features, enhancing the robustness of model in handling symbol-dense and irregular textual patterns within HTTP requests. Consequently, all these designs lead to better performance in web attack detection.

3.2. URL Feature Extraction Module

We first provide an overview of URLBERT used to learn URL representation, then describe the process of URL feature extraction.

3.2.1. URLBERT Model. URLBERT [23] is a BERT-based model specifically designed for URL representation learning. It is pre-trained on a large-scale corpus of three billion URLs through three pre-training tasks: masked language modeling (MLM), self-unsupervised contrastive learning (SSCL) and virtual adversarial training (VAT). MLM helps the model learn contextual representations by predicting masked tokens within URL sequences. SSCL strengthens the ability of model to discriminate subtle variations in URLs by aligning different augmented views of the same instance. VAT improves robustness and generalization by enforcing consistency between original and adversarial embeddings through KL divergence. The experiment results show that URLBERT achieves superior performance on various URL-related tasks.

3.2.2. URL Feature Extraction.. The URL feature extraction process consists of three parts: data preprocessing, embedding construction, and contextual encoding with URLBERT.

Data Preprocessing. This stage consists of three steps. Firstly, we normalize the URL by collapsing consecutive slashes ("//") into a single slash ("/"). Next, we convert all

characters to lowercase, eliminating inconsistencies caused by case-sensitive variations. Lastly, we explicitly prepend the request method (e.g., GET, POST, PUT, DELETE) to the URL, as it is strongly associated with malicious behaviors.

Embedding Construction. After preprocessing, WADBERT employs HGE to generate enriched URL embeddings. Specifically, the URL is first tokenized into a sequence of tokens with WordPiece. Then, the sequence is wrapped with two special tokens [CLS] and [SEP]. For each token, HGE derives a character-level embedding from its forward and backward differential representations. Then, these embeddings are linearly projected into the WordPiece space, whose outputs are added to the original WordPiece embeddings. By Equation (5), this process produces the URL embeddings E_{URL}.

Contextual Encoding. The embeddings E_{URL} are fed into the URLBERT for semantic encoding. Specifically, E_{URL} are processed through multiple stacked layers within URLBERT. Note that each layer consists of a multi-head attention mechanism and a position-wise feed-forward neural network, both followed by residual connections and layer normalization. The multi-head attention mechanism enables each token to capture contextual dependencies by attending to other tokens in the sequence. Meanwhile, the position-wise feed-forward network introduces non-linear transformations, which enhance the representational capacity of model. URLBERT iteratively enhances the contextual representations of the tokens by stacking these layers, yielding richer semantic features.

Ultimately, the encoding process produces a sequence of hidden states $\{U_0, U_1, U_2, \ldots, U_n\}$. Notably, the hidden state of [CLS], U_0, represents the global semantic feature of the URL, denoted as f_{URL}. It is subsequently employed for the downstream classification task.

3.3. Payload Feature Extraction Module

We firstly review the SecBERT model for learning parameter representation, then detail the process of payload feature extraction.

3.3.1. SecBERT Model. SecBERT [24] is a BERT-based model to enhance the understanding of cybersecurity-related terms and contexts. It is pre-trained using the MLM task on datasets, such as APTnotes, Stucco-Data threat records and CASIE vulnerability events. Besides, its primary hyperparameters are the same as BERT, only with one difference in the vocabulary and embedding layer weights. Their experimental results demonstrate that the fine-tuned SecBERT model achieves superior performance on various downstream cybersecurity tasks.

3.3.2. Payload Feature Extraction. The payload feature extraction process consists of three stages: data preprocessing, per-parameter encoding, and parameter-level fusion.

Data Preprocessing. At this stage, payload parameters first undergo recursive URL decoding. Specifically, we decode all valid UTF-8 characters repeatedly until their decoded outputs are unchangeable (e.g., %253C $\rightarrow$ <), and meanwhile retain all illegal or incomplete encodings (e.g., %00) straightforwardly. Then, we utilize the Unicode normalization to convert those full-width characters to half-width characters. This reduces semantic variations caused by character variants. Lastly, we build a list with these decoded parameters such that each element corresponds to a payload parameter (key–value pair).

Per-parameter Encoding. We use SecBERT to encode parameters independently. Specifically, each parameter is first tokenized with WordPiece, then its output sequence is wrapped with [CLS] and [SEP]. For each token, HGE first computes a character-level embedding using its forward and backward differential representations, and linearly projects the embedding into the WordPiece embedding space. Then, we sum all outputs and original WordPiece embeddings to generate enriched token embeddings. Lastly, we feed these embeddings into the Transformer encoder of SecBERT to extract the hidden state of [CLS] as the parameter feature. Repeating this process for all parameters yields a parameter-level feature list $P = [P_1, P_2, \ldots, P_n]$.

Parameter-level Fusion. Now, let P be fed into a multi-head attention mechanism without positional encoding. Then, aggregate its output via average pooling to generate a unified semantic feature of payload f_{payload}. Note that P is treated as an unordered list without positional information. Therefore, the multi-head attention mechanism captures combinatorial relationships among parameters only using their features. This ensures that our modeling remains independent of the parameter order.

Besides, the multi-head attention mechanism enables WADBERT to dynamically focus on the most influential parameters within the payload. Specifically, it assigns higher attention weights to parameters that contribute more to the detection decision. In other words, by analyzing these attention weight distributions, we can further identify which parameters are likely to be malicious, thereby enhancing the interpretability of the detection results. This property is particularly practical in attack analysis and security response.

3.4. Feature Fusion and Classification Module

In this section, we describe the process of feature fusion and classification. We first fuse the URL and payload features, then feed the fused feature into a classifier to detect web attacks.

Specifically, we obtain the semantic features of the URL and the payload, denoted as f_{URL} and f_{payload}, which are output by the URL and payload feature extraction modules, respectively. Then, we concatenate f_{URL} and f_{payload} to yield a fused feature:

$$f = [f_{\text{URL}}; f_{\text{payload}}], \tag{6}$$

where $[\,;]$ denotes the vector concatenation operator and f is a comprehensive representation of the HTTP request.

Then, pass the fused feature f to a fully connected neural network for classification defined in Equation (7).

The classifier consists of a linear layer with an input dimension of $hidden\_size \times 2$ (i.e., 768×2), and an output dimension of 2, which corresponds to the classes `benign` and `malicious`.

$$\hat{y} = \text{Softmax}(Wf + b), \tag{7}$$

where W and b denote the weight and bias of the classifier, and $\hat{y}$ represents the categorical probability distribution predicted by the model.

The optimization objective is to minimize the cross-entropy loss defined in Equation (8), which enables the model to optimize the network weights, and further allows the URL and payload feature extraction modules to learn collaboratively. As a result, the performance of web attack detection is improved.

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{8}$$

where y_i represents the true label, $\hat{y}_i$ denotes the predicted probability of class i, and K is the total number of classes. In our web attack detection task, we take $K = 2$.

4. Evaluation

In this section, we conduct experiments to validate the advantages of `WADBERT` and answer the following questions:

- **RQ1:** Can `WADBERT` efficiently converge during training while maintaining a high accuracy in testing?
- **RQ2:** How effective is `WADBERT` compared with existing methods?
- **RQ3:** How do different embedding methods affect the detection performance of `WADBERT`?
- **RQ4:** How do the various components of `WADBERT` contribute to its overall performance?
- **RQ5:** How effective is `WADBERT` compared with a variant that concatenates parameters in their original order?
- **RQ6:** Can `WADBERT` effectively identify malicious parameters by leveraging multi-head attention?

4.1. Experiment Setup

Now we detail the experimental setup, including the datasets, baselines, evaluation metrics and configurations.

Datasets. We evaluate `WADBERT` on two benchmark datasets: CSIC2010 [25] and SR-BH2020 [26]. CSIC2010 includes 36,000 benign and 21,065 malicious HTTP requests, covering attacks such as SQL injection, XSS, CRLF injection, and parameter tampering. SR-BH2020 is collected from a WordPress server monitored by ModSecurity. After manual validation and deduplication, it comprises 161,334 benign and 345,942 malicious requests, including SQL, code, OS command injections, and path traversal. Both datasets are split into 70% for training and 30% for testing.

Baselines. We select the following six advanced models, categorized into traditional deep learning models and Transformer-based models. For clarity, we also summarize the differences of web attack detection models in Table 1.

1) **Traditional deep learning models**
   - **EDL** [20]: It converts the concatenated URL-payload sequences into TF-IDF and CBOW feature vectors, then processes them through MRN, LSTM, and CNN, and fuses the outputs via an MLP classifier.
   - **CNN-BiLSTM** [22]: It encodes the concatenated URL-payload sequences with char embedding, then extracts features via CNN and BiLSTM, and classifies through a fully-connected layer.
2) **Transformer-based models**
   - **BERT-BiLSTM** [14]: It represents the concatenated URL-payload sequences with the pre-trained BERT, then models sequential dependencies via BiLSTM, and classifies through MLP .
   - **DistilBERT** [15]: It encodes the concatenated URL-payload sequences with DistilBERT, then uses the representation of the [CLS] token for classification through a fully-connected layer..
   - **TransURL** [16]: It extracts features from concatenated URL-payload sequences via CharBERT [41], then enhances all layer representations with dilated convolutions and the spatial pyramid attention, and classifies through global average pooling followed by a linear layer..
   - **PMANET** [17]: It transforms the concatenated URL-payload sequences with CharBERT, then refines multi-layer outputs via CBAM-based [42] channel attention and spatial pyramid pooling, and classifies through a fully-connected network.

The above baselines take the concatenated URL and payload sequence as input. Therefore, they cannot model unordered parameters and locate specific malicious parameters.

Evaluation Metrics. We employ four evaluation metrics to assess the performance of model (accuracy, precision, recall and F1-score).

Implementation. We implement `WADBERT` and all baselines in PyTorch on Python 3.9. Experiments are conducted on a server running Ubuntu 18.04 with an NVIDIA A100 GPU, and 64 GB RAM. The model is trained for 10 epochs with a batch size of 32, using the AdamW optimizer. The initial learning rate is set to 2e-5, with a 1% linear warmup followed by a linear decay schedule. The URLBERT and SecBERT encoders adopt BERT-base default configurations [19] with vocabulary sizes of 5K and 52K, respectively. The HGE module employs character-level embeddings (dimension = 768, vocab size = 416) and a single-layer Bi-GRU (hidden size= 384). The multi-head attention module for payload parameters fusion uses 12 attention heads (head size = 64, intermediate size = 3072) with GELU activation.

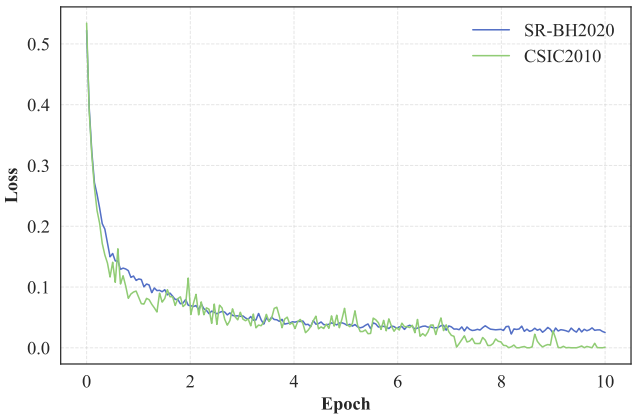


Figure 4: Loss curves during the training process.

TABLE 1: Comparison of differences of web attack detection models.

Model	Multi-granularity Embedding	Combinatorial Relationships	Attack Traceability	Robustness
EDL [20]	✗	✗	✗	✗
CNN-BiLSTM [22]	✗	✗	✗	✗
TransURL [16]	✓	✗	✗	✓
BERT-BiLSTM [14]	✗	✗	✗	✗
DistilBERT [15]	✗	✗	✗	✗
PMANET [17]	✓	✗	✗	✓
WADBERT	✓	✓	✓	✓

4.2. Performance of Proposed Model (RQ1)

In this section, we evaluate the performance of the proposed WADBERT model. We first a nalyze t he convergence of training loss, then discuss the testing performance.

Training Loss Convergence. Figure 4 shows the average training loss on the CSIC2010 and SR-BH2020 datasets across epochs, which demonstrates that WADBERT achieves efficient convergence during training. As the number of epochs decreases, the loss steadily decreases and gradually converges. This indicates that the model effectively captures the patterns within the data during training.

Testing Performance. Table 2 reports the performance of WADBERT on the test sets of CSIC2010 and SR-BH2020 across different training epochs. As the training loss decreases, the test accuracy on CSIC2010 gradually increases from 97.40% to 99.70%. The precision, recall, and F1-score exhibit a similar upward trend with a slight exception for precision. Similarly, on SR-BH2020, the test accuracy increases from 96.28% to 99.19%. These results demonstrate that WADBERT maintains a high detection accuracy on unseen data.

4.3. Overall Performance Comparison (RQ2)

In this part, we compare the performance of WADBERT with existing deep learning methods on the CSIC2010 and SR-BH2020 datasets.

TABLE 2: Performance of WADBERT model across different epochs on test sets.

Dataset	Epoch	Accuracy	Precision	Recall	F1-score
CSIC2010	1	97.40%	99.79%	93.87%	96.74%
	5	99.34%	99.69%	98.71%	99.20%
	10	99.70%	99.87%	99.38%	99.63%
SR-BH2020	1	96.28%	96.73%	96.26%	96.49%
	5	98.98%	99.40%	98.95%	98.18%
	10	99.32%	99.72%	99.29%	99.50%

The results, as shown in Table 3, show that WADBERT can distinguish effectively between benign and malicious requests. From these result, we can make the following observations:

1) Among the traditional deep learning models, EDL performs better than CNN-BiLSTM. Its strength lies in the parallel use of CNN, LSTM and MRN. This enables it to capture global, sequential and hierarchical features simul-taneously. In contrast, CNN-BiLSTM relies on a single CNN–LSTM pipeline, resulting in less comprehensive features and weaker representations of
2) HTTP requests. Transformer-based models treat the concatenated URL and payload sequence as a single text input, ignoring the unordered nature of payload parameters. This causes unrelated parameters to interfere with each other during the attention calculation, and mistakes parameter permu-tations as meaningful changes. Thus, these models fail to capture combinatorial relationships among payload parameters. From a performance perspective, DistilBERT performs better on the CSIC2010 dataset, and BERT-BiLSTM achieves superior results on the SR-BH2020 dataset.
3) On the CSIC2010 dataset, WADBERT attains an accuracy of 99.70% and an F1-score of 99.63%, outperforming all baselines. Compared with the best Transformer-based model DistilBERT, WADBERT improves accuracy by 0.52% and F1-score by 0.64%.
4) On SR-BH2020, a dataset that contains more diverse and realistic payload patterns, WADBERT achieves 99.32% accuracy and 99.50% F1-score, with an improvement of 1.65% in accuracy and 1.23% in F1-score over the strongest baseline.

The overall results highlight that the effectiveness of our dual-encoder architecture, which enables WADBERT to detect subtle anomalies in complex HTTP requests. We achieve this by capturing fine-grained semantic patterns and modeling combinatorial relationships among parameters. Furthermore, our attention weight analysis strategy enhances interpretability by pinpointing malicious payload parameters. These advantages make WADBERT highly applicable to the real-world web application firewall scenarios, offering both high accuracy and interpretability.

4.4. Impact of Embedding Methods (RQ3)

This section analyzes the impact of four different embedding methods (i.e., word embedding, WordPiece embed-

TABLE 3: Comparison results of different methods on CSIC2010 and SR-BH2020 datasets

Datasets	CSIC2010 Dataset				SR-BH2020 Dataset			
Method	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Precision**	**Recall**	**F1-score**
EDL [20]	99.13%	99.45%	98.42%	98.93%	96.67%	99.48%	95.62%	97.51%
CNN-BiLSTM [22]	96.72%	96.19%	95.81%	96.00%	96.27%	98.22%	96.27%	97.24%
BERT-BiLSTM [14]	98.38%	97.84%	98.23%	98.04%	97.67%	99.56%	97.02%	98.27%
DistilBERT [15]	99.18%	99.82%	98.16%	98.99%	96.84%	99.38%	95.97%	97.64%
TransURL [16]	99.15%	99.88%	98.05%	98.95%	96.72%	99.62%	95.56%	97.55%
PMANET [17]	98.17%	99.14%	96.38%	97.74%	96.61%	99.21%	95.80%	97.47%
WADBERT	**99.70%**	**99.87%**	**99.39%**	**99.63%**	**99.32%**	**99.72%**	**99.29%**	**99.50%**

ding, char embedding and HGE) on detection performance. Recall that word embedding uses a regex-based tokenizer, while WordPiece embedding employs a same tokenization as BERT. Additionally, char embedding represents each individual character as a separate token. Our HGE fuses fine-grained character information with subword-level semantic, preserving both symbolic information and meaningful semantic content. A performance comparison of embedding methods across CSIC2010 and SR-BH2020 datasets is presented in Figure 5.

On the CSIC2010 dataset, char embedding achieves the best overall performance, which is followed by the HGE and WordPiece embedding. HGE achieves improvements of 0.09% in accuracy and 0.11% in F1-score over WordPiece embedding. This indicates that integrating character-level information can effectively enhance robustness and representation capacity of our WADBERT model. In contrast, word embedding performs poorest, reflecting its limited ability to represent the symbolic and irregular patterns appearing in HTTP requests.

On the SR-BH2020 dataset, our HGE achieves the best results, with an accuracy of 99.32% and an F1-score of 99.50%. These scores surpass WordPiece embedding by 0.17% and 0.12%, respectively. This shows that combining subword and character features can effectively capture complex HTTP request patterns. Whereas, char embedding exhibits a noticeable performance decline compared to its strong results on CSIC2010. This suggests that pure character-level representations fall short in capturing richer semantic dependencies and contextual patterns. Moreover, word embedding consistently yields the weakest performance across all evaluation metrics.

Overall, all these results show that HGE achieves excellent performance compared to other embedding methods. Although char embedding performs well on CSIC2010, its performance degrades on SR-BH2020. The reason is that char embedding uses single characters as tokens, weakening the ability of capturing semantic structure. HGE addresses this by preserving fine-grained symbolic variations (via char embedding) without disrupting the semantic integrity of words (via WordPiece embedding). In other words, WordPiece embedding provides sufficient representation for plain texts. For irregular strings, character-level information enhances adaptability. Moreover, char embedding increases training time since it needs to generate longer token sequences. In contrast, HGE uses a lightweight GRU to incorporate character features efficiently, maintaining fine-grained details without extending token sequences.

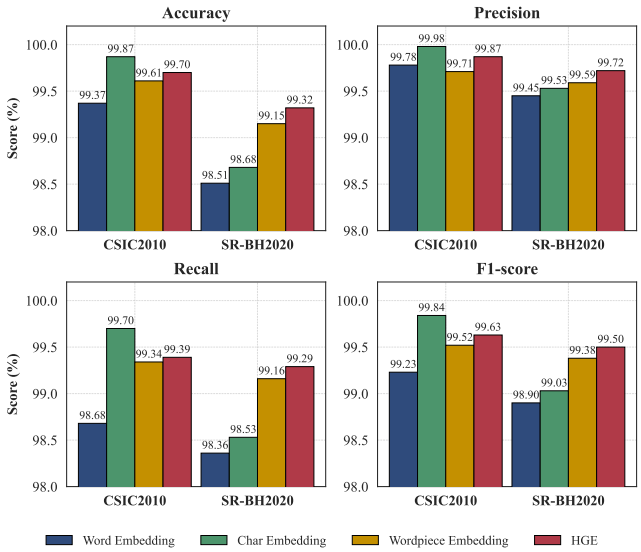


Figure 5: Performance comparison of different embeddings method.

4.5. Module Ablation Study (RQ4)

In this section, we assess the contribution of individual components in WADBERT. We construct single-channel models by using only URL information (WADBERT w/o SecBERT) or only payload information (WADBERT w/o URLBERT). Their performance on the CSIC2010 and SR-BH2020 is compared with the dual-channel WADBERT, as summarized in Figure 6.

On the CSIC2010 dataset, WADBERT without URLBERT achieves a high accuracy of 90.30% and an F1-score of 86.67%. The reason is that the payload parameters contain rich attack features, such as `alert('XSS')` and `OR' 1=1`, etc. In contrast, WADBERT without SecBERT exhibits low accuracy of 73.63% and F1-score of 75.40%. This reflects that the information in URL paths is limited, and the model lacks access to payload parameters. However,

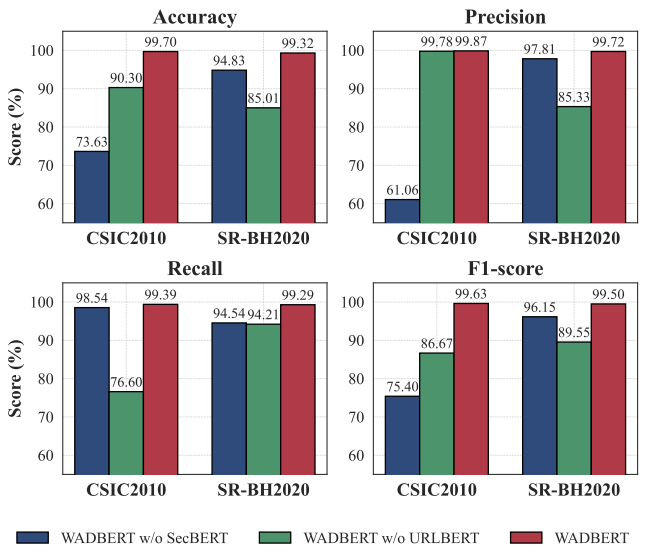


Figure 6: Performance evaluation of WADBERT modules

WADBERT without URLBERT demonstrates limited effectiveness with a recall of only 76.70%, as it fails to detect attacks occurring in URL paths (e.g., path traversal attacks).

Their comparison results reverse on SR-BH2020. Namely, WADBERT without SecBERT outperforms WADBERT without URLBERT. This is because SR-BH2020 contains many RESTful API requests, so attackers can inject malicious content directly into URL path segments. For example, consider a legitimate API endpoint like "/blog/{id}/uploads/{filename}", where an attack may manifest as "/blog/<script>alert(1);</script>/uploads/test.jpg". In this case, attackers can inject an XSS payload directly into the URL path instead of the parameters.

The experimental results confirm that our dual-channel architecture effectively improves overall detection performance. Models only relying on payload or URL cannot fully cover all attack types. By fusing both URL and payload information, WADBERT can capture abnormal URL access paths and malicious payload parameters.

4.6. Effect of Payload Order (RQ5)

We introduce FlatPayload, a comparative model to evaluate the effect of payload order. The model employs the same encoder backbone and classifier as in WADBERT. However, it concatenates all parameters sequentially into a single text input for SecBERT encoding, and removes the multi-head attention mechanism that captures inter-parameter combinatorial relationships.

The experimental results reveal a performance gap between the two models across both datasets, as shown in Figure 7. On the CSIC2010 dataset, WADBERT outperforms FlatPayload, achieving higher accuracy, precision, and F1-score, while FlatPayload exhibits a noticeable drop in precision. Similarly, on the SR-BH2020 dataset, WADBERT consistently demonstrates superior detection performance.

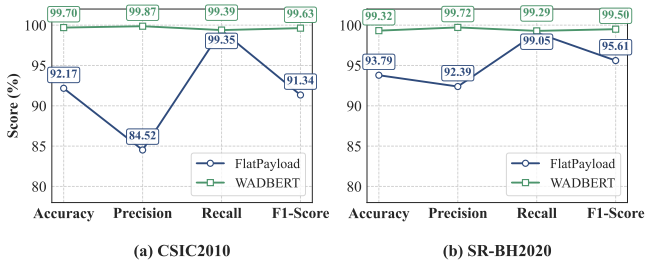


Figure 7: Performance comparison between WADBERT and FlatPayload.

The results show that order-sensitive models (FlatPayload) produce more false positives than order-independent ones (WADBERT). This highlights the significance of capturing inter-parameter combinatorial relationships for web attack detection.

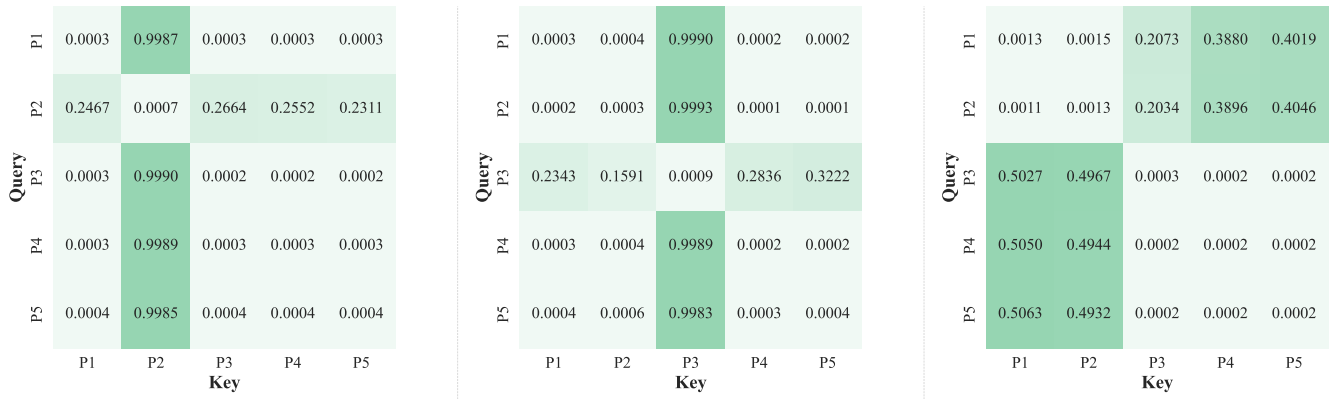
From a protocol perspective, HTTP request parameters are inherently unordered, and the server processes the request without relying on the parameter order. In other word, attackers can freely permute the parameter order without affecting the attack logic. Therefore, the models that rely on sequential input (FlatPayload) are sensitive to such perturbations. They potentially interpret order variations as semantic differences, leading to misclassifications. In contrast, WADBERT uses multi-head attention to capture inter-parameter combinatorial relationships, enhancing robustness to variations of parameter order.

4.7. Interpretability Analysis (RQ6)

We first describe how WADBERT identifies malicious parameters using attention mechanisms, then present visual examples for different attack types.

The multi-head attention mechanism enhances interpretability by indicating which parameters the model focuses on during classification. Specifically, for each attention head, we first linearly project the parameter-level features P (defined in §3.3) into $Query$ and Key matrices. Then, we compute the attention weights by calculating the scaled dot-product between the $Query$ and Key matrices, and apply the $Softmax$ function to normalize the results [39]. Multiple attention heads apply this process in parallel, with each head capturing combinatorial relationships among parameters in different subspaces. Lastly, we average the outputs of all attention heads to generate an aggregated attention weight matrix, which can be visualized as a heatmap. This heatmap indicates which parameters are focused on during parameter-level feature fusion.

We visualize representative attack samples to illustrate how the heatmaps highlight malicious parameters. In the XSS sample, WADBERT focuses almost all attention on $P2$, which contains a malicious function (`alert`) as shown in Figure 8(a). Ordinary parameters like `id=2` and `precio=39` receive minimal attention. Similarly, in the SQLi sample, WADBERT concentrates almost entirely on $P3$, which contains a malicious segment (`or 'a='a'`),



(a) The XSS Attack Sample **(b) The SQLi Attack Sample** **(c) The Composite Attack Sample**

Figure 8: Attention weight distributions of attack samples.

while ignoring other benign parameters (Figure 8(b)). Moreover, in composite attacks sample, WADBERT distributes its attention primarily across multiple malicious parameters, such as XSS ($P1$) and command injection ($P2$), as shown in Figure 8(c). These observations indicate that our model effectively identifies malicious parameters by assigning them higher attention weights.

To further validate this ability, we quantify the contribution of each parameter by aggregating the multi-head attention weights. Specifically, we first average the attention matrices across heads, followed by a column-wise averaging of the resulting matrix. This derives the attention degree of each parameter during the parameter-level feature fusion. This measure represents the cumulative attention that a parameter receives when all parameters act as queries, reflecting the relative importance of parameter in the decision-making process. Taking the XSS as an example in Figure 8(a), the attention degrees of these five parameters are 0.0496, 0.7991, 0.0535, 0.0513, and 0.0465, respectively. The attention degrees clearly indicate that $P2$ dominates the attention distribution. The overall results demonstrate that the multi-head attention mechanism is capable of accurately localizing the attack parameters, validating its effectiveness in traceability of attack.

5. Conclusion

In this paper, we propose WADBERT, a dual-channel web attack detection model, which achieves high detection accuracy while improving interpretability of detection results. Firstly, WADBERT employs HGE to embed URLs and payloads, enhancing robustness against irregular or obfuscated HTTP requests. Secondly, to capture combinatorial relationships among unordered parameters, WADBERT introduces a multi-head attention mechanism. Besides, an attention weight analysis strategy is proposed to identify malicious parameters, which improves interpretability of prediction results.

Evaluations on the CSIC2010 and SR-BH2020 datasets show that WADBERT achieves excellent performance with accuracy of 99.70% and 99.23%, respectively. Ablation studies further confirm that our designed components (e.g., HGE, multi-head attention mechanism and dual-channel fusion) are indeed effective. Our future work will focus on (1) expanding the training data with larger and more diverse real-world datasets to improve generalization; (2) utilizing adversarial training to enhance robustness against sophisticated attacks.

References

[1] M. Wessels, S. Koch, G. Pellegrino, and M. Johns, "Ssrf vs. developers: a study of ssrf-defenses in php applications," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6777–6794.

[2] P. Chen, J. Chen, M. Zhang, Q. Wang, Y. Zhang, M. Xu, and H. Duan, "Cross-origin web attacks via http/2 server push and signed http exchange." in *NDSS*, 2025.

[3] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, "A survey on malware detection with graph representation learning," *ACM Comput. Surv.*, vol. 56, no. 11, 2024.

[4] S. A. Mirheidari, S. Arshad, K. Onarlioglu, B. Crispo, E. Kirda, and W. Robertson, "Cached and confused: Web cache deception in the wild," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 665–682.

[5] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, "Detecting malicious urls using lexical analysis," in *International conference on network and system security*. Springer, 2016, pp. 467–482.

[6] T. Li, G. Kou, and Y. Peng, "Improving malicious urls detection via feature engineering: Linear and nonlinear space transformation methods," *Information Systems*, vol. 91, p. 101494, 2020.

[7] R. Patgiri, A. Biswas, and S. Nayak, "deepbf: Malicious url detection using learned bloom filter and evolutionary deep learning," *Computer Communications*, vol. 200, pp. 30–41, 2023.

[8] R. Smitha, K. S. Hareesha, and P. P. Kundapur, "A machine learning approach for web intrusion detection: Mamls perspective," in *Soft Computing and Signal Processing*, J. Wang, G. R. M. Reddy, V. K. Prasad, and V. S. Reddy, Eds. Singapore: Springer Singapore, 2019, pp. 119–133.

[9] O. Chakir, A. Rehaimi, Y. Sadqi, E. A. A. Alaoui, M. Krichen, G. S. Gaba, and A. Gurtov, "An empirical assessment of ensemble methods and traditional machine learning techniques for web-based attack detection in industry 5.0," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 3, pp. 103–119, 2023.

[10] B. A. Tama and S. Lim, "Ensemble learning for intrusion detection systems: A systematic mapping study and cross-benchmark evaluation," *Computer Science Review*, vol. 39, p. 100357, 2021.

[11] M. Krishnan, Y. Lim, S. Perumal, and G. Palanisamy, "Detection and defending the xss attack using novel hybrid stacking ensemble learning-based dnn approach," *Digital Communications and Networks*, 2022.

[12] K. Kuppa, A. Dayal, S. Gupta, A. Dua, P. Chaudhary, and S. Rathore, "Convxss: A deep learning-based smart ict framework against code injection attacks for html5 web applications in sustainable smart city infrastructure," *Sustainable Cities and Society*, vol. 80, p. 103765, 2022.

[13] H. Mohammadian, A. A. Ghorbani, and A. H. Lashkari, "A gradient-based approach for adversarial attack on deep learning-based network intrusion detection systems," *Applied Soft Computing*, vol. 137, p. 110173, 2023.

[14] L. S. Ramos Júnior, D. Macêdo, A. L. Oliveira, and C. Zanchettin, "Logbert-bilstm: Detecting malicious web requests," in *International Conference on Artificial Neural Networks*. Springer, 2022, pp. 704–715.

[15] B. G. Bokolo, L. Chen, and Q. Liu, "Detection of web-attack using distilbert, rnn, and lstm," in *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2023, pp. 1–6.

[16] R. Liu, Y. Wang, Z. Guo, H. Xu, Z. Qin, W. Ma, and F. Zhang, "Transurl: Improving malicious url detection with multi-layer transformer encoding and multi-scale pyramid features," *Computer Networks*, vol. 253, p. 110707, 2024.

[17] R. Liu, Y. Wang, H. Xu, Z. Qin, F. Zhang, Y. Liu, and Z. Cao, "Pmanet: Malicious url detection via post-trained language model guided multi-level feature attention network," *Information Fusion*, vol. 113, p. 102638, 2025.

[18] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1715–1725.

[19] 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, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[20] C. Luo, Z. Tan, G. Min, J. Gan, W. Shi, and Z. Tian, "A novel web attack detection system for internet of things via ensemble classification," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, p. 5810–5818, 2021.

[21] H. Liu, B. Lang, M. Liu, and H. Yan, "Cnn and rnn based payload classification methods for attack detection," *Knowledge-Based Systems*, vol. 163, p. 332–341, 2019.

[22] L. Zhou, W.-C. Yau, Y. Gan, and S.-T. Liong, "E-webguard: Enhanced neural architectures for precision web attack detection," *Computers & Security*, vol. 148, p. 104127, 2025.

[23] Y. Li, Y. Liu, P. Li, Y. Jia, and Y. Wang, "Continuous multi-task pre-training for malicious url detection and webpage classification," 2025. [Online]. Available: https://arxiv.org/abs/2402.11495

[24] Jackaduma, "Secbert is a bert model trained on cyber security text, learned cybersecurity knowledge," 2023. [Online]. Available: https://github.com/jackaduma/SecBERT

[25] A. M. Torrano Gimenez, Perez Villegas, "Csic 2010 http dataset," 2010. [Online]. Available: https://www.isi.csic.es/dataset/

[26] T. Sureda Riera, J. R. Bermejo Higuera, J. Bermejo Higuera, J. A. Sicilia Montalvo, and J. J. Martínez Herráiz, "SR-BH 2020 multi-label dataset," 2022. [Online]. Available: https://doi.org/10.7910/DVN/OGOIXX

[27] N. M. Sheykhkanloo, "Sql-ids: evaluation of sqli attack detection and classification based on machine learning techniques," in *Proceedings of the 8th International Conference on Security of Information and Networks*. ACM, 2015, pp. 258–266.

[28] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "Detection of sql injection based on artificial neural network," *Knowledge-Based Systems*, vol. 190, p. 105528, 2020.

[29] A. Luo, W. Huang, and W. Fan, "A cnn-based approach to the detection of sql injection attacks," in *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*. IEEE, 2019, pp. 320–324.

[30] Y. Fang, Y. Li, L. Liu, and C. Huang, "Deepxss: Cross site scripting detection based on deep learning," in *Proceedings of the 2018 international conference on computing and artificial intelligence*. ACM Press, 2018, pp. 47–51.

[31] A. Moradi Vartouni, M. Teshnehlab, and S. Sedighian Kashi, "Leveraging deep neural networks for anomaly-based web application firewall," *IET Information Security*, vol. 13, no. 4, p. 352–361, 2019.

[32] Y. Pan, F. Sun, Z. Teng, J. White, D. C. Schmidt, J. Staples, and L. Krause, "Detecting web attacks with end-to-end deep learning," *Journal of Internet Services and Applications*, vol. 10, no. 1, pp. 1–22, 2019.

[33] I. Jemal, M. A. Haddar, O. Cheikhrouhou, and A. Mahfoudhi, "Performance evaluation of convolutional neural network for web security," *Computer Communications*, vol. 175, p. 58–67, 2021.

[34] A. Tekerek, "A novel architecture for web-based attack detection using convolutional neural network," *Computers & Security*, vol. 100, p. 102096, 2021.

[35] W. B. Shahid, B. Aslam, H. Abbas, S. B. Khalid, and H. Afzal, "An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling," *Journal of Network and Computer Applications*, vol. 198, p. 103270, 2022.

[36] 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, p. 1963–1971, 2020.

[37] I. Jana and A. Oprea, "Appmine: Behavioral analytics for web application vulnerability detection," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW)*, 2019, pp. 69–80.

[38] N. Stevanović, B. Todorović, and V. Todorović, "Web attack detection based on traps," *Applied Intelligence*, vol. 52, no. 11, pp. 12 397–12 421, 2022.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, 2017, pp. 5998–6008.

[40] K. Cho, B. van Merriënboer, C. Glcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[41] W. Ma, Y. Cui, C. Si, T. Liu, S. Wang, and G. Hu, "Charbert: Character-aware pre-trained language model," *arXiv preprint arXiv:2011.01513*, 2020.

[42] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.