Hindawi
IET Information Security
Volume 2024, Article ID 5565950, 16 pages
https://doi.org/10.1049/2024/5565950

Research Article

Deep Learning in Cybersecurity: A Hybrid BERT–LSTM Network for SQL Injection Attack Detection

Yixian Liu and Yupeng Dai

Xi'an University of Posts and Telecommunications, Xi'an 710000, China

Correspondence should be addressed to Yixian Liu; liu-yi-xian@xupt.edu.cn

Received 30 November 2023; Revised 19 March 2024; Accepted 21 March 2024; Published 5 April 2024

Academic Editor: Taimur Bakhshi

In the past decade, cybersecurity has become increasingly significant, driven largely by the increase in cybersecurity threats. Among these threats, SQL injection attacks stand out as a particularly common method of cyber attack. Traditional methods for detecting these attacks mainly rely on manually defined features, making these detection outcomes highly dependent on the precision of feature extraction. Unfortunately, these approaches struggle to adapt to the increasingly sophisticated nature of these attack techniques, thereby necessitating the development of more robust detection strategies. This paper presents a novel deep learning framework that integrates Bidirectional Encoder Representations from Transformers (BERT) and Long Short-Term Memory (LSTM) networks, enhancing the detection of SQL injection attacks. Leveraging the advanced contextual encoding capabilities of BERT and the sequential data processing ability of LSTM networks, the proposed model dynamically extracts word and sentence-level features, subsequently generating embedding vectors that effectively identify malicious SQL query patterns. Experimental results indicate that our method achieves accuracy, precision, recall, and F1 scores of 0.973, 0.963, 0.962, and 0.958, respectively, while ensuring high computational efficiency.

1. Introduction

In the modern digital era, while web applications significantly enhance convenience in our daily lives, they concurrently present multiple cybersecurity vulnerabilities. These applications store a vast array of user data in their databases through network data flows, including user credentials, financial details, personal identifiers, and other confidential data. In the absence of comprehensive security reviews by web application developers, malicious actors may inject harmful code to pilfer such valuable information, thereby posing substantial risks to information security. As the manufacturing industry transitions towards Industry 5.0, characterized by the integration of advanced technologies such as highly integrated cyber-physical systems, artificial intelligence, and the Internet of Things (IoT), cyberattacks pose significant threats, potentially resulting in production downtime, data breaches, and even physical harm [1]. Injection attacks represent a prominent security concern, as highlighted by the Open Web Application Security Project (OWASP) in its 2021 report [2]. Among the various types of injection attacks, SQL injection stands out as one of the most pernicious and prevalent methods.

In instances where the webpage fails to adequately verify or inaccurately verify the security of user-entered or uploaded information, an unauthorized attacker may insert a malicious database query code into the user's webpage request, subsequently transmitting it to the webpage server [3]. In this manner, the database server may be deceived into executing unauthorized queries, thereby acquiring the privacy data. In certain circumstances, malicious actors gain elevated privileges to execute increasingly devastating attacks, including taking control of the entire system [4]. This form of attack has the potential to result in grave security concerns, including privacy breaches, identity theft, and compromised traffic security.

SQL injection attacks are varied and can be surreptitious. They exploit various vulnerabilities, making it challenging to establish consistent inspection or defense mechanisms. Certain specially crafted malicious attack queries can even circumvent firewalls [5, 6]. Addressing these threats often demands the expertise of specialized security personnel,

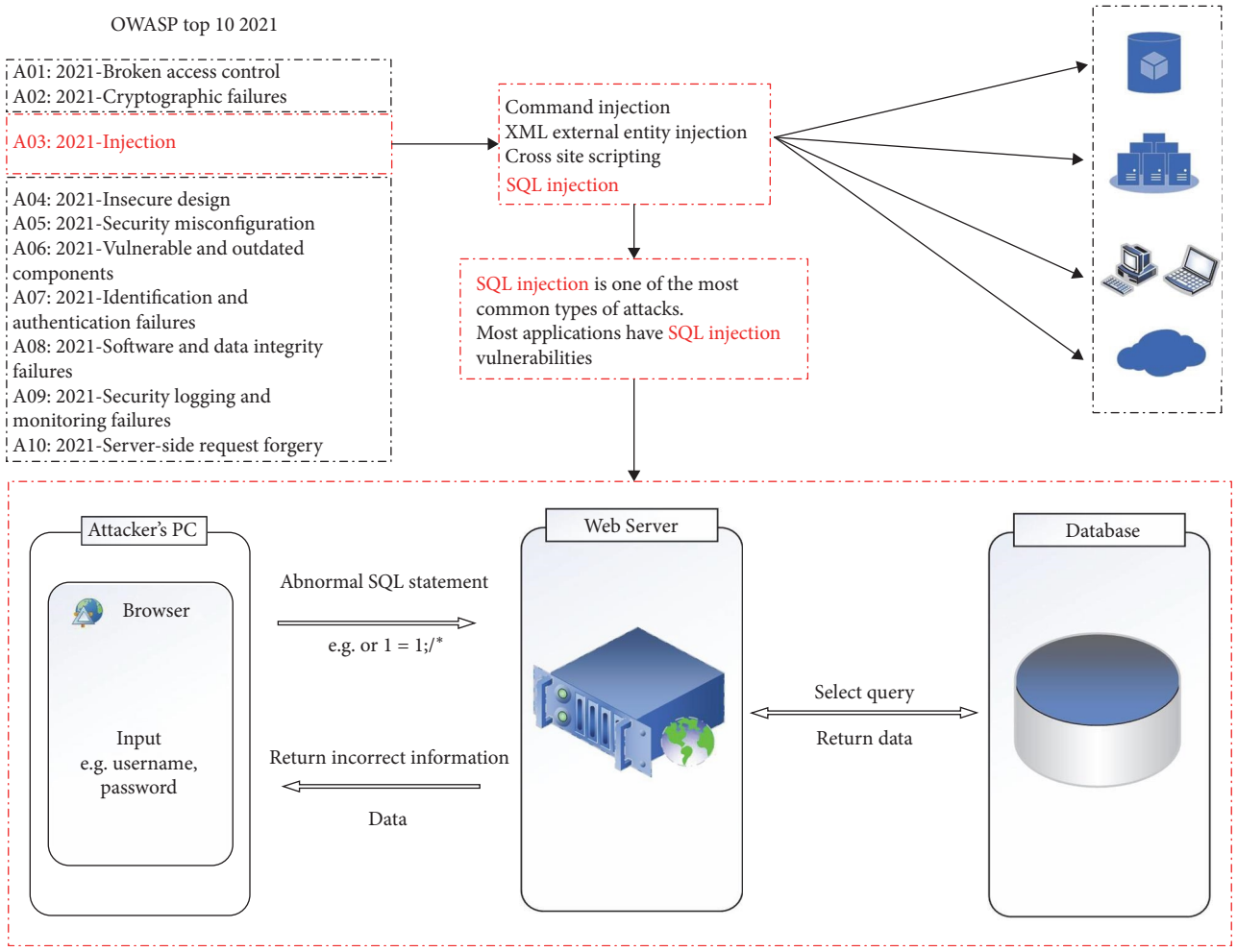

Figure 1: The risks and process of SQL injection.

leading to considerable human resource and time investments, especially when dealing with extensive datasets. Figure 1 underscores the significance of SQL injection in the landscape of security vulnerabilities and provides a comprehensive illustration of the SQL injection process.

Websites with databases integrated into their backend are vulnerable to SQL injection attacks [7]. Malicious queries can be embedded in URL parameters, post data, cookies, and even HTTP header information [8]. Depending on the techniques used to extract information, SQL injection attacks can be categorized into union query injection, error-based injection, Boolean-based blind injection, time-based blind injection, and stacked injection. Depending on the mode of attack, SQL injection can be categorized into nine types: tautology, logical error-based queries, union queries, stored procedures, piggy-backed queries, alternative encoding, inference, blind injection, and timing attacks [9]. Each type of injection demands a tailored solution. Although rule-based web application firewalls (WAF) are prevalent today, attackers often circumvent developer-implemented rules using specially crafted statements. Consequently, rule-based WAFs frequently exhibit high rates of false positives and detection failures.

At present, the primary methods employed for detecting SQL injection attacks encompass static analysis [10], dynamic analysis [11], and a hybrid approach integrating both [12]. Nevertheless, static analysis detection methods are limited to restricting or examining the syntax of user input to determine if an attack has occurred, which can result in certain attacks that adhere to the input rules being easily evaded. Dynamic detection is capable of identifying vulnerabilities that are predefined by application developers. However, due to the assorted syntax of malicious statements, it fails to meet the detection requirements in intricate systems [13].

While pinpointing a shared characteristic across diverse SQL injection payloads can be challenging, a closer examination of their attack principles reveals that malicious SQL queries are invariably embedded within network requests. These SQL queries must also adhere to the database language syntax. Consequently, SQL injection attacks can be identified by leveraging both the malicious SQL injection payload's contextual features and keyword attributes. Furthermore, with the advent of machine learning and deep learning in network security, we introduced a novel network architecture that amalgamates the Bidirectional Encoder Representations

from Transformers (BERT) model with the Long Short-Term Memory (LSTM) network model. Empirical testing confirmed the efficacy of our model in detecting SQL injection attacks.

In the chapters that follow:

Chapter 2 offers a review of the literature, succinctly outlining both traditional and contemporary strategies for identifying SQL injection vulnerabilities. Chapter 3 delineates the theoretical underpinnings and the comprehensive network architecture inherent to our suggested approach. Chapter 4 documents the experimental procedures undertaken and assesses the efficacy of our model. In Chapter 5, the superiority of our method is highlighted by way of comparative experimental analysis. Finally, Chapter 6 culminates with a concise summarization of the study's conclusions.

2. Relevant Work

2.1. Traditional SQL Injection Attack Detection Methods. Fu et al. [10] introduced SAFELI, a static analysis framework designed to identify SQL injection vulnerabilities at compile time. This framework utilizes a hybrid constraint solver to pinpoint malicious user input at the hotspot of each submitted SQL query. However, a significant limitation is its manual operation, rendering it unsuitable for analyzing extensive data streams within brief periods.

Khalid and Yousif [11] presented a unique dynamic SQL injection detection tool, grounded on the differentiation of HTTP requests dispatched by users or clients. Nevertheless, it has the limitation of detecting only a restricted range of SQL injection types.

Prakash and Saravanan [12] devised a mechanism incorporating both static and dynamic analysis tools. During this endeavor, they introduced SQLi Instrumentation (SQLID) as an intermediary virtual database layer situated between the primary database and the application. A notable drawback is the elevated false positive rate stemming from the incorporation of static analysis.

2.2. SQL Injection Attack Detection Method Based on Machine Learning. Kar et al. [14] introduced a detection methodology wherein SQL queries are transformed into token sequences, maintaining their inherent structure. These sequences are then modeled as a graph with tokens as nodes and their interactions represented by weighted edges. Subsequently, an SVM classifier is trained using labeled centrality measures, and this method exhibits commendable performance on the designated dataset.

Abdulhamza and Al-Janabi [15] unveiled a distinct two-dimensional convolutional neural network (2DCNN) architecture. This approach involves the conversion of the dataset into a two-dimensional matrix via the Skip-Gram model and takes into account both the contextual and syntactic aspects of SQL. The experimental outcomes were notably satisfactory.

Zhao et al. [16] crafted a detection model that analyzes the URL and body content within HTTP requests. Time series features, derived from the gate recurrent unit (GRU) neural network, are integrated with discrete features manually curated by experts. These combined features are then fed into fully connected neural networks for classification. Owing to the introduction of a feature fusion method, there was a noticeable enhancement in the detection accuracy.

Jothi et al. [17] and Zhang et al. [18] employed different approaches in data preprocessing compared to ours. Jothi et al. [17] utilized one-hot encoding to generate word index, while Zhang et al. [18] employed the TF–IDF algorithm to assess word importance based on word frequency and inverse document frequency.

The integration of machine learning and deep learning into network security represents a pivotal avenue for future research. Such approaches effectively address the limitations of conventional detection methods, particularly when contending with voluminous data. In the forthcoming discussion section of this article, the method delineated herein will be juxtaposed with the previously mentioned five techniques. Upon comparison, the methodology presented in this study demonstrated superior outcomes both on the identical dataset and in real attack detection experiments.

2.3. Other Novel Methods Applied to Network Intrusion Detection. Zivkovic et al. [19] decreased the false positive rate of the network intrusion detection system by employing an enhanced firefly algorithm to establish optimal hyperparameters for the XGBoost classifier. Similarly, during our experiment, we continuously and meticulously adjusted the network's hyperparameters to secure the most effective detection outcomes.

Crespo-Martínez et al. [20] directed their research toward the data collection process. Their work suggests that in scenarios where inspection of all packets in a high-traffic wide-area network (WAN) is not feasible, effective detection of attacks, such as SQL injection, is still achievable by monitoring flow data using lightweight protocols, like NetFlow. Detecting attacks within high-traffic WANs represent a key future direction for our applications. The methodology presented in this paper offers a new perspective for our endeavors.

Shah et al. [21] developed a secure, lightweight key management framework to ensure network security, prevent unauthorized access, and play a vital role in underwater wireless sensor networks (UWSNs). Ensuring the network's lightweight design has been a pivotal aspect of our research. This strategy aims to bolster computational efficiency, enabling the swift detection of large volumes of data, while simultaneously preserving network compatibility.

3. Detection Model of Hybrid BERT–LSTM Network

3.1. BERT Model. The BERT is a widely employed pretraining model in the discipline of Natural Language Processing (NLP). It was introduced by the research team at Google in 2018, marking a significant advancement in contemporary deep learning within the realm of NLP [22].

The primary characteristic of BERT is its bidirectional encoder architecture, achieved through the use of the Transformer model [23]. In contrast to the conventional unidirectional approach, BERT has the capability to utilize contextual information from both preceding and succeeding directions to predict the semantic understanding of a word within a

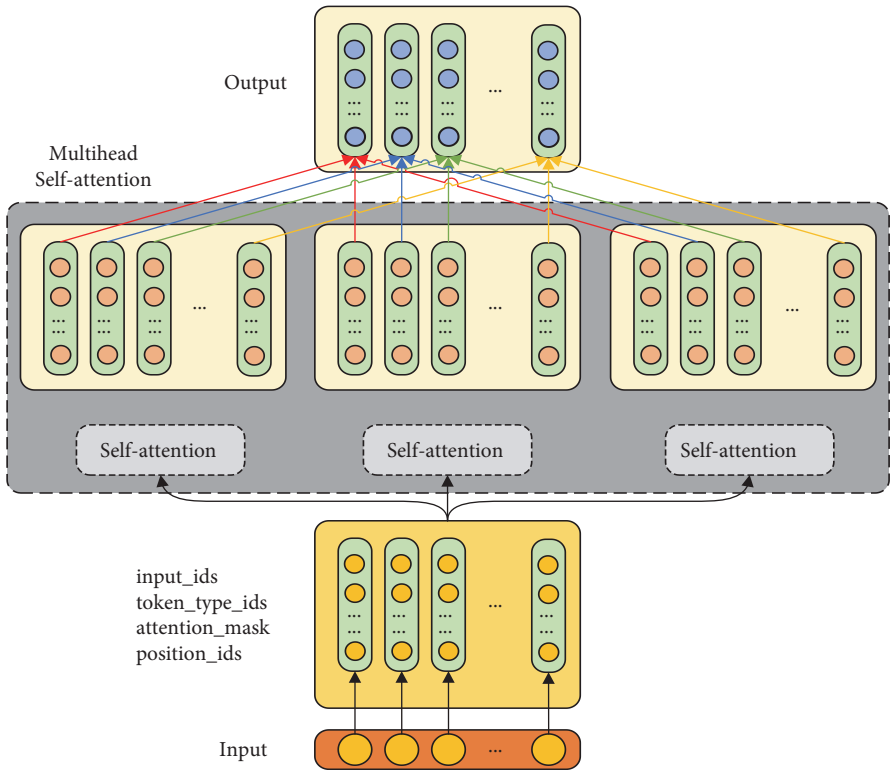

Figure 2: The framework of the BERT model.

sentence. BERT significantly enhances the performance of various natural language processing tasks, including sentence classification, named entity recognition, and textual entailment. The framework of the BERT model is shown in Figure 2.

The BERT model also incorporates a multihead self-attention mechanism, which allows each word to assimilate the contextual semantic information of other words within the sentence. The multihead attention mechanism extends the ordinary attention mechanism by employing multiple independent attention heads simultaneously to process the input sequence. Each attention head learns its own query, a linear transformation of keys and values, and independently performs attention computations. Finally, the results from the multiple attention heads are combined through linear transformation and splicing to obtain the final output. This integration enhances the model's ability to capture the relationships and dependencies between words, thereby improving its understanding of the overall sentence meaning. By incorporating the multihead attention mechanism, the model's expressive capability and generalization performance are improved while the computational efficiency is maintained.

This paper uses the BERT model to generate embedded word vectors for the samples, which are input into the LSTM network to complete the classification task. BERT necessitates that the input data conform to a specific structure, namely "[CLS] sentence [SEP]." Here, "[CLS]" serves as the sentence's starting marker for tasks at the sentence level, and "[SEP]" acts as the delimiter. Following preprocessing, each word is transformed into an embedding vector. BERT, by layering multiple Transformer blocks, leverages the self-attention mechanism to assess the interrelations among all words, thus producing word vectors infused with contextual information. Concurrently, the vector associated with the "[CLS]" marker embodies the representation for the entire input sentence. The self-attention mechanism enables BERT to discern the varied meanings words may hold across different contexts, a capability vital for grasping context-sensitive security threats like SQL injection. It will have a higher detection accuracy than the static word vector generated by traditional Word2vec training.

Word embedding is a significant methodology within the field of NLP to support language modeling and facilitate feature learning techniques [24]. This technique involves the mapping of words or phrases from the vocabulary to vectors consisting of real numbers. Additionally, word embedding serves as a mandatory input format for LSTM networks.

The traditional one-hot vector has two disadvantages. First, the matrix dimension is very large, which wastes space and requires more computing resources. Second, the one-hot matrix only distinguishes different word numbers, and the semantic relationship between words and words cannot be reflected. The embedded word vector assigns a fixed length vector representation to each word. In this study, the model discussed in this article is defined as a length of 128, which marks a significant reduction compared to the conventional one-hot vector. Table 1 presents three-dimensional vector representations of the common database query terms "SELECT," "select," "FROM," and "WHERE."

The semantic similarity between two words in the vector space can be determined by calculating the cosine distance

Table 1: Example of embedded word vector.

Words	Vector 1	Vector 2	Vector 3
SELECT	0.51	0.13	−0.15
select	0.50	0.12	−0.13
FROM	1.37	0.94	−0.05
WHERE	1.56	1.23	−0.09

using Equation (1). When two words have similar semantics, their cosine distance will be smaller [25].

$$\text{similarity} = \cos\theta = \frac{\vec{A} \cdot \vec{B}}{||A|| \cdot ||B||}. \tag{1}$$

From the provided example, the cosine similarity between "SELECT" and "select" stands at 0.2901, while there's 0.8284 between "SELECT" and "FROM." This difference arises because the first pair varies only in the case while retaining identical semantics. Hence, their similarity is lower, indicating a closer proximity in vector space. These four words, which typically appear together in malicious SQL queries within network requests, exhibit similar three-dimensional vector representations. Consequently, they occupy the same octant in the three-dimensional vector space.

3.2. LSTM Model. The LSTM neural network is an optimized network of the recurrent neural network (RNN). RNN stores past information and current input by introducing state variables to determine the current output. Previous input values have an impact on subsequent outputs [26]. Hence, RNN exhibits impressive ability in the processing of sequential data types, including linguistic reasoning texts and trend graphs. Presently, RNN finds its primary usage in various tasks such as fault prediction [27], text modeling [28], load prediction [29], sentiment analysis [30], and speech recognition [31]. The primary objective of detecting SQL injection attack behavior is to identify the presence of SQL injection attack statements in the everyday traffic of web applications. While malicious attack sentences exhibit various forms, they also possess numerous logical features within their contextual context. These characteristics render the utilization of RNNs as classifiers technically feasible.

Figure 3 portrays the overall framework of RNN, wherein A represents a recurrent neuron, x_t signifies the input of a recurrent neuron, and h_t denotes the output of a recurrent neuron. Following the x_t input, a portion of the information is computed and output, while another portion influences the calculation of the subsequent cycle.

It can be seen in Figure 3 that the output h_t of a certain recurrent neuron is jointly affected by the input x_t of this neuron and the hidden state h_{t-1} of the previous neuron. After getting h_t, part of the information is used as output and backpropagated according to the value of loss. Another part of the information is passed to the next recurrent neuron. In instances where the input sentence is excessively lengthy, the neurons that receive input from distinct keywords are considerably distant from each other. Consequently, the neurons located towards the rear lack the capability to sufficiently comprehend the information transmitted by the neurons situated at the front. Consequently, the network is unable to effectively discern the contextual relevance of long-sequence data. This situation caused the RNN gradient disappearance problem, known as the long-term dependency (Long-Term Dependencies) problem. So, the LSTM neural network was produced, which is aimed at the internal improvement and optimization of the recurrent neuron so that the long-term dependency problem can be obtained as an effective solution.

The fundamental principle of LSTM is the same as RNN, as the preceding input value influences the ensuing output. To address the issue of gradient disappearance in RNN, multiple gate calculations are used in the LSTM neuron, accompanied by the introduction of a novel variable denoted as the "cell state" [32].

The feature screening and memory of neurons for temporal input is accomplished by employing several gate calculation units, which additionally facilitate the storage and updating of the cell states. The cell state runs through the entire neural network, which ensures that information flows through all cyclic neural units, and the network can learn long-term dependance on information, improving the problem of gradient disappearance. The input and control signals of each gate computing unit are derived from the output of the preceding neuron, thereby preserving the fundamental properties of RNN [33].

3.2.1. Forget Gate. The forget gate is responsible for determining the retention of information in the cell state. The present neuron is fed with the hidden state h_{t-1} from the previous neuron and the current input x_t, and generates a probability value ranging from zero to one using the sigmoid function. In case the probability value approaches zero, it indicates that the information is less significant and should be disregarded to a greater extent. In Equation (2), W_f and U_f are the coefficients of the linear relationship, whereas b_f denotes the bias term.

$$f_t = \text{sigmoid}\left(W_f h_{t-1} + U_f x_t + b_f\right). \tag{2}$$

3.2.2. Input Gate. The input gate is responsible for determining the information that the cell state receives from the current neuron, as well as the values that require updating. Firstly, the sigmoid function is utilized to calculate the parameter i_t, which indicates the portion of the cell state to be updated. A value approaching one signifies a greater need for updates. Next, the a_t parameter is derived through the tanh function. This parameter signifies the updated value of the cell state. In Equation (3) and (4), W_f and U_f are the coefficients of the linear relationship, whereas b_f denotes the bias term.

$$i_t = \text{sigmoid}\left(W_i h_{t-1} + U_i x_t + b_i\right), \tag{3}$$

$$a_t = \tanh\left(W_a h_{t-1} + U_a x_t + b_a\right). \tag{4}$$

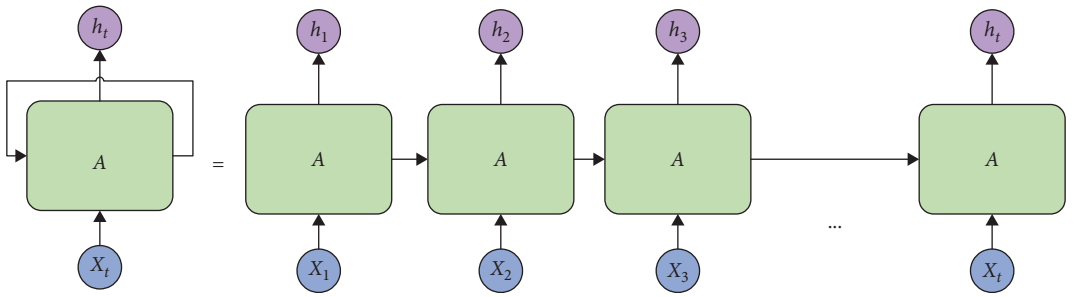

Figure 3: Overall framework of RNN.

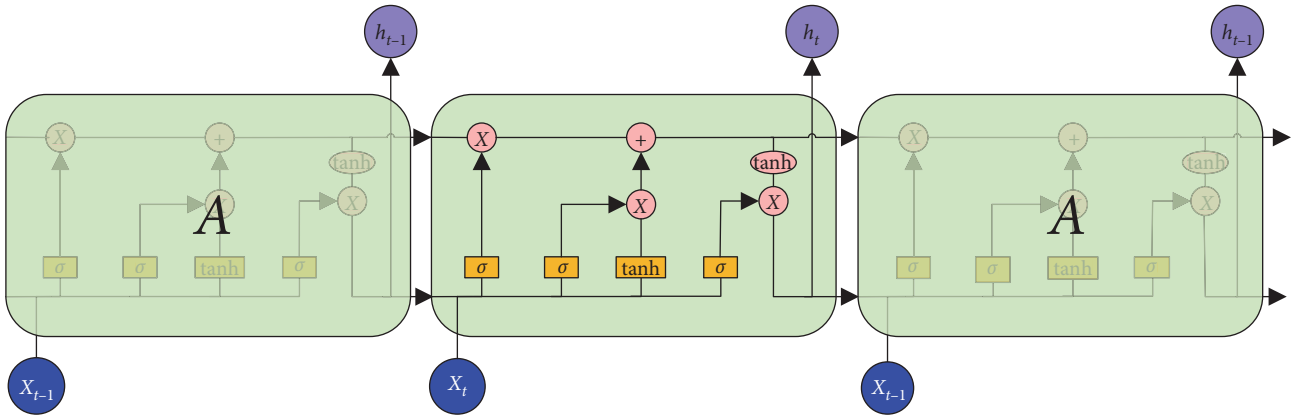

Figure 4: Detailed structure of LSTM units.

3.2.3. Cell State Update. Compute the Hadamard product of the cell state C_{t-1} from the previous neuron and the output f_t of the forge gate to eliminate the information to be forgotten. Next, incorporate the Hadamard product of i_t and a_t as an adjusted value into the previous cell state after forgetting. Equation (5) is as follows. Whereas ⊙ denotes the Hadamard product.

$$C_t = C_{t-1} \odot f_t + i_t \odot a_t. \tag{5}$$

3.2.4. Output Gate. The output gate is responsible for determining the output value of the present unit, which will be obtained after filtering the current cell state. First, o_t is used to indicate the specific information within the cell state that necessitates output. As above, its value is determined by the previous output h_{t-1} and the input x_t of this unit via the utilization of the sigmoid function. A value approaching one signifies a greater need for output. Next, the cell state is substituted into the tanh function to calculate a value between minus one and one. Then, the Hadamard product is applied between this value and o_t. Finally, the resultant output value h_t is derived. Equation (6) and (7) are as follows. Whereas ⊙ denotes the Hadamard product.

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

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

Figure 4 shows the details of the gate computation units within the LSTM neural units and the transmission status of the data flow.

3.3. Overall Framework of the Hybrid BERT-LSTM Network. The generation of embedded vectors is accomplished through the training of the BERT model, subsequently employed as input for the following LSTM networks. In the initial stage, the model's embedding layer integrates the "input_ids," "token_type_ids," and "position_ids" for each word, assigning them random initial values as vectors in a 768-dimensional space. These vectors correspond to word embedding vectors, sentence embedding vectors, and position embedding vectors. Next, the vector resulting from the addition of the three initial vector matrices is fed into the BERT layer. Ultimately, the output comprises two components. The first component uses the word vector associated with the "[CLS]" symbol, which we preinsert before each sentence, to represent the entire sentence as its vector. This element is referred to as the "pooler output" in the network. The second component comprises word vectors for individual words, obtained through word segmentation from the sample, and is referred to as the "sequence output" in the network.

Continue feeding the "sequence output" into the LSTM layer for sustained training. The output should initially be processed through a fully connected layer to acquire a matrix of vectors that match the dimensions of the "pooler output." Subsequently, these two vector outputs are concatenated to

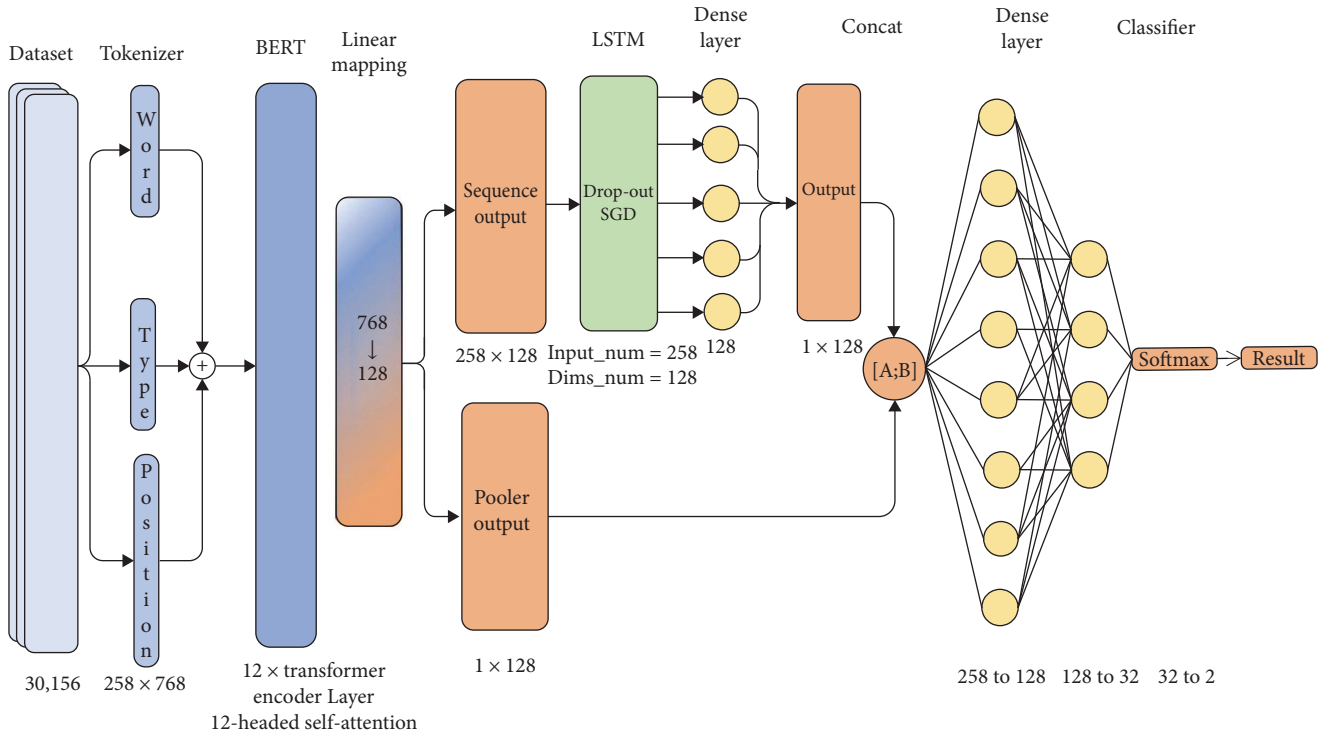

Figure 5: The structure of network.

forge a more intricate feature representation, adept at discerning subtle semantic distinctions. Next, the result of matrix concatenation between these two is input into the double-layer fully connected layer and classifier to obtain the detection results.

The hybrid BERT-LSTM network's overall framework and changes in data dimensions are illustrated in Figure 5. The specific parameters are: the LSTM layer has 128 input nodes and hidden nodes. The parameter of the forget gate is set to 0.1. Simultaneously, the dropout operation is incorporated with a parameter value of 0.5 to mitigate the risk of overfitting during the training process. The BERT part is composed of 12 layers of transformer encoder layers stacked. Each layer is added with 12 head attention mechanisms, and the output vector dimension is 768 dimensions. The dimensions are projected onto 128 dimensions, aligning with the quantity of hidden layer nodes within LSTM. Simultaneously, parameter count is minimized to ensure the model's lightweight design and computational efficiency. This architecture amalgamates BERT's robust linguistic comprehension with LSTM's prowess in processing time series data, empowering the model to comprehend intricate language structures while also discerning underlying sequence patterns. Consequently, it enhances the efficacy and precision of SQL injection attack detection.

4. Experiments and Results

4.1. Experimental Environment. The configuration of the lab environment is shown in Table 2. The BERT model comes from Google Github open source code. The LSTM model is implemented by Keras.

Table 2: Experimental environment.

Platforms	Content
PC system	Windows11
Hardware	Intel® Core™ i7-8565U CPU GTX 1070 GPU 16 GB RAM
Cuda version	Cuda 11.4.1
Software	Python 3.6 Keras 2.06 TensorFlow 1.3.0 Matplotlib 3.3.4 Numpy 1.19.5 Scikit-learn 0.24.2

4.2. Data Collection and Preprocessing. A total of 10,852 instances of malicious SQL injection attack statements were acquired from the Httpparams dataset open source database, serving as positive samples. Additionally, 19,304 samples of normal web page requests made by users were obtained as negative samples. The dataset contains various types of malicious entries, including joint query injection, error-based injection, Boolean-based blind injection, time-based blind injection, stacking injection, and inferential injection. Additionally, we have integrated several injection methodologies

Table 3: Rules of word segmentation.

Category	Examples
Function	sleep (int), char (69)
Equation	1,557 = 1,557, 6,324 = 6,324
Content inside single quotes	"0 : 0:5", "otqy"
Others	&, #, (, /, -, ||

designed to circumvent established rule-based detections, such as synonym substitution, copy bypass, string bypass, and encoding bypass. Concurrently, the benign data segment comprises typical POST and GET requests. It also uniquely incorporates benign user inputs embedded with malicious keywords to evaluate the resilience of the detection algorithms against sophisticated obfuscations. The samples are then divided in a ratio of 6 : 2 : 2 into training, validation, and test sets.

Conduct data normalization on both positive and negative samples, eliminating the host and path details from the URL while retaining only the portion related to the malicious attack payload. Substitute the link with "http://u". The aforementioned data is URL-encoded and stored in a CSV file. Ultimately, all the data were systematically partitioned into distinct sets: 60% for training, 20% for validation, and the remaining 20% for testing.

First of all, conduct a preliminary word segmentation process on each sample. For the payload part of SQL injection, in addition to the common database terminology words: "select," "order," and "union," additionally, establish specific word segmentation rules outlined in Table 3 to handle special cases. These rules encompass function bodies that combine characters with brackets, equations that combine characters with equal signs, special symbols, characters in single quotes, and the quotes surrounding them. The special characters are treated as words through regular expressions and enclosed in single quotes. Count the length and number of occurrences of all words, and establish the maximum length and minimum number of occurrences. All words whose length is less than the maximum length and whose number of occurrences are greater than the minimum number are summarized to create a basic word list.

The BERT model uses the Wordpiece word segmentation algorithm to further divide words. In the event that a word exceeds the maximum length stated in the aforementioned basic word list, or its frequency of occurrence falls below the specified minimum number of times, the word shall undergo further division into subwords. Such as the word "embeddings," it will be divided into subwords ("em," "##bed," "##ding," and "## s"), followed by the symbol "##," which means that it is not a subword at the beginning of a word, which can effectively reduce the problem of out-of-vocabulary (OOV), and low-frequency words can also be well trained. For malicious sentences with uncommon words and the attacker's personal idiomatic grammar, reducing the possibility of false positives. Set an index for each word to form the final word list.

4.3. Tokenize the Sample. Tokenize the positive and negative samples using the BERT's tokenization method to ensure they meet the input requirements of the BERT model. First, Prepend each sentence with the "[CLS]" token and append "[SEP]" at the end.

Then transform all samples into an indexed format based on the vocabulary dictionary, and employ the variable "input_ids" to denote the index assigned to each word. Utilize the variable "token_type_ids" to ascertain the associated sentence sample for each word. Simultaneously, the variable "position_ids" is used to indicate the position of the word in the corresponding sample. With the aim of ensuring uniform sentence length, padding complements all samples to 258 dimensions with the value of "−1." At the same time, the variable "attention_mask" serves to distinguish the actual sample data from the useless padded data. Consequently, subsequent training can bypass this irrelevant data. The process of tokenize is shown in Figure 6.

After obtaining the word vectors and sentence vectors from BERT's output, which are necessary for subsequent processes, these word vectors are further fed into an LSTM network for training to achieve predictions. Following numerous iterations of training and validation, the model's hyperparameters are adjusted until an optimal level is reached and the prediction accuracy converges at a higher level. Consequently, we have a predictive model.

4.4. System Deployment. In subsequent experiments, we tested scenarios in which the detection model was deployed in real-world applications. To enable a more practical implementation of our application, we have developed a straightforward graphical interface that visually presents the monitoring count and the number of alarms generated from malicious detection. The interface is illustrated in Figure 7.

We extracted valuable information from network traffic data sourced from logs and security defense equipment. After processing this data, our prediction model was applied. If malicious activity is predicted, we initiate emergency responses or vulnerability hardening. If the request is determined to be ordinary, it will be executed, and the relevant data will be returned to the user. The deployment process of the model for detection is illustrated in Figure 8.

4.5. Experiment Results. In this section, we present the evaluation data for the Detection Model based on BERT and LSTM. The model was trained on the aforementioned dataset for a duration of 30 epochs. Additionally, to validate the efficacy of the trained model in practical scenarios, we established a platform containing vulnerabilities to simulate attacks and collected relevant netflow data. This data was subsequently utilized for detection purposes.

4.5.1. Experimental Validation of Model Effectiveness on Datasets. In this paper, we use accuracy, recall, and F1 score for model evaluation. Accuracy rate (Acc) is the proportion of correctly categorized samples to the total samples; Recall rate (R) is the proportion of correctly predicted malicious statements among the samples that are actually malicious statements; F1 score metrics is the reconciled average of precision rate and recall rate, which is an important score

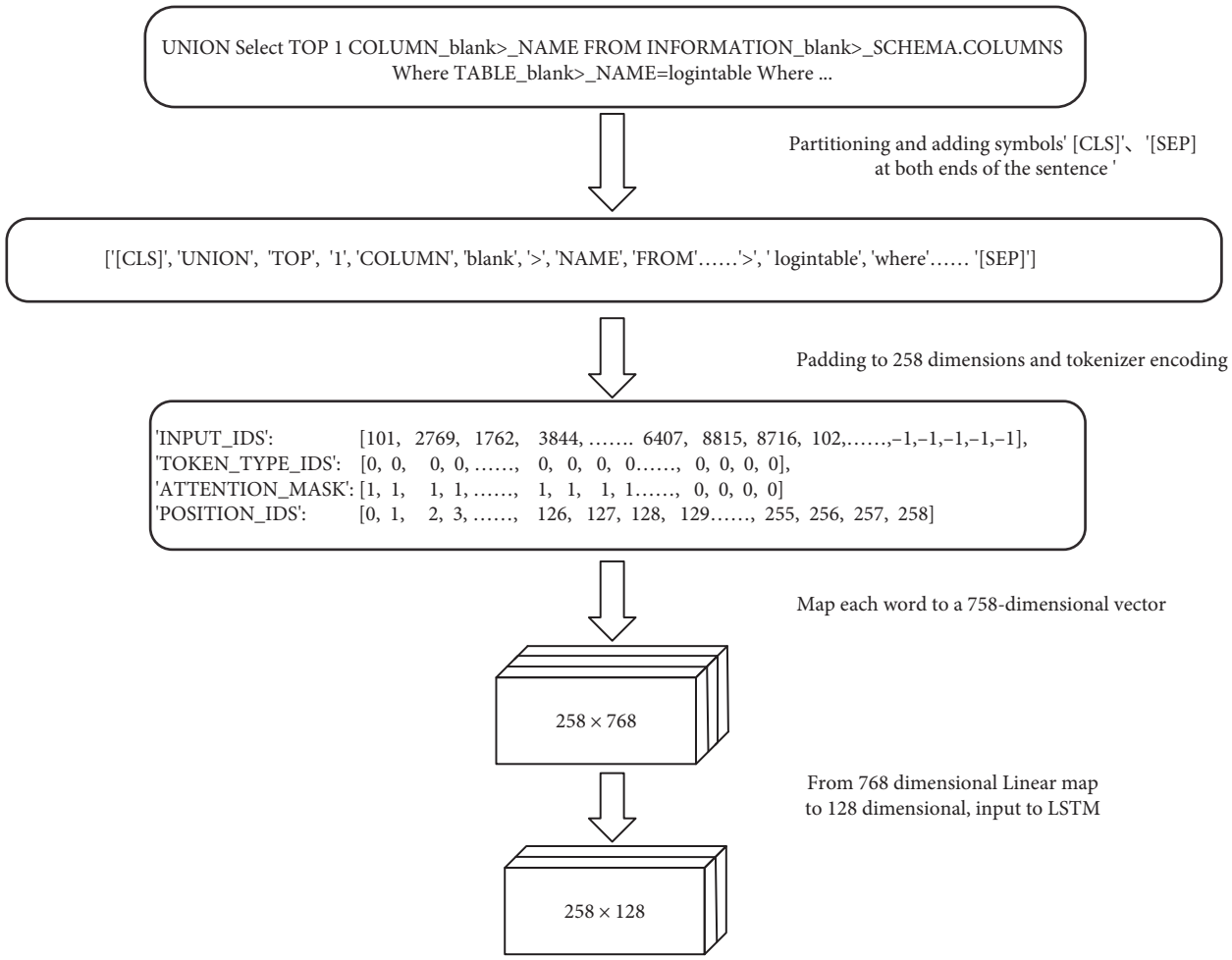

Figure 6: The process of tokenize.

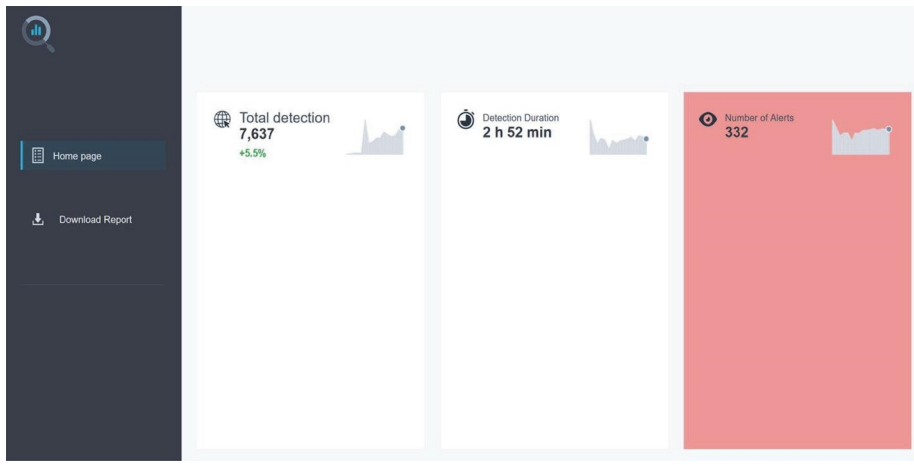

Figure 7: The user interface.

index for model evaluation, where precision rate (P) is the proportion of samples predicted to be malicious statements that are actually positively categorized [34]. The evaluation formula is shown below:

Precision is defined as Equation (8).

$$P = \frac{TP}{TP + FP}. \tag{8}$$

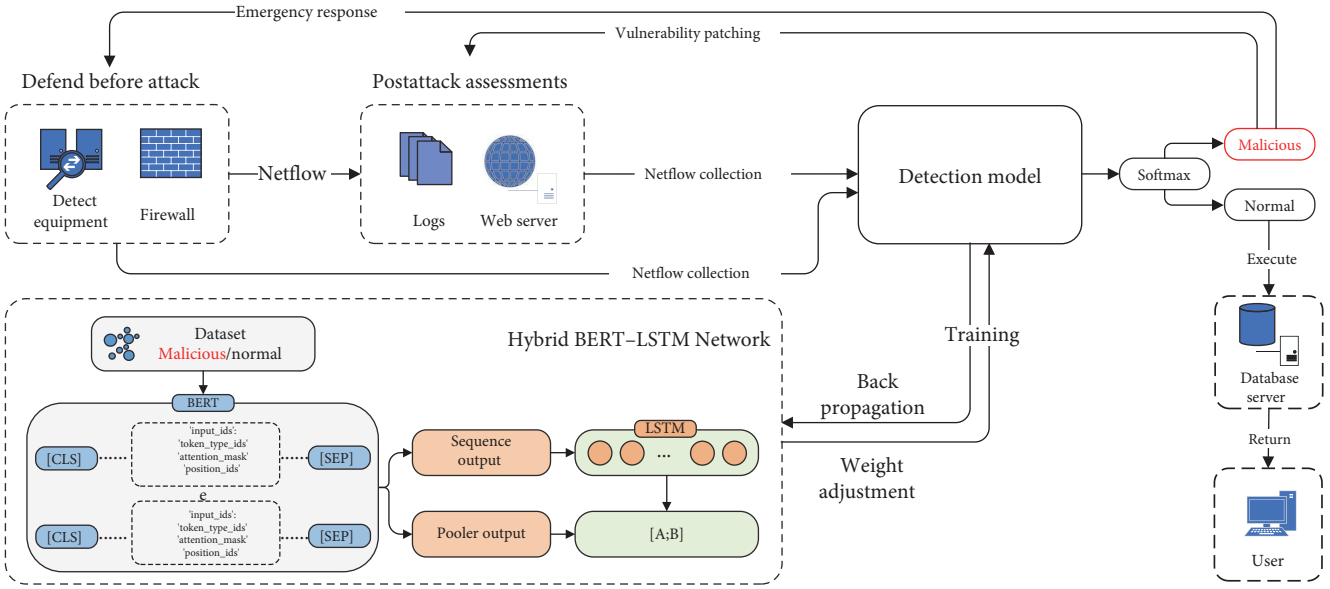

Figure 8: The deployment process of the model for detection.

Table 4: Confusion matrix.

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

Recall is defined as Equation (9).

$$R = \frac{TP}{TP + FN}. \tag{9}$$

Accuracy is defined as Equation (10).

$$Acc = \frac{TP}{TP + FN + TN + FP}. \tag{10}$$

F1 score is defined as Equation (11).

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

The confusion matrix [35] for each evaluation parameter in the above is shown in Table 4.

The learning rate is set to 0.01 during training, employing the stochastic gradient descent algorithm, and utilizing the classification cross-entropy as the loss function. After four epochs of data training, the accuracy rate reached 90%. After 30 epochs of training, the final recognition accuracy of the model converged to 97.3%; the precision rate reached 96.3%; the recall rate reached 96.2% and the F1 score reached 95.8%.

In the experiment, multihead self-attention was used, and dropout was used to remove neural units in the hidden layer according to a certain proportion. At the same time, the number of hidden layers, the number of units in the hidden layer, batch_size, and other parameters were continuously adjusted in multiple experiments, which effectively prevented overfitting. Despite a decrease in the validation accuracy rate, it still achieved a level of 93.3% by the 30th epoch.

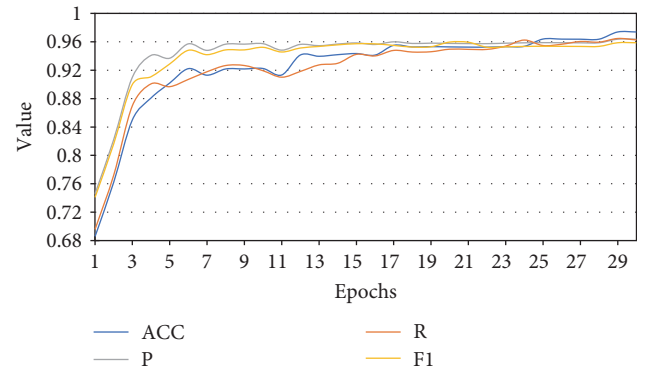

Figure 9: Accuracy, recall, and F1 score of Training.

The accuracy value, the precision value, the recall value, and the F1 score value of the training process are shown in Figure 9.

The accuracy values of the training process and the validation process are shown in Figure 10.

The loss of the training process and the verification process are shown in Figure 11.

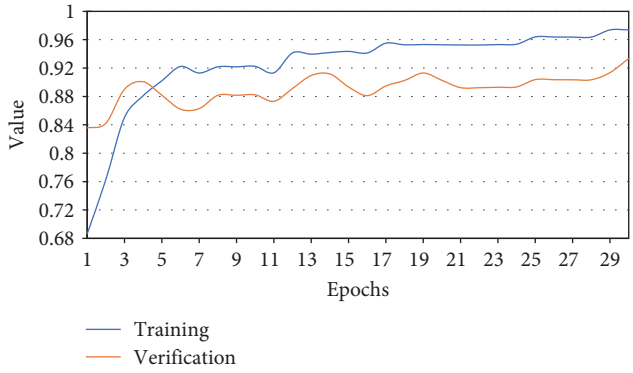

Figure 10: Accuracy of Training and Validation.

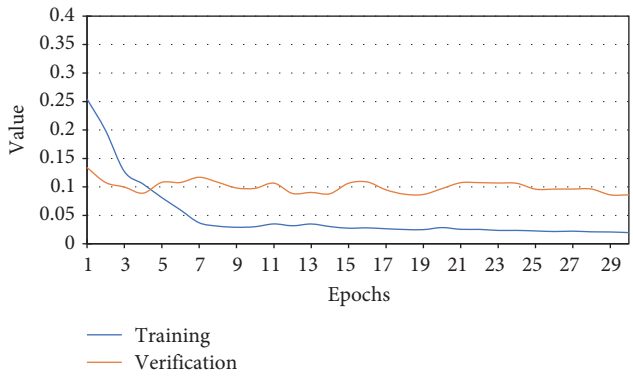

Figure 11: Loss of Training and Validation.

4.5.2. Experimental Validation of Model Effectiveness on Simulated Attack. To evaluate the model's generalization ability to classify SQL injection vulnerabilities, a testing platform was constructed. This platform encompasses prevalent types of injections, filtering rules were also added to test the detection of bypass injection:

(i) Error-based injections.
(ii) Boolean-based BLIND injections, time-based BLIND injections.
(iii) Stacked SQL injections.
(iv) Referer-based header injections, useragent-based header injections, cookie-based header injections.
(v) Bypassing blacklist filters stripping comments, stripping OR and AND, stripping SPACES and COMMENTS, stripping UNION and SELECT.

The simulated attack platform has been implemented on the server in order to intercept the data packets that are exchanged between the user and the server. Subsequently, the data contained within these packets is forwarded to the model for detection. Our approach includes incorporating different forms of payloads, such as typical SQL injection payloads, obfuscated payloads devised to evade filtering blacklists, benign statements embedding potentially harmful words, as well as payloads targeting Cross Site Scripting attacks (XSS), all intended for input and attack purposes. The experimental results are shown in Table 5.

It can be observed from the experimental results that the detection model can identify the majority of malicious payloads. Additionally, it exhibits certain generalization capabilities and can accurately predict some simple bypass attacks in character filtering. Due to the emphasis of the BERT model on the semantic relationship between words and context during the generation of embedded word vectors, it effectively reduces false positives in identifying benign statements that may include potentially harmful words as harmful.

However, a notable shortcoming of the model is its inability to accurately predict specially encoded payloads lacking malicious characteristics. Adding encoding to the dataset cannot effectively address this issue since there exist attack methods that can bypass WAF even after multiple encodings. In practical applications, it is advisable to configure the corresponding decoder. Before sending the netflow to the detection model, the decoder restores the payload to its original state, thereby reducing the possibility of false negatives.

In addition, it has been noted that the absence of XSS attack data within the dataset has impeded the model's ability to extract features associated with XSS attack payloads, particularly for non-SQL injection attacks. While it aligns with the experimental results, this circumstance poses a considerable risk in practical applications. Consequently, future research efforts will primarily concentrate on enhancing the model's capability to detect a wider range of threats.

5. Comparison and Discussion

In this section, three comparative experiments on SQL injection detection capabilities and computational efficiency were set up, using the SQL injection attack detection based on BERT and LSTM proposed in this paper with the malicious code classification and detection model based on the graph of tokens and Support Vector Machine (SVM) [14], the attack detection model based on 2D-Convolutional Neural Networks (2D-CNN) [15], and the detection injection attack model using Gate Recurrent Unit (GRU) [16] for comparison. At the same time, a comparative evaluation is conducted to assess the natural language feature extraction capabilities of the BERT model, the Word2vec model [36], as well as the One-hot [17] and TF-IDF [18] methods.

5.1. Advantages Demonstrated through Comparative Experiments

5.1.1. Comparison of Performance of Different Prediction Models on the Dataset. Deploy all detection models on the server in a consistent environment. Use the same dataset for training and validation processes, and analyze their accuracy (ACC), recall (R), and F1 scores on the same test set for comparison. The results of this evaluation are shown in Figure 12.

Regarding the accuracy (Acc) aspect, it can be observed that the detection model proposed in this paper along with the CNN and GRU classification detection models exhibit an accuracy rate exceeding 90%. Due to the presence of specific malicious attack statements in the test set intentionally

Table 5: Detection results of the model for different payloads.

Payload	Description	Detection results
id = 1 union select 1,2,group_concat (table_name), 3 from information_schema.tables where table_schema = database()	Union select injections	Malicious
id = 1′ and ascii(substr (database(),1,1)) > 114	Boolean-based BLIND injections	Malicious
id = 1′ and if (ascii(substr (database(),1,1)) > 114, sleep (3),null)	Time-based BLIND injectionst	Malicious
id = 1′ union select 1,2,"<?php @eval ($_GET['string'])?>" into outfile xxx.php	Webshell	Malicious
id = 1′^ (ascii (mid ((select (GROUP_CONCAT (TABLE_NAME))from (information_schema.TABLES)where (TABLE_SCHEMA = database())),1,1)) = 1) = '1′	Bypassing blacklist filters stripping SPACES	Malicious
Set @a = concat ('selec','t from xxx'); prepare h from @a; execute @a;	Stacked SQL injections	Malicious
Set @a = 0x73656c656374202a2066726f6d20787878;prepare h from @a;execute @a;	Stacked SQL injections and hexadecimal execution bypasses string filtering	Normal[1]
?id = −1 union select group_concat (`123`),2 from (select 123 union select * from flag)a	Bypassing blacklist filters stripping OR	Malicious
id = 1 unionunion selectselect 1,2,group_concat (table_name),3 from information_schema.tables where table_schema = database()	Double writing bypasses UNION and SELECT filtering	Malicious
M!T!@MzIGF.@uZ!CB.1c.GR@.h.dGV.4b.@Ww.!oM!! Sxjb@25jYX.Qo@M.Hg.@x.LH.V@.zZXI!oKSksM!S!.k. =	Insert special characters "!, @,." in based64 encoding to bypass string filtering	Normal[1]
Select and union and order by	Benign statements embedding potentially harmful words	Normal
<script >alert (document.cookie);</script >	XSS	Normal[1]

[1]The results indicate that the predicted result is opposite to the actual situation.

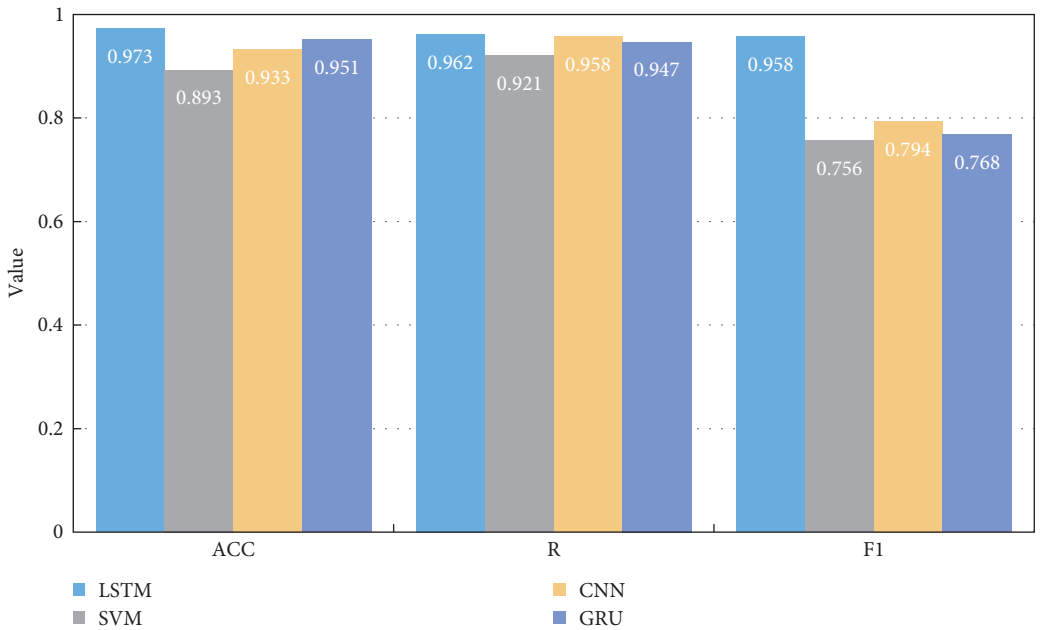

Figure 12: Comparison of different deep learning models.

designed to evade the WAF's detection rules, the SVM-based detection model experiences a decreased accuracy in recognizing these statements. Regarding the recall (R) aspect, the detection model proposed in this paper and the CNN-based detection model have more advantages and exhibit enhanced precision in recognizing the characteristics of malicious statements in SQL injection attacks. The detection model proposed in this paper has a higher F1 score because other detection models cannot perceive the semantic features and contextual relationships of the words in the utterance, which can easily cause misjudgment of the normal user input request containing sensitive features, so it leads to its lower

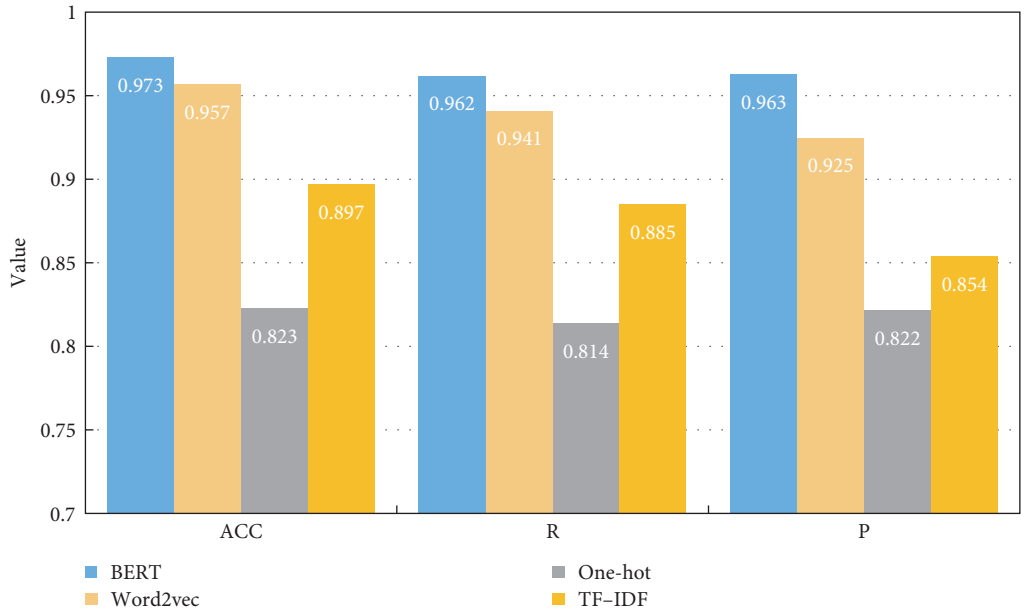

Figure 13: Comparison of different data processing techniques.

Table 6: The quantity of alerts generated by each distinct model.

Model	Quantity of alerts	False positive rate	False negative rate
BERT-LSTM	1,247	13%	17%
Word2Vec-LSTM	1,202	25%	31%
GRU	1,271	27%	29%
CNN	980	16%	38%
SVM	2,594	62%	25%

precision rate (*P*), resulting in a lower F1 score. Therefore, the approach presented in this paper exhibits superior efficacy in identifying SQL injection attacks when juxtaposed with GRU, SVM, and CNN models.

This experiment additionally examined the detection efficacy on the test sets of dynamically generated word vectors and sentence vectors using the BERT model, in contrast to the conventional Word2vec [36], One-hot [17], and TF-IDF [18] methods, which produce static word vectors. Both approaches utilize LSTM as the prediction model. The precision and recall rates of the ultimate model were assessed and compared. The outcomes of this comparison are presented in Figure 13.

Based on the comparison results, The BERT model generates vectors by combining keywords with their respective sentences. This, in conjunction with advanced word segmentation, leads to enhanced precision (P), recall (R), and F1 scores, surpassing those achieved by the other three static processingmethods. Hence, the BERT model yields more effective results in data preprocessing.

5.1.2. Comparison of Different Prediction Models in Netflow Detection. The four models are linked to the cyberspace situation awareness device that has been implemented in the laboratory. This device actively monitors the data flow traversing over 500 network assets, encompassing servers, personal computers, network equipment, etc. Whenever the detection model recognizes any malicious information, an alert is activated. Following one week of implementation, we carried out 1307 SQL attack experiments on multiple devices. The chart labeled as Table 6 presented beneath showcases the quantity of alerts generated by each distinct model.

All alarms that were generated were examined manually in order to determine the rates of false positive and false negative detection. Based on the findings, it is evident that the combined BERT and LSTM model proposed in this paper has the lowest rates of false positive and false negative detection. In contrast, other models encounter a greater number of words categorized as “unknown” outside the defined vocabulary, or encounter more normal inputs with similar characteristics to malicious sentences in the detection of extensive and intricate network traffic. As a result, these models exhibit poor generalization performance, rendering them unsuitable for practical applications.

5.1.3. Comparison of Computational Efficiency among Different Prediction Models. Furthermore, for the purpose of assessing the computational efficiency of the models, we meticulously documented the duration of training and test for all models with equal epoch numbers. Moreover, additional software and hardware environments are constructed with the objective of

Table 7: This is a table, T. Comparison of resources consumed by different model training and test.

Model	Environment	Time of training	Time of test
BERT-LSTM	Python3.5/GPU	204.3 s	51.7 s
Word2Vec-LSTM		198.6 s	32.2 s
GRU		247.1 s	49.9 s
CNN		279.5 s	76.2 s
SVM	Python3.5/CPU	143.8 s	18.2 s
BERT-LSTM	Python2.7/CPU	407.6 s	142.6 s
Word2Vec-LSTM		392.3 s	123.1 s
CNN		430.7 s	158.7 s
GRU		467.2 s	182.9 s

guaranteeing the genuineness of the outcomes. The outcomes of this comparison can be observed in Table 7.

In the present section of the comparison, the model combining BERT and LSTM proposed in this article does not achieve the best performance, but it still ranks among the top performers. Among these models, the SVM model, despite having the highest computational efficiency in the same experimental environment, experiences challenges due to its poor generalization ability. Consequently, its usage potentially compromises safety.

Considering all aforementioned comparative test results, it is believed that the method proposed in this article exhibits distinct advantages over other detection models in terms of detection accuracy, practical performance, and computational efficiency. Consequently, the method suggested in this article holds greater potential and promising application prospects in the field of security.

5.2. Outlooks. As individuals and the general public observe a growing emphasis on network security, diverse defense and detection systems are perpetually advancing. Ultimately, our strategies will be consolidated and incorporated into software or hardware to fortify the security of cyperspace. The future applications can be divided into two main categories: the first category involves examining the system log file in order to determine if it has been targeted by an attack, and identifying the specific area of vulnerability that requires strengthening. The second category consists of deploying a protective system between the main system and the user, similar to a firewall, to filter out data packets that contain harmful payloads, effectively safeguarding the main system from potential attacks.

In order to achieve the objective, it is imperative to improve the dataset by integrating the attributes of diverse attacks. This includes features like HTML tags used in XSS attacks ("<script >"), system commands employed in command injection ("sys", "ping"), etc. Simultaneously, modifications are made to the network structure so that it can directly predict the attack type. Finally, it is essential to incorporate functions capable of decoding and sanitizing the original data prior to inputting it into the model.

In the future, we intend to integrate our method with emerging technologies such as homomorphic encryption [37], federated learning, and zero-knowledge proof (ZKP) [38] to bolster its security, scalability, and detection capabilities.

6. Conclusion

Compared with the existing technology, since the BERT model dynamically generates word vectors and sentence vectors, and then combines the two during prediction, the vector values mapped to the same word in different sentences are different. For keywords containing malicious features in the sample, such as "union", when used as a word normally input by the user, the possibility of categorizing the sample as malicious is reduced. Hence, it possesses the benefits of a diminished occurrence of false-positive alerts and an elevated level of precision. The Wordpiece word segmentation algorithm is employed, enabling effective mitigation of out-of-vocabulary (OOV) issues, and low-frequency words can also be trained well. It has a high recall rate and low false negative rate for detecting malicious sentences with uncommon words and personal characteristics of the attacker.

In conclusion, this study presents a model for SQL injection attack statement detection, integrating BERT and LSTM methodologies. The model showcases superior precision rates and is notably adept at identifying malevolent statements characterized by rare lexicon and unique attacker signatures. Future research initiatives will aim to refine this model, enhancing its capability to discern and elevate the performance of injection statement recognition. Moreover, efforts will be directed towards expanding its application within the broader domain of network security research.

Data Availability

Httpparams dataset open source database were analyzed in this study. This data can be found here: https://github.com/Morzeux/HttpParamsDataset. The source code of the BERT model from Google Github open source code can be found here: https://github.com/google-research/bert. The processed data required to reproduce the experiment is available on request from the corresponding author as the data also forms part of an ongoing study.

Additional Points

Limitations. The experimental results clearly illustrate the advantages of this approach in extracting SQL injection statements and identifying SQL injections. However, our research identifies several limitations that necessitate further

improvement. The model demonstrates insufficient accuracy in detecting payloads that have undergone multiple encodings to bypass censorship. Moreover, attackers often employ multiple attack techniques concurrently. However, this method proves ineffective in identifying other forms of injection attacks, such as XSS and system command injections. Lastly, it is crucial to make additional adjustments to the dataset and network hyperparameters to enhance both the model's detection accuracy and computational efficiency.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] A. Salam, F. Ullah, F. Amin, and M. Abrar, "Deep Learning techniques for web-based attack detection in industry 5.0: a novel approach," *Technologies*, vol. 11, no. 4, Article ID 107, 2023.

[2] OWASP, "OWASP top ten," 2021, https://owasp.org/Top10/.

[3] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: a systematic literature review," *Journal of Cybersecurity and Privacy*, vol. 2, no. 4, pp. 764–777, 2022.

[4] A. Katole, S. Swati, and T. Vilas, "Detection of SQL injection attacks by removing the parameter values of SQL query," in *Proceedings of 2018 2nd IEEE International conference on inventive systems and control (ICISC)*, pp. 736–741, IEEE, Coimbatore, India, 2018.

[5] D. Chen, Q. Yan, C. Wu, and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *Journal of Physics: Conference Series*, vol. 1757, no. 1, Article ID 012055, 2021.

[6] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "WAF-A-MoLE: evading web application firewalls through adversarial machine learning," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 1745–1752, ACM, Brno, Czech Republic, 2020.

[7] S. Manmadhan and M. Thankappan, "A method of detecting SQL injection attack to secure web applications," *International Journal of Distributed and Parallel Systems*, vol. 3, no. 6, pp. 1–8, 2012.

[8] A. Rai, M. M. I. Miraz, D. Das, H. Kaur, and Swati, "SQL injection: classification and prevention," in *Proceedings of 2021 2nd IEEE International Conference on Intelligent and Management (ICIEM)*, pp. 367–372, IEEE, London, United Kingdom, 2021.

[9] M. A. Azman, M. F. Marhusin, and R. Sulaiman, "Machine learning based technique to detect SQL injection attack," *Journal of Computer Science*, vol. 17, no. 3, pp. 296–303, 2021.

[10] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, "A static analysis framework for detecting SQL injection vulnerabilities," in *Proceedings of the 31st Annual International Computer Software and Applications Conference*, vol. 1, pp. 87–97, IEEE, Beijing, China, 2007.

[11] A. Khalid and M. M. F. Yousif, "Dynamic analysis tool for detecting SQL injection," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, no. 2, pp. 224–232, 2016.

[12] J. Prakash and G. Saravanan, "SQLID: SQL injection detection based on static and dynamic analysis," Transylvanian Review, no. 1, 2016.

[13] M. Nasereddin, A. ALKhamaisehat, M. Qasaimeh, and R. Al-Qassas, "A systematic review of detection and prevention techniques of SQL injection attacks," *Information Security Journal: A Global Perspective*, vol. 32, no. 4, pp. 252–265, 2023.

[14] D. Kar, S. Panigrahi, and S. Sundararajan, "SQLiGoT: detecting SQL injection attacks using graph of tokens and SVM," *Computers & Security*, vol. 60, pp. 206–225, 2016.

[15] F. R. Abdulhamza and R. J. S. Al-Janabi, "SQL injection detection using 2D-convolutional neural networks (2D-CNN)," in *Proceedings of 2022 International Conference on Data Science and Intelligent Computing (ICDSIC)*, pp. 212–217, IEEE, Kerbala, Iraq, 2022.

[16] C. Zhao, S. Si, T. Tu, Y. Shi, and S. Qin, "Deep-learning based injection attacks detection method for HTTP," *Mathematics*, vol. 10, no. 16, Article ID 2914, 2022.

[17] K. R. Jothi, S. B. Balaji, N. Pandey, P. Beriwal, and A. Amarajan, "An efficient SQL injection detection system using deep learning," in *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pp. 442–445, IEEE, Dubai, United Arab Emirates, 2021.

[18] W. Zhang, Y. Li, X. Li et al., "Deep neural network-based SQL injection detection method," *Security and Communication Networks*, vol. 2022, Article ID 4836289, 9 pages, 2022.

[19] M. Zivkovic, M. Tair, V. K, N. Bacanin, Š. Hubálovský, and P. Trojovský, "Novel hybrid firefly algorithm: an application to enhance XGBoost tuning for intrusion detection classification," *PeerJ Computer Science*, vol. 8, Article ID e956, 2022.

[20] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego-DelCastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, "SQL injection attack detection in network flow data," *Computers & Security*, vol. 127, Article ID 103093, 2023.

[21] S. Shah, A. Munir, A. Waheed et al., "Enhancing security and efficiency in underwater wireless sensor networks: a lightweight key management framework," *Symmetry*, vol. 15, no. 8, Article ID 1484, 2023.

[22] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp. 4171–4186, Association for Computational Linguistics, Minneapolis, Minnesota, USA, 2019.

[23] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: a review of BERT-based approaches," *Artificial Intelligence Review*, vol. 54, no. 8, pp. 5789–5829, 2021.

[24] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo, "Evaluating word embedding models: methods and experimental results," *APSIPA Transactions on Signal and Information Processing*, vol. 8, no. 1, Article ID e19, 2019.

[25] M. Farouk, "Measuring text similarity based on structure and word embedding," *Cognitive Systems Research*, vol. 63, pp. 1–10, 2020.

[26] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," Eprint Arxiv, 1409.2329, 2014.

[27] E. Borandag, "Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques," *Applied Sciences*, vol. 13, no. 3, Article ID 1639, 2023.

[28] C. Wang, J. Feijun, and Y. Hongxia, "A hybrid framework for text modeling with convolutional RNN," in *Proceedings of 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data MIning*, pp. 2061–2069, ACM, Halifax, Canada, 2017.

[29] A. Ajitha, M. Goel, M. Assudani, S. Radhika, and S. Goel, "Design and development of residential sector load prediction model during COVID-19 pandemic using LSTM based RNN," *Electric Power Systems Research*, vol. 212, Article ID 108635, 2022.

[30] B. M. Ehsan, N. Shahla, A. Moloud, C. Erik, and A. U. Rajendra, "ABCDM: an attention-based bidirectional CNN-RNN deep model for sentiment analysis," *Future Generation Computer Systems*, vol. 115, pp. 279–294, 2021.

[31] G. Saon, Z. Tueske, D. Bolanos, and B. Kingsbury, "Advancing RNN transducer technology for speech recognition," in *Proceedings of ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5654–5658, IEEE, Toronto, ON, Canada, 2021.

[32] S. Hochreiter and J. Jürgen, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] C. Staudemeyer and M. Eric, "Understanding LSTM–a tutorial into long short-term memory recurrent neural networks," Eprint Arxiv.1909.09586, 2019.

[34] R. Yacouby and D. Axman, "Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models," in *Proceedings of Association for Computational Linguistics First Workshop on Evaluation and Comparison of NLP Systems*, pp. 79–91, Association for Computational Linguistics, Red Hook, NY, United States, 2020.

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

[36] P. C. Wen, C. W. He, W. Xiong, and J. H. Liu, "SQL injection detection technology based on BiLSTM-attention," in *Proceedings of 2021 4th IEEE International Conference on Robotics, Control and Automation Engineering (RCAE)*, pp. 165–170, IEEE, Wuhan, China, 2021.

[37] A. Salam, M. Abrar, F. Ullah, I. A. Khan, F. Amin, and G. S. Choi, "Efficient data collaboration using multi-party privacy preserving machine learning framework," *IEEE Access*, vol. 11, pp. 138151–138164, 2023.

[38] A. Salam, M. Abrar, F. Amin et al., "Securing smart manufacturing by integrating anomaly detection with zero-knowledge proofs," *IEEE Access*, vol. 12, pp. 36346–36360, 2024.