applied sciences

MDPI

Article

Deep Learning-Based Detection Technology for SQL Injection Research and Implementation

Hao Sun [1], Yuejin Du [2] and Qi Li [1,*]

[1] Academy of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China; sh666@bupt.edu.cn
[2] Beijing Qihoo Technology Co., Ltd., Beijing 100015, China
* Correspondence: liqi2001@bupt.edu.cn

Abstract: Amid the incessant evolution of the Internet, an array of cybersecurity threats has surged at an unprecedented rate. A notable antagonist within this plethora of attacks is the SQL injection assault, a prevalent form of Internet attack that poses a significant threat to web applications. These attacks are characterized by their extensive variety, rapid mutation, covert nature, and the substantial damage they can inflict. Existing SQL injection detection methods, such as static and dynamic detection and command randomization, are principally rule-based and suffer from low accuracy, high false positive (FP) rates, and false negative (FN) rates. Contemporary machine learning research on SQL injection attack (SQLIA) detection primarily focuses on feature extraction. The effectiveness of detection is heavily reliant on the precision of feature extraction, leading to a deficiency in tackling more intricate SQLIA. To address these challenges, we propose a novel SQLIA detection approach harnessing the power of an enhanced TextCNN and LSTM. This method begins by vectorizing the samples in the corpus and then leverages an improved TextCNN to extract local features. It then employs a Bidirectional LSTM (Bi-LSTM) network to decipher the sequence information inherent in the samples. Given LSTM's modest effectiveness for relatively long sequences, we further integrate an attention mechanism, reducing the distance between any two words in the sequence to one, thereby enhancing the model's effectiveness. Moreover, pre-trained word vector features acquired via BERT for transfer learning are incorporated into the feature section. Comparative experimental results affirm the superiority of our deep learning-based SQLIA detection approach, as it effectively elevates the SQLIA recognition rate while reducing both FP and FN rates.

Keywords: deep learning; SQL injection detection; TextCNN; LSTM

Citation: Sun, H.; Du, Y.; Li, Q. Deep Learning-Based Detection Technology for SQL Injection Research and Implementation. *Appl. Sci.* **2023**, *13*, 9466. https://doi.org/10.3390/app13169466

Academic Editor: Krzysztof Koszela

Received: 20 June 2023
Revised: 11 August 2023
Accepted: 16 August 2023
Published: 21 August 2023

1. Introduction

SQL injection attacks pose a significant threat to web applications, constituting one of their gravest security vulnerabilities. These attacks present unique features such as a multitude of variants, rapid evolution, covert attack vectors, and potentially severe consequences. Traditional SQL injection detection methodologies—including static and dynamic analysis as well as instruction randomization—predominantly rely on rule-based detection. However, these approaches often suffer from low accuracy, high false positive rates, and an inability to adequately handle the escalating complexity of contemporary SQL injection attacks. Hence, the exploration and development of an effective detection strategy for SQL injection attacks remain a considerable challenge and of paramount importance.

At present, feature extraction serves as the cornerstone for the majority of SQL injection detection methods. Most of these employ shallow machine learning techniques to identify SQL injection attacks, typically by extracting distinct elements from the SQL injection statements. To this end, researchers frequently extract explicit elements from these attack statements and employ simple machine learning methods for detection. For instance, Buchanan et al. leveraged a combinatorial support vector machine [1] as a

classifier for SQL injection attack detection, integrating HMM and similarity distance algorithms. Krishnan et al. employed a convolutional neural network [2] for the same purpose. Farooq et al. utilized an ensemble of four machine learning algorithms, namely the Gradient Boosting Machine (GBM), Adaptive Boosting Machine (AdaBoost), Extended Gradient Boosting Machine (XGBM), and Light Gradient Boosting Machine (LGBM) [3], which resulted in relatively high accuracy. Furthermore, Adebiyi et al. concluded that the Decision Tree Algorithm [4] emerged as the best classifier for SQLIA detection and prevention due to its highest sensitivity and specificity. While these methods can effectively detect a portion of SQL injection attacks with precise feature extraction, they demand a substantial allocation of human resources. Additionally, the accuracy of feature extraction significantly impacts detection results.

As deep learning advances, researchers have attempted to utilize it for detecting SQLIA, but this approach often leads to overfitting problems. To address this issue, some researchers have proposed data set expansion to mitigate it. For example, Zhang et al. proposed a case clone extreme learning machine to expand the dataset [5]. Yet, this strategy remains reliant on the precision of selected features. Therefore, it is of the utmost importance to explore a scheme based on deep learning that can effectively detect SQLIA.

In this paper, we propose a deep learning-based method for SQL injection attack detection that integrates TextCNN, LSTM, and the ATTENTION mechanism to construct a deep network. Initially, this method proposes a feature vectorization method for SQL injection samples based on TF-IDF and Word2Vec. Coupled with the word embedding method, each word or phrase is mapped to a vector in the low-dimensional real number field. The similarity performance of the vector symbolizes the semantic similarity between words. Subsequently, upon vectorizing the samples in the corpus, we employ an enhanced TextCNN to extract the local features from the samples and use the Bi-LSTM network to extract the sequence information. Finally, to address the limited efficacy of LSTM on long sequence data, the ATTENTION mechanism is applied to reduce the distance between any two words in the sequence to one. This improves the effectiveness of the model.

The rest of the paper is organized as follows: Section 2 describes the background of SQL injection detection and the related work. Section 3 introduces the framework of our SQL injection detection. Section 4 details the deep learning model based on improved TextCNN and Bi-LSTM. Section 5 presents the experiment and evaluates the results, and Section 6 concludes our work.

2. Related Studies

Due to the variety of SQL injection attacks and their great harm, researchers have proposed many SQLIA detection methods. The conventional methods for detecting SQL injection can be categorized into three fundamental approaches: static detection, dynamic detection, and dynamic and static combination [6].

Static detection. Static detection methodologies, such as white-box testing, involve the evaluation of potential SQL injection vulnerabilities via static source code analysis [7] without necessitating program execution. Gould et al. developed a code analysis tool, the JDBC Checker [8]. While this tool can identify specific types of SQL injection attacks, it lacks preventive measures. Wassermann et al. extended white-box testing [9] to detect tautology attacks. Notably, static detection must check the source code of the web application and requires specific professional knowledge. The relatively high false positive rate results in considerable consumption of time and resources, thereby limiting its practicality.

Dynamic detection. Dynamic detection, unlike static detection, identifies SQL injection vulnerabilities in web applications without requiring access to the application source code. This process involves dynamic penetration testing or real-time model generation. Yi et al. proposed a dynamic taint analysis method [10] and devised an embedded analysis model within the web application. They utilized SQL lexical and syntax analysis techniques to parse SQL statements into SQL syntax trees using taint analysis. SQL injection attacks can be identified by examining the tainted state. Appiah et al. developed a signature-based

framework for SQL injection attack detection, improving the pattern-matching [11] method. This approach integrates fingerprint recognition and pattern matching to differentiate between legitimate SQL queries and malicious ones. Although this method has a low false positive rate, it struggles to identify concealed and newly emerging SQL injection attacks. Bisht et al. introduced CANDID, which uses the expected parse tree [12] in comparison with the runtime parse tree to detect SQLIA. Queries flagged as SQLIA result from a mismatch between the trees. However, the runtime parse tree comparison is resource-intensive and only capable of detecting certain types of attacks. Consequently, most dynamic detection methods suffer from drawbacks such as complex feature extraction, slow detection speed, and high system resource consumption.

Dynamic and static combination. Combining static and dynamic detection leverages the advantages of both approaches by integrating static source code analysis with dynamic real-time monitoring. In the static analysis phase, a secure SQL statement model is established based on the source code of the web application. In the dynamic analysis phase, the SQL statements are intercepted before they are submitted to the database and compared against the model obtained from the static analysis phase. Discrepancies are flagged as SQL injection attacks, and the query execution is terminated and recorded. AMNESIA [13], as proposed by William, implements this strategy. It performs static analysis of the web application to discern the locations where SQL queries are sent to the database and generates a normal SQL query model. During the dynamic phase, it monitors dynamically created queries to ensure they conform to the static model. Queries that do not match the model are identified as SQLIA and blocked, and the relevant information is reported to developers and administrators. Xiao et al. proposed a method based on URL-SQL mapping [14], which analyzes user behavior and SQL execution responses to detect and defend against SQL injection. This method introduces minimal additional cost to the web application and can still detect and prevent SQL injection even under high volumes of attacks. However, there are uncertainties in system SQL execution, and the extraction of invariants (normal states of the web application) is not comprehensive, leading to a significant number of false positives and false negatives.

The above methods can be well expanded in the traditional string matching at that time. Now attackers use more complex tools to launch automatic injection attacks. These methods cannot detect various SQL injection attacks well.

The problem of SQLIA detection is a typical classification problem, so many researchers started to focus on the utilization of machine learning technology to solve the problem of SQL injection attack detection. For instance, Kim et al. employed n-gram features extracted from SQLIA code [15] to train an SVM classifier and achieve better detection performance. Nonetheless, the accuracy of this method is contingent upon various patterns, which restricts its utility. Lei et al. designed a SQL injection scanning tool based on crawler and machine learning methods [16], which can automatically detect potential SQL injection security vulnerabilities in websites. However, due to the limitations of scannable datasets, it only recognizes a small assortment of attack types. Komiya et al. extracted features using blank-separated and token methods [17]. They proposed a method for adaptive classification of malicious web code using machine learning methods, including NB, SVM, and KNN. While the experiments showed promising results, the dearth of test samples and training samples could lead to overfitting, and not all types of SQLIA can be effectively detected. Moreover, manual feature construction proves inefficient due to its high cost, excessive reliance on expert knowledge, difficulty in ensuring feature efficacy, substantial feature redundancy, escalated training costs, and a lack of ability to capture complex patterns and scalability.

Traditional machine learning methods usually require manual feature engineering, while deep learning models [18–20] can automatically learn hierarchical feature representations of data. Additionally, deep learning is scalable and usually works well for large-scale datasets, with its performance continually improving as the dataset expands. Zhuo et al. proposed a new detection method based on long short-term memory and

an abstract syntax tree [21], which can detect SQLIA from the raw query strings, even in SQL detection bypass scenarios. Inspired by the layered architecture design of LSTM [22], Dawadi et al. developed a layered architecture model for various web attack detection, achieving an SQLIA detection accuracy rate of 89.34%. Gandhi et al. proposed a hybrid detection method based on CNN-Bi-LSTM [23], using a convolutional neural network for feature extraction and Bi-LSTM for learning long-term dependencies of data. This method achieved an experimental accuracy rate of 98%. Li et al. put forth an SQLIA detection method based on adaptive deep forests [24], which effectively solved the problem that the original features of deep forests degenerate with the increase in layers, and the experimental results obtained an accuracy rate of 98.75%. Alarfaj et al. utilized a probabilistic neural network [25] to detect SQL injection attacks and employed the BAT algorithm to optimize parameter selection. This method can produce more accurate results than the multi-layer perceptron network and is relatively insensitive to outliers; the experimental results obtained an accuracy rate of 99.19%. However, methods based on deep learning algorithms still face numerous challenges. Firstly, since there are few publicly available SQL injection sample datasets, the lack of enough samples will lead to an overfitting problem during the training process. Secondly, the method of converting SQL samples into feature vectors also needs to be further studied.

In general, traditional SQL injection attack detection methods are not only difficult to deal with large-scale SQL injection detection due to their low detection speed and cost [10–14], but also because the detection effect of SQLIA on websites with different protection measures is often uneven. More importantly, the application of shallow learning [1–4,15–17] in the field of SQL injection detection has problems with poor generalization performance and poor classification results. As deep learning technologies continue to mature and find applications across diverse fields, their effective and accurate deployment in SQLIA detection is a topic of significant interest.

3. Framework Overview

The schematic representation of the system's overall architecture is illustrated in Figure 1. The overall architecture of our proposed system includes two stages: offline training and online testing. During the offline training phase, substantial samples crawled by crawlers and generated by SQL injection tools are labeled as positive and negative samples. After that, through the sample preprocessing module, structured data can be obtained, including encoding processing, generalization, word segmentation, etc. At last, the model is trained using the preprocessed structured data to obtain a classifier after vectorization. In the online testing phase, the data to be tested is preprocessed in the same way as in the offline training phase. Subsequently, the preprocessed data are fed into the trained classifier to ascertain the presence of an SQL injection attack.

The data preprocessing module consists of data cleaning, decoding, and restoration of various encoded data, generalization, excessive rewriting, and tokenization. The generalization process mainly includes converting all letters to lowercase, representing strings in date format with the term "date," representing integers with the digit 0, and representing hexadecimal data starting with "0x" with the field "0x1234". In response to the overriding bypass method, it mainly filters long strings containing keywords and uses keywords instead of long strings. Tokenization is performed based on blank separation, with the preservation of certain sensitive special characters.

The sample vectorization module is responsible for converting preprocessed samples into numerical matrices that can be directly input into the algorithm module. This is achieved through various steps such as TF-IDF processing and Word2Vec processing, which extract both term frequency and semantic information from the preprocessed samples. The resulting sample matrix is then used for subsequent model training.

The algorithmic model module is primarily responsible for implementing the overall structure of the deep learning models based on the improved TextCNN and Bi-LSTM. It

involves training the models to obtain the detector, and the detailed model architecture will be elaborated in Section 4.

The online testing module is primarily responsible for using the trained classifier to detect newly appearing SQL queries and output the detection results.

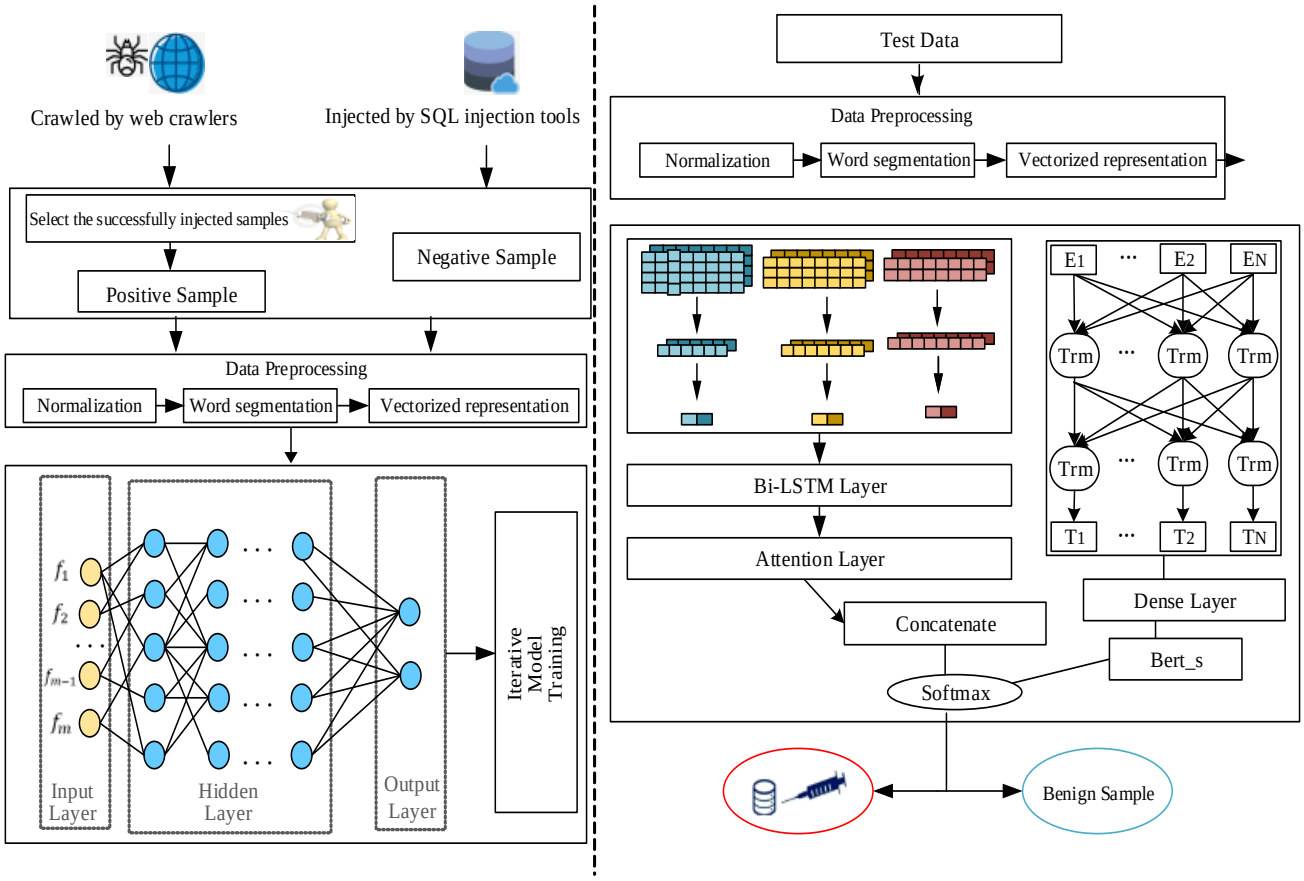

Figure 1. The overall architecture of the SQL injection attack detection system.

4. Deep Learning Model Based on Improved TextCNN and Bi-LSTM

4.1. Description of the Model of Improved TextCNN and Bi-LSTM (ITCBL)

Traditional machine learning methods usually require manual feature engineering, and many features are extracted by researchers, such as "percentage of uppercase characters," "the existence of typical SQL injection keywords," etc. However, these features themselves make it difficult to effectively detect different types and variants of SQL injection statements. To this end, we propose a SQLIA method based on deep learning. In the network structure, the improved TextCNN algorithm is combined with the Bi-LSTM and ATTENTION mechanisms, supplemented by the word vector representation extracted by BERT, so that the model can learn finer features and perform better in detecting various variants of SQL injection attacks.

The deep model network structure proposed in this paper for detecting SQLIA is shown in Figure 2. First, the SQL injection samples are input into the TF-IDF and Word2Vec fusion algorithms to obtain their vectorized representations. Then, the pre-trained word vectors are input to the improved TextCNN network to learn local feature representation, extract local complex advanced features, and form n features for each sample after the pooling operation. These features serve as input for the Bi-LSTM network to learn sequence timing information and extract sequential features. The ATTENTION mechanism is then employed to strengthen the weight of important features, weaken the influence of noise, and obtain the feature vector s. Additionally, the BERT algorithm is utilized for migration

learning to obtain the Transformer's bidirectional encoder representation of each word in the sample. These representations are flattened to obtain the feature vector bert_s, which is subsequently combined with s. Finally, a fully connected layer is introduced, followed by the application of SoftMax for probability normalization, culminating in the attainment of the ultimate category classification.

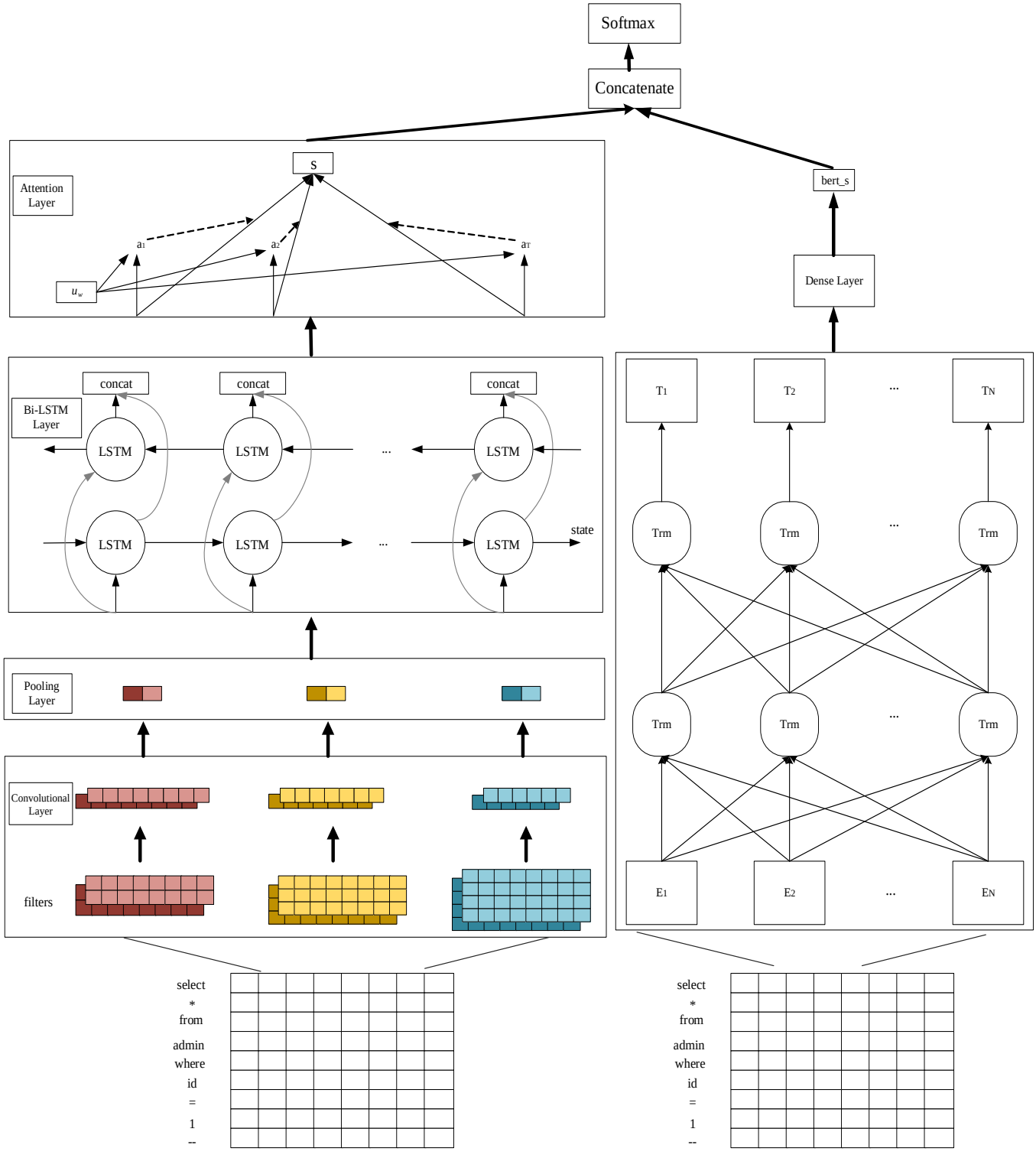

Figure 2. Deep learning model based on improved TextCNN and LSTM algorithms.

4.2. Improved TextCNN Algorithm

TextCNN is an algorithm that applies convolutional neural networks to sentence classification, as proposed by Yoon Kim in 2014 [26]. The convolutional neural network has always been a sharp tool in the direction of computer vision. Yoon Kim improved on CNN and proposed a text classification model called TextCNN. Because of its simple network structure, less calculation, fast training speed, and excellent effect, it was widely used by researchers. Since SQL is essentially a formal language that can be understood as a meaningful language sequence, the SQL injection attack samples studied in this paper are also text. Consequently, this paper makes appropriate improvements to the TextCNN algorithm to make it more applicable to the SQL injection attack detection task.

The basic TextCNN mainly includes basic neural network components such as the embedding layer, convolutional layer, and pooling layer, as follows:

- Embedding layer: In the text field, it is mostly used to convert words appearing in the text into fixed-size vectors. It is a flexible layer that can be used in various ways. It can be used only to learn word vectors used in another model it can also be used as part of a deep learning model to learn both word vectors and the model itself or it can be used to load pre-trained word vectors.
- Convolutional layer: It is composed of convolutional units, the parameters of which are obtained through backward propagation learning. The convolutional layer in the SQL sample is mainly for feature extraction and feature mapping of the vectorized SQL text and iterates from low-level features. Extract more complex, high-level features.
- Pooling layer: It mainly performs down-sampling operations on the feature map extracted by the convolutional layer and divides the space into multiple regions according to the size of the convolution kernel. And use a different pooling method to calculate a pooled value from these areas to replace the commonly used average pooling and maximum pooling.

To enhance its applicability in the realm of SQLIA detection, we improved the pooling layer of TextCNN. The subsequent section outlines the comprehensive processing procedure of the enhanced TextCNN model discussed in this paper, specifically tailored for the detection and handling of SQL injection. This allows for effective handling of sample processing tasks. The architecture of TextCNN for processing SQL injection text is shown in Figure 3. The embedding layer first embeds the SQL injection text into a low-dimensional vector space, and the convolution layer uses multiple convolution kernels of different sizes to perform convolution operations on the embedded word vectors. Next, the pooling layer converts the result of the convolution operation into a long feature vector and adds a regular term. The specific steps and principles are as follows: The mathematical symbols appearing in the following text are represented as shown in Table 1.

$x_i \in R^k$ is the k-dimensional word vector corresponding to the i-th word in the sample. Due to the use of dual channels, each word corresponds to two k-dimensional word vectors. Therefore, a SQL statement with a length of n (if it is shorter than n, it needs to be filled with zero vectors, and if the length is greater than n, it needs to be truncated) is expressed as Formula (1):

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{1}$$

Among them, the symbol $\oplus$ is the splicing operation, $x_{i:i+j}$ represents the splicing operation on x_i, x_{i+1}, …, x_{i+j}, and the convolution kernel, $W \in R^{h*k}$, is applied to all words with a window size of h to generate feature c_i. The feature generation formula corresponding to the convolution operation on the window $x_{i:i+h-1}$ is shown in (2):

$$c_i = f(W \cdot x_{i:i+h-1} + b) \tag{2}$$

$b \in R$ is a paranoid term, f is a nonlinear transformation function, such as tan h, and the step size used when applying the above convolution is 1, so the corresponding word

window set is $\{x_{1:h}, x_{2:h+1i}, \ldots, x_{n-h+1:n}\}$. The feature map generated after applying the convolution operation is shown in Formula (3):

$$c = [c_1, c_2, \ldots, c_{n-h+1}] \tag{3}$$

Therefore, there is $c \in R^{n-h+1}$, and the basic TextCNN structure is to apply the maximum pooling operation on this basis. The current convolution kernel will generate a scalar $\bar{c} = \max\{c\}$, so the $\bar{c}$ generated by each convolution after the above operations represents the most significant feature proposed by the convolution. In practical applications, the window h can take different values and take l convolutions for each type of window h. Therefore, in the end, the feature corresponding to each SQL sample is expressed as a vector of $z = [\bar{c}_1, \ldots, \bar{c}_m]$, and the dimension of the vector is $m = l \times h$.

When applied to SQL injection detection, since the maximum pooling makes it easy to lose some important information, the average pooling will weaken the required information, so this paper improves the pooling operation. Use k-max pooling to extract the top K eigenvalues with the highest score among all eigenvalues while preserving the order of the eigenvalues. For 2-max pooling, the convolution kernel will generate a vector containing the top 2 eigenvalues $c' = \text{top2}\{c\}$, the final generated SQL sample pair is expressed as a vector of $z' = [c_1', \ldots, c_m']$, and the vector dimension is $m = 2 \times l \times h$. In addition, k-max pooling can better retain its cumulative characteristics for recurring important features and preserve the size order of features during operation. Therefore, after this operation of pooling, the strength information of the feature is still preserved, and the locally advanced features in the sample can be well extracted.

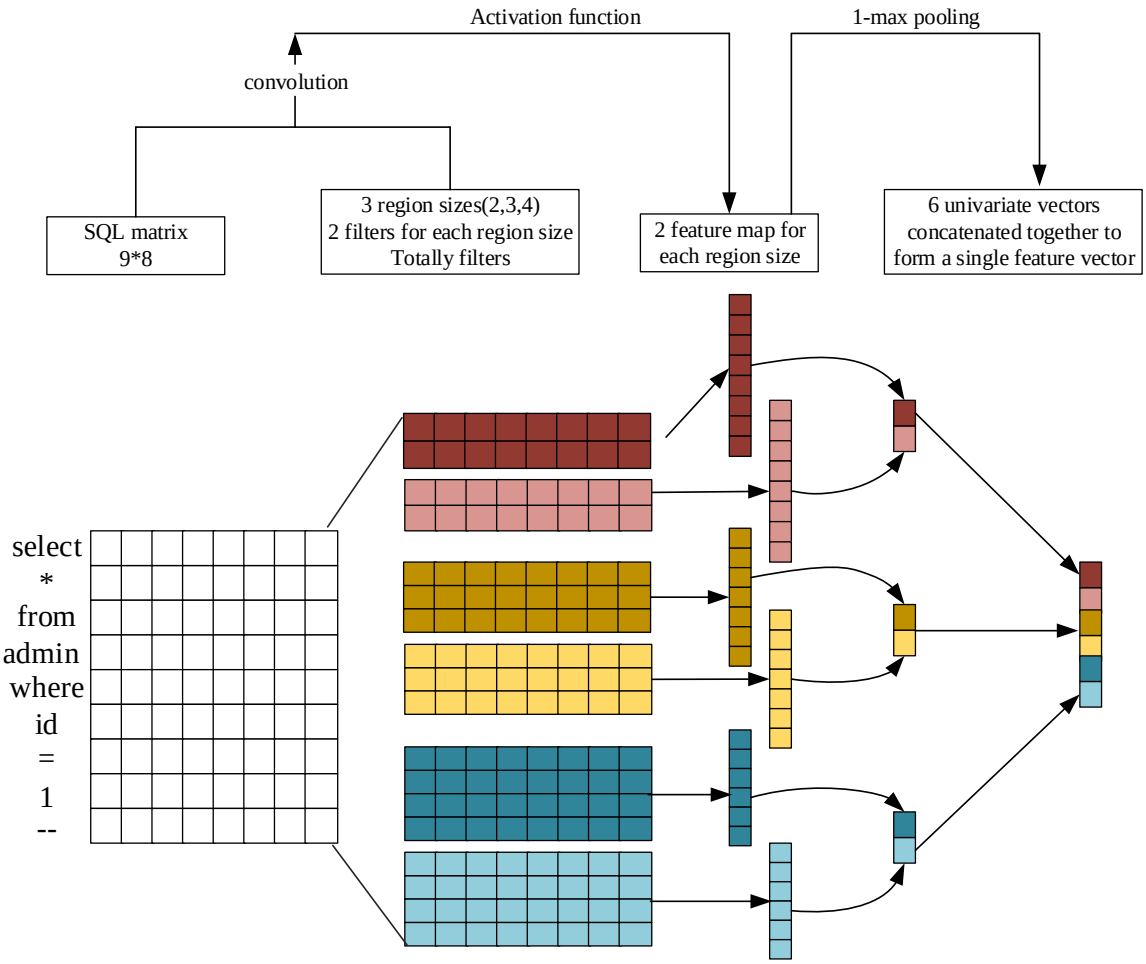

Figure 3. Architecture diagram of TextCNN processing SQL injection text.

Table 1. Notation and significance.

Notation	Significance
$x_i \in R^k$	The k-dimensional word vector corresponding to the i-th word in the sample, which belongs to the k-dimensional real space
$x_{i:i+j}$	The concatenation operation from the i-th word vector to the $(i+j)$-th word vector
$W \in R^{h*k}$	The convolutional kernel, which belongs to the $(h*k)$-dimensional real space
h	The window size when generating features
l	The number of convolutional operations
m	The dimension of the feature vector for each SQL sample
z	The feature vector generated for each SQL sample
c_i	The features generated by the convolution operation over the window $x_{i:i+h-1}$
$b \in R$	A paranoid term that belongs to the set of real numbers
$\oplus$	The splicing operation
$\odot$	The Exclusive NOR operation

5. Experiment and Result Analysis

5.1. Experimental Dataset

Since there are few authoritative SQL injection data sets publicly available, the data used in this experiment are samples collected by a specific SQL injection sample generation tool [27] in our previous work. In the experiment, $DSi(i = 1, 2, 3, 4, 5, \text{and } 6)$ is used to represent the i-th data set, and Table 2 describes the sample situation of each dataset. Some of the original SQL injection samples in our dataset come from the data warehouse [28]. The others are to run SQLmap [29] and its tamper script on some specific websites over a period of five weeks, capture a large number of SQL injections through Wireshark Traffic, process them, and obtain a total of 10,052 SQL injection samples. These samples encompass the most commonly used open-source and commercial database management systems, including MySQL, Microsoft SQL Server, Oracle, PostgreSQL, etc. This ensures that the training dataset used covers SQL injection attack samples from various types of SQL dialects. This allows our model to maintain a degree of universality and adaptability when faced with SQL injections involving various dialects. Then, we employ the positive sample generation approach elucidated in [27] to generate and select different numbers of injection samples and overlay the collected positive samples in the dataset DSi. The normal samples in the dataset are mainly collected from various social media platforms.

Table 2. Sample situation for each dataset.

Dataset	DS1	DS2	DS3	DS4	DS5	DS6
Negative samples	50,370	50,370	50,370	50,370	50,370	50,370
Generated samples in positive samples	0	10,000	20,000	30,000	40,000	50,000
Positive sample	10,052	20,052	30,052	40,052	50,052	60,052
Positive to negative sample ratio	1: 5	2:5	3:5	4:5	5:5	6:5

5.2. Experimental Environment

To fulfill the requirements of the deep learning-based SQL injection attack detection environment proposed in this paper and substantiate the effectiveness of the detection scheme, all experiments are simulated on a Dell server with 24 cores of Intel Xeon 6240R CPU @2.4 GHz, operating on Ubuntu 20.04, equipped with 64 GB memory, and outfitted with

2×NVIDIA Quadro RTX 5000 GPUs. The evaluation employs a 10-fold cross-validation to compute the average and standard variance of metrics to ensure generality.

5.3. Testing Index

There are various evaluation indicators for classification models, including accuracy rate, precision rate, recall rate, AUC, etc. In the field of SQLIA detection, according to actual needs, the focus of selecting indicators will also be different. When detecting SQL injection attacks, it not only pays attention to the accuracy rate but also has strict requirements on the false negative rate and high requirements on the discrimination speed and generalization performance. This paper mainly uses the deep learning model based on the improved TextCNN and LSTM. The cross-entropy loss serves as the loss function employed during the training procedure in this study. In the domain of SQL injection attack detection, the prevalence of imbalanced positive and negative samples renders the use of accuracy as an evaluation metric unreasonable. Therefore, in addition to the detection accuracy (Accuracy), recall (Recall), and precision (Precision), the evaluation indicators used for the detection of SQL injection attack statements also use F1-Score as a comprehensive evaluation standard; its calculation formula is shown in (4)–(7). Meanwhile, this paper presents a comparative analysis of the model proposed herein with other existing models. Discrimination speed is also very important in the era of big data, so the experiment also uses discrimination speed as an auxiliary evaluation index.

$$\mathrm{Acc} = \left(1 - \frac{errors\_sum}{sum}\right) \times 100\% \tag{4}$$

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \times 100\% \tag{5}$$

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \times 100\% \tag{6}$$

$$\mathrm{F1} = \frac{2 \times \mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}} \times 100\% \tag{7}$$

where *errors_sum* quantifies the quantity of misclassified samples, and *sum* denotes the aggregate quantity of samples.

FP is a false positive example, representing the number of samples in which normal SQL statements are predicted to be SQL injection statements. FN is a false negative example, indicating the number of samples of SQL injection that are predicted to be normal.

The F1 score is an evaluation metric used in machine learning for classification models. It is a statistical metric employed to assess the accuracy of a binary classification model. It considers both the precision and recall of the classification model simultaneously, representing a harmonic mean of the model's precision and recall.

5.4. Experimental Results and Analysis

In order to reduce the problem of overfitting, the experimental results in this paper are based on five ten-fold cross-validations.

5.4.1. Comparative Analysis of Important Hyperparameter Adjustment

Since the effectiveness of the sample expansion method has been verified and the trained classifier has the best detection effect when the ratio of positive and negative samples in the training data set is 1:1, the hyperparameter adjustments are all tested on the data set DS. The hyperparameters mainly adjusted in the experiment include the optimization algorithm, batch_size, K-value in k-max pooling, etc. Since the positive and negative sample ratios in the dataset DS are balanced, the accuracy of the evaluation index can be selected when adjusting the hyperparameters.

(1) Optimization algorithm

The optimization algorithm is to accelerate the convergence of the model during the model training process, and it is an algorithm to find the minimum loss function of the model. Using the correct optimization algorithm, it is possible to quickly and efficiently train the model and find the best parameter values. For the training of deep neural networks, the commonly used first-order optimization algorithms include the gradient descent method, momentum method, AdaGrad, RMSProp, and Adam algorithm.

The gradient descent method is an iterative algorithm. Usually, the appropriate initial parameters are selected first, and an estimated output, y_i, is generated for each input in the training set. The estimated output is then compared with the actual output, y_i, and the average error is obtained after all errors are averaged. The value of the parameter is updated in the direction of the negative gradient, and the objective function is minimized iteratively until convergence. It mainly includes three types: SGD, BGD, and MBGD. SGD only randomly samples one sample at each step to estimate the current gradient. It has fast calculation speeds and takes up little memory. However, due to the limited information used in each step, the estimated gradient will be inaccurate, making the convergence of the objective function unstable and accompanied by violent fluctuations. When BGD updates parameters, it will traverse all the data, resulting in slow parameter updates when the amount of data is large. MBGD combines the advantages of SGD and BGD and only uses a fixed number of samples each time the parameters are updated. Generally, the value is a power of two. This can make full use of matrix operations, reduce the number of parameter updates, and achieve more stable results. Generally, this method is often used in deep learning to calculate the gradient and error according to Formulas (8) and (9) and update the parameters according to Formula (10). Where $\{x_1, x_2, \ldots, x_m\}$ represents a batch of samples randomly drawn from the training set, the number of samples is m, ε represents the learning rate, and θ represents the initial parameters.

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i) \tag{8}$$

$$\theta \leftarrow \theta - \varepsilon g \tag{9}$$

The momentum method (Momentum) uses the concept of momentum in physics to use a new variable v to represent the accumulation of the gradients of the previous few times. It has a certain attenuation in each round and is calculated based on the moving exponentially weighted average of the gradient. The momentum method can better alleviate the problem of large shocks caused by excessive learning rates in MBGD, especially in the case of small gradients, continuous noises, and large noises, and can speed up learning. The gradient and error are calculated according to Formulas (10)–(12), subsequently facilitating the update of the velocity v and parameter θ. Among them, ε represents the learning rate, θ represents the initial parameter, v is the initial velocity, and α is the momentum decay parameter, which represents the degree of decay of the velocity v in each round. Therefore, the momentum method can speed up learning when the front and back gradient directions are consistent and suppress oscillation when the gradient directions are inconsistent.

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i) \tag{10}$$

$$v \leftarrow \alpha v - \varepsilon g \tag{11}$$

$$\theta \leftarrow \theta + v \tag{12}$$

The AdaGrad method enables automatic adjustment of the learning rate by setting a global learning rate ε, where the actual learning rate is inversely proportional to the square root of the cumulative sum of the previous parameter values. The sparseness of the gradient for various parameters is measured by utilizing the sum of squares of the historical gradient. A smaller value indicates a higher level of sparsity. Parameters exhibiting a low update

frequency possess a wider range of updates. In contrast, parameters characterized by a high update frequency tend to diminish the extent of updates, and the learning rate will become smaller and smaller as time changes. The parameter update amount is calculated according to the Formulas (13)–(16). Among them, the initial value of r is 0, ε indicates the global learning rate, θ indicates the initial parameter, and δ indicates the numerical stability, which is a small constant, and the value is about 10^{-7} to prevent division by 0. Hence, the AdaGrad technique enables automatic adjustment of the learning rate, where a higher gradient leads to faster decay of the learning rate while a lower gradient results in slower decay. However, it is still necessary to establish the global learning rate ε.

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i) \tag{13}$$

$$r \leftarrow r + g \odot g \tag{14}$$

$$\Delta\theta \leftarrow -\frac{\varepsilon}{\delta + \sqrt{r}} \odot g \tag{15}$$

$$\theta \leftarrow \theta + \Delta\theta \tag{16}$$

The RMSProp method uses a decay coefficient to make r decay in a certain proportion in each round, similar to the momentum method. Compared with AdaGrad, this method solves the early termination problem in deep learning, is suitable for processing non-stationary targets, and has a good effect on the training of recurrent neural networks. However, with the introduction of a new hyperparameter, the decay coefficient ρ still depends on the global learning rate. The gradient and error are calculated according to Formulas (17)–(20). In these formulas, the initial value of the gradient cumulant r is 0, ε represents the global learning rate, δ represents the numerical stability, and ρ represents the decay rate.

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i) \tag{17}$$

$$r \leftarrow \rho r + (1-\rho) g \odot g \tag{18}$$

$$\Delta\theta \leftarrow -\frac{\varepsilon}{\delta + \sqrt{r}} \odot g \tag{19}$$

$$\theta \leftarrow \theta + \Delta\theta \tag{20}$$

Adam is a RMSprop with a momentum item; the advantage is that the parameters are relatively stable. The calculation formula is shown in (21)–(27). In these formulas, the initial values of the first-order momentum s and the second-order momentum r are 0. The value of the numerical stability δ is 10^{-8}, and the value of the first-order momentum decay coefficient ρ_1 is 0.9. The value of the second-order momentum decay coefficient ρ_2 is 0.999, and the learning rate ϵ requires us to fine-tune during training.

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i) \tag{21}$$

$$s \leftarrow \rho_1 s + (1-\rho_1) g \tag{22}$$

$$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g \tag{23}$$

$$\bar{s} \leftarrow \frac{s}{1-\rho_1} \tag{24}$$

$$\bar{r} \leftarrow \frac{r}{1-\rho_2} \tag{25}$$

$$\Delta\theta = -\varepsilon \frac{\bar{s}}{\delta + \sqrt{\bar{r}}} \tag{26}$$

$$\theta \leftarrow \theta + \Delta\theta \tag{27}$$

The accuracy rate of the model using these five optimization algorithms and the change in the loss value of the model are shown in Figures 4 and 5. The training time of the model using the five optimization algorithms is basically the same, and the relative time of Adam and RMSProp is shorter. Compared with the small-batch gradient descent method, the momentum descent method can speed up the convergence speed to a certain extent. Using the small-batch gradient descent method has relatively large fluctuations in the model training process, and the trained classifier has the lowest accuracy rate on the test set. In the experiment, since both the gradient descent method and the momentum method need to manually set the learning rate, it is difficult to adjust the hyperparameters. Adam only needs to set the global learning rate, which can automatically adjust the learning rate during the model training process, and the parameter adjustment is relatively simple. Based on the figure, it is evident that in the deep learning model employed for SQL injection attack detection, the performance of the Adam optimization algorithm is better and its memory requirement is relatively low. In subsequent experiments, the Adam algorithm is used as the optimization algorithm for training classifiers.

(2) batch_size

This experiment also tuned the batch_size. Based on the size of the training data set, the batch_size was selected as 32, 64, 128, and 256. The accuracy of the model is shown in Figure 6.

From Figure 6, we can see that when the batch_size value is 32, the algorithm fluctuates repeatedly within 80 epochs without convergence, and the accuracy rate is always very low, fluctuating around 0.35. When the batch_size value is 64, the accuracy rate gradually increases with the increase in the number of training rounds. However, the accuracy rate of the model also fluctuates slightly in the later stages of training. When the batch_size is 128 and 256, the final accuracy of the two is almost the same.

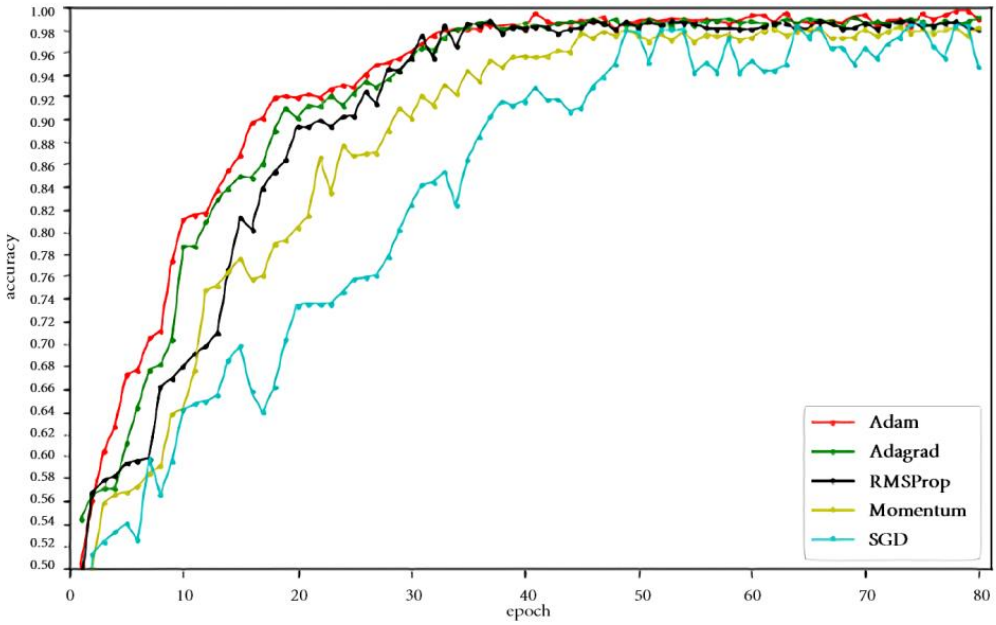

Figure 4. Model accuracy after using different optimization algorithms.

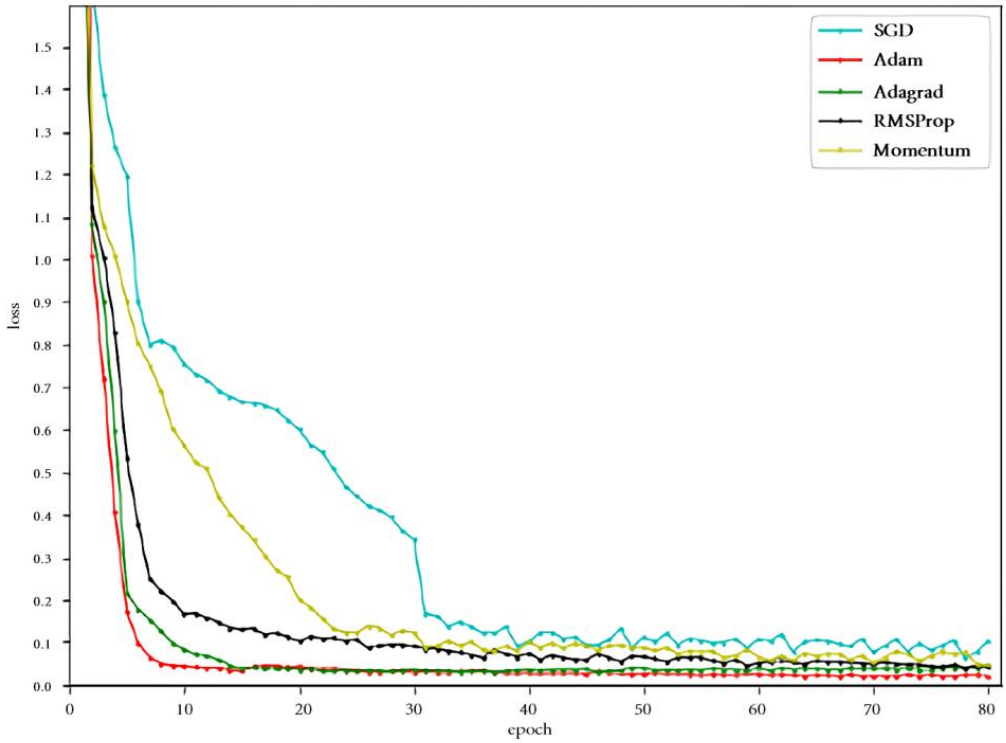

Figure 5. Model loss values after using different optimization algorithms.

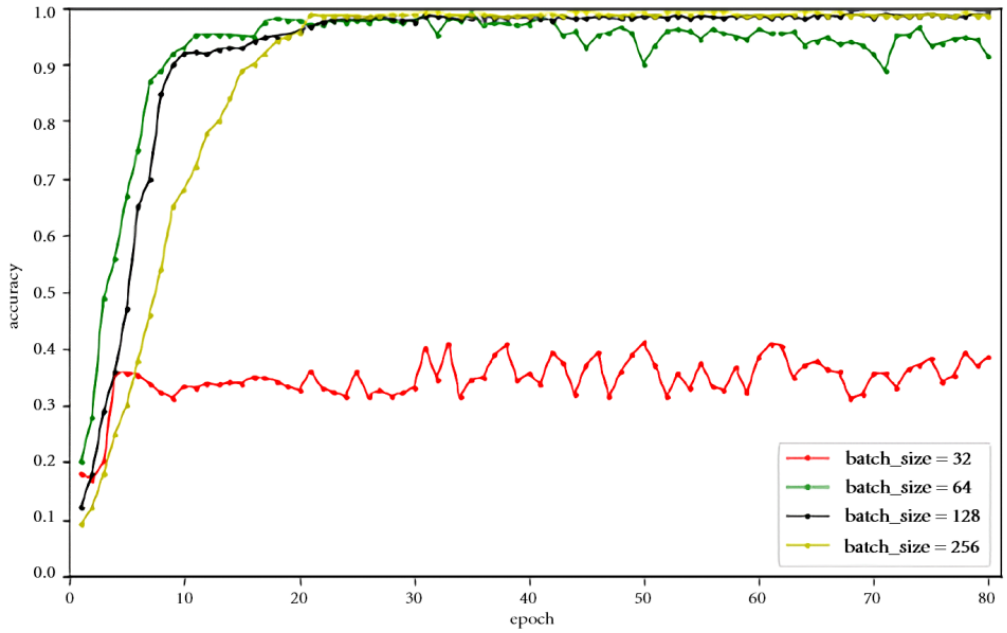

Figure 6. Model accuracy under different batch_sizes.

The training time of the model under different batch_sizes is shown in Table 3. It can be seen from the execution time that when the batch_size exceeds 128, the model loss decreases slowly and the execution speed slows down significantly. However, due to the high precision of the model itself, the adjustment of the batch_size parameter is difficult to deal with individual bad cases, and there is no significant improvement in the final result. Therefore, considering the time and detection rate, the value of the batch_size parameter is finally set to 128, with the highest execution efficiency.

Table 3. Training time of models under different batch_sizes.

Batch_size	32	64	128	256
Training time (s)	78	161	340	672

(3) K-value in K-Max Pooling

In the experiment, the hyperparameter K-value of the k-max pooling layer in the improved TextCNN structure in the deep learning model was also adjusted. When the K-value is 1, 2, 3, and 4, respectively, when K is 1, it is the maximum pooling, the sliding window is 3, 4, 5, the dropout is 0.5, and the L2 constraint value is 3. Among them, the activation layer in TextCNN uses the ReLU function, and the accuracy of the model changes with the K-value, as shown in Figure 7. It can be found that when the K-value is 2, it means that the 2-max pooling operation is used in the model. When other K-values are used, the accuracy of the final model is higher and the detection effect is better. Unlike the enhanced TextCNN proposed in this study, the conventional TextCNN can solely extract the feature possessing the highest eigenvalue, and the accuracy rate is very close in the initial training stage. However, as the number of training rounds increases, the traditional TextCNN algorithm will lose part of the information due to ignoring the secondary features. This results in a final detection accuracy that is slightly lower than the improved TextCNN algorithm by about 0.9%.

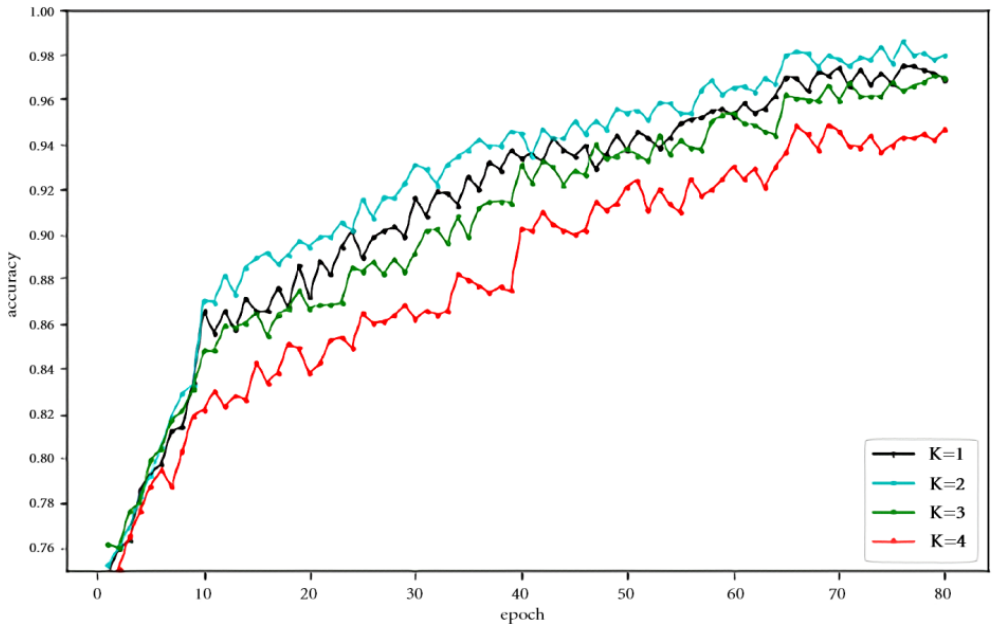

Figure 7. Changes in the accuracy rate of the maximum pooling parameter model with different K-values.

5.4.2. Comparison with Other Detection Methods

(1) Comparison with Classical Machine Learning Methods

The machine learning methods used for comparison include SVM [30], KNN, decision tree, NB, and RF. In the feature selection part, this paper draws from the literature [31], which employs the correlated feature selection (CFS) algorithm for feature selection and uses a genetic algorithm for further optimizing the selection of feature subsets. The CFS algorithm considers both the correlation between features and the target variable as well as redundancy among the features themselves. It can select feature subsets that are more related and less redundant, thereby enhancing the model's performance and interpretability. The genetic algorithm can search a broader feature combination space, which assists in

finding a more optimal feature subset. We tested the above algorithm on the data set DSi (i = 1, 2, 3, 4, 5, 6). The three kernel functions of SVM are introduced below:

(i) Linear kernel (kernel = 'linear')

$$\kappa(x, y) = x^T y + c \tag{28}$$

(ii) Gaussian kernel (kernel = 'rbf')

$$\kappa(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right) \tag{29}$$

(iii) Sigmoid kernel (kernel = 'sigmoid')

$$\kappa(x, y) = \tan h\left(ax^T y + c\right) \tag{30}$$

Choosing the best one for comparison, the experimental test results are shown in Figure 8, where (a), (b), (c), and (d) are the results under the four metrics.

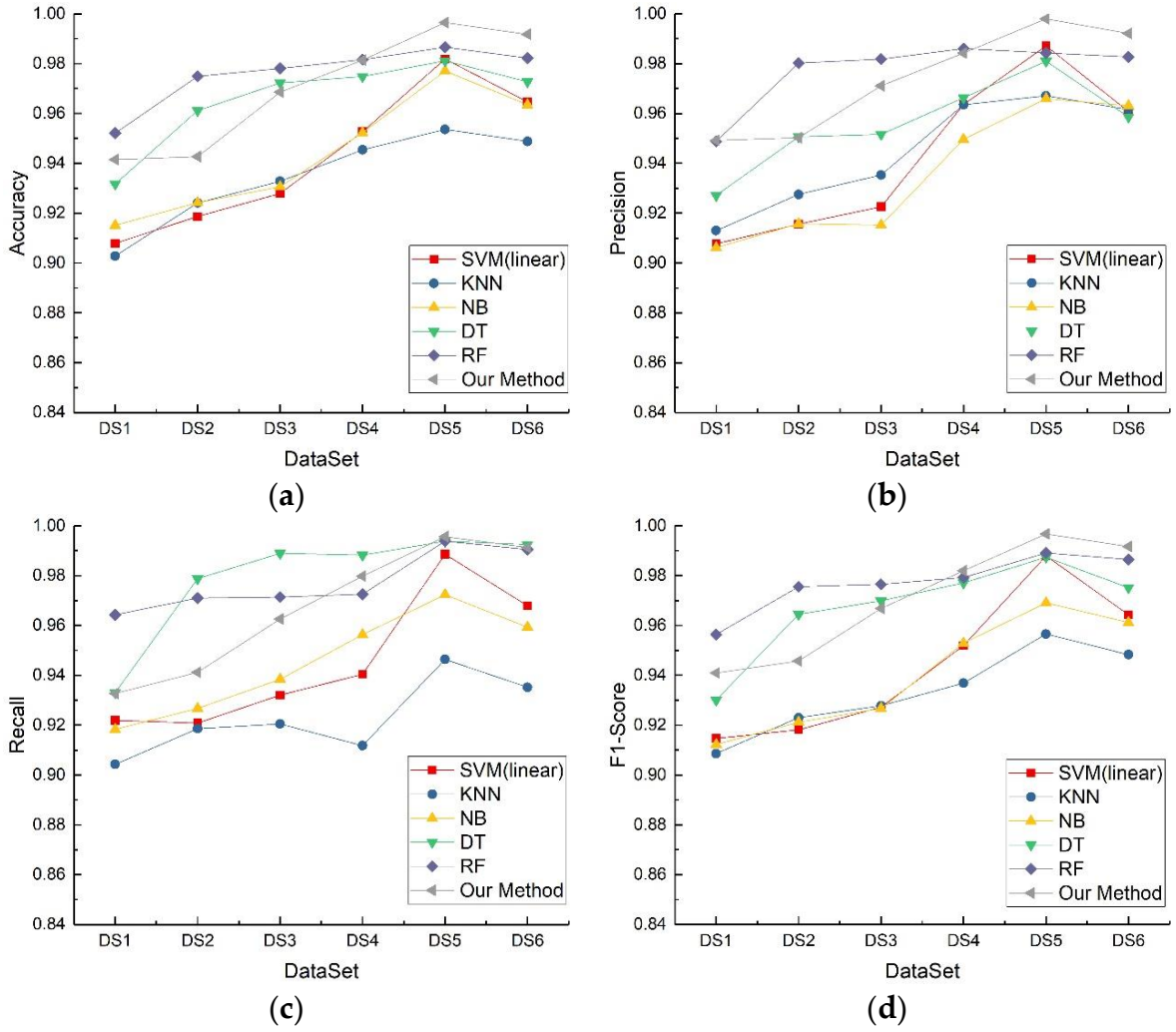

Figure 8. Comparison with classical machine learning approaches. (**a**) Accuracy. (**b**) Precision. (**c**) Recall. (**d**) F1-Score.

After adjusting the parameters during the experiment, it was found that the SVM detection effect using the linear kernel function is better than that using other kernel

functions. From Figure 8, it can be seen that the random forest detection effect in the shallow machine learning method shows the best performance. On the data set DS1, the detection accuracy F1-Score reaches 95.64%. The possible reason could be attributed to the better generalization ability of random forests, as they can automatically handle feature selection, capture nonlinear relationships, and exhibit robustness against noise and outliers. However, one drawback is that the model training time is relatively longer. Following this, the decision tree exhibits noteworthy performance, as it can handle mixed data types, process large-scale datasets, and remain insensitive to feature selection. This may be why the decision tree algorithm performs well.

Then, we have Naive Bayes and SVM with a linear kernel, both of which showed similar detection performance. The KNN algorithm has the worst detection effect, at only 91.22%. The possible reasons for the poor performance of the KNN algorithm can be analyzed as follows: Firstly, the KNN algorithm is sensitive to feature dimensionality, particularly in high-dimensional data where the distances between data points become sparse, leading to inaccurate distance calculations. Moreover, the performance of the KNN algorithm highly depends on the selection of an appropriate K-value, and it is also sensitive to the weights of different features in the feature space. These factors can contribute to the suboptimal performance of the algorithm. With the increase in training samples, the F1-Score of detecting SQL injection using various shallow machine learning algorithms is also gradually increasing. The ratio of positive and negative samples is between 4:5 and 5:5. The largest increase is when the ratio is 1:1 and the experimental detection effect reaches its highest value. Then, with the increase in the number of positive samples, the detection effect gradually shows a downward trend.

It can be found that after the expansion of the training data set, the precision rate, accuracy rate, and F1-Score of the shallow machine learning algorithm have been greatly improved. The classifier is obtained using the deep learning algorithm based on the improved TextCNN, LSTM, and ATTENTION mechanisms to automatically learn and extract the feature training of various SQL injection statements. All evaluation indicators of the classifier for SQLIA detection we proposed are above 99.57%. Deep learning models can automatically learn feature representations through multi-layer neural networks, extracting high-level abstract features from raw data and reducing reliance on manual feature engineering. They also exhibit advantages in handling large-scale data, enabling better utilization of abundant information in the data and facilitating the modeling and understanding of highly nonlinear and complex problems. The performance in SQL injection detection is significantly better than the method using shallow machine learning.

(2) Comparison with Common Deep Neural Networks

It is compared with several commonly used deep neural networks (CNN, RNN, and TRANSFORMER) [32] in the field of deep learning. For a fair comparison, the above network also sets three hidden layers as per the method proposed in this paper. CNN uses a one-dimensional convolution kernel for detection, and TRANSFORMER uses a three-layer encoder structure. The GRU + ATTENTION network structure is input → GRU1 → GRU2 → ATTENTION → Dense. After parameter tuning, the best results are selected for comparative analysis. The experimental results of each method are shown in Figure 9, where (a), (b), (c), and (d) are the results under the four metrics.

As shown in Figure 9, from the perspective of the comprehensive index F1 of SQL injection detection effect, the effect of TextCNN on the dataset DS1 is basically the same as that of TRANSFORMER and RNN, and both are relatively poor, followed by the GRU+ ATTENTION structure. The method in this paper has the highest F1 value. With the increase in the number of positive samples in the training samples, the ratio of positive and negative samples gradually tends to one, and the detection rate of the classifier obtained by training is also gradually increasing. The detection effect on the data set DS5 reaches its peak, and the false negative rate is less than 1%. At this time, the deep learning method proposed in this paper still has the best effect, followed by the GRU + ATTENTION structure, and TextCNN has the worst comprehensive effect.

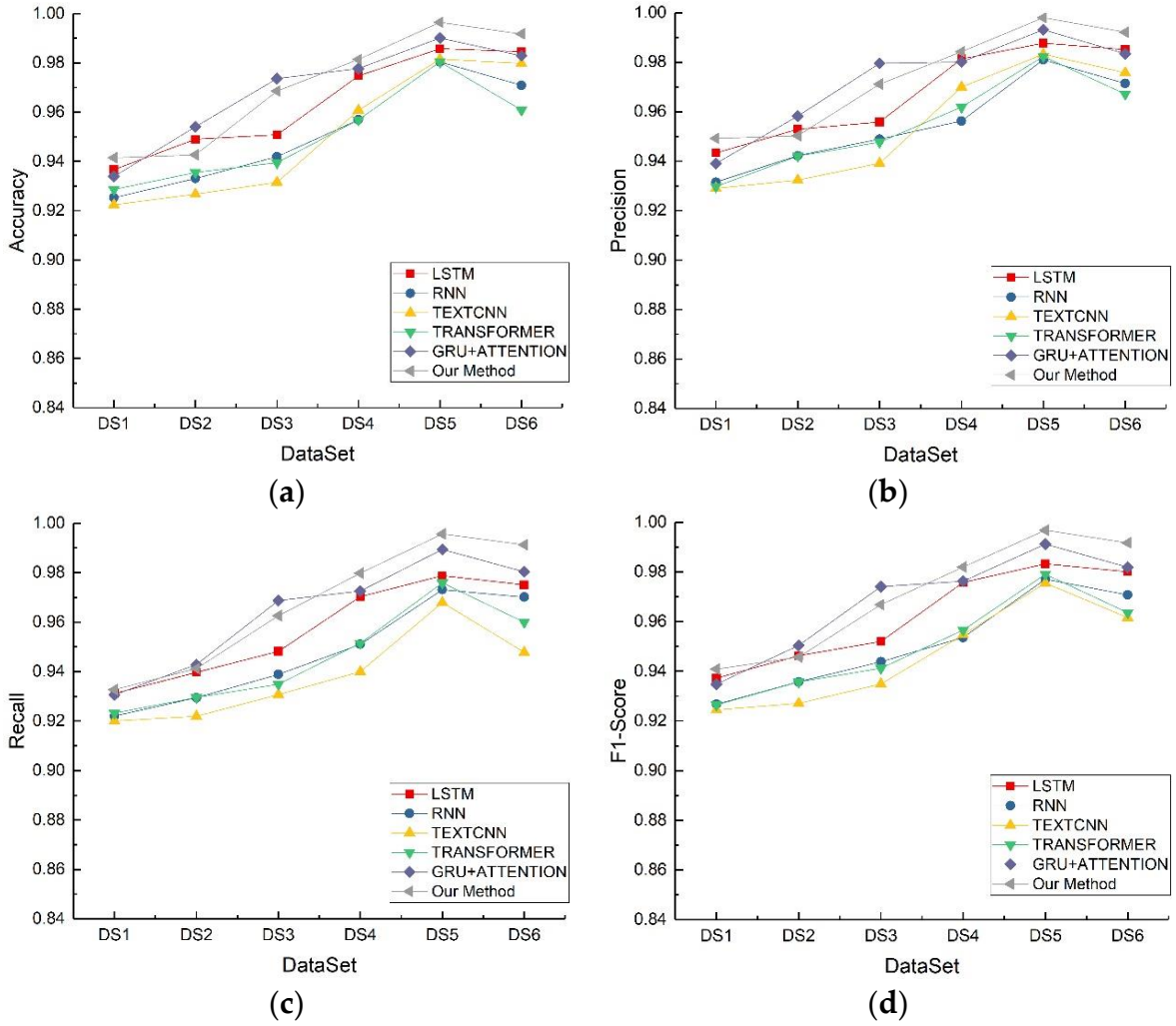

Figure 9. Comparison with common deep neural networks. (**a**) Accuracy. (**b**) Precision. (**c**) Recall. (**d**) F1-Score.

CNN needs to obtain the local correlation of data, while RNN mainly obtains the sequence characteristics of data. The test results show that the sequence characteristics of SQL injection data can better reflect the characteristics of SQL injection data than local correlation. Because the TRANSFORMER structure uses the ATTENTION mechanism completely, it is difficult to learn the sequence, so the effect is not very good. The effect of adding the ATTENTION mechanism to the GRU model is second only to the method proposed in this paper. It shows that the use of the ATTENTION mechanism can solve the problem of long-distance gradient disappearance in LSTM or GRU to a certain extent, thereby improving the detection effect. It can also be seen from Figure 9 that the deep learning method based on the improved TextCNN, LSTM, and ATTENTION mechanisms proposed in this paper is basically better than other structure detection methods in four indicators. This is mainly because the structure proposed in this paper takes into account both the local features in the sample and the sequence features in the sample. It also improves the problem of the model itself. Therefore, the method in this paper is more suitable for SQL injection attack detection than other deep neural networks.

According to Figure 10, it can be concluded that our method achieves an average false negative rate of 3.2% and an average false positive rate of 1.2% across different datasets. As the number of positive samples in the training set increases, the false negative rate and false positive rate of our method gradually decrease. The lowest rates are achieved on the DS5 dataset, with a false negative rate of 0.4% and a false positive rate of 0.39%. These

results demonstrate that the proposed deep learning-based SQLIA detection approach not only effectively improves the detection rate of SQLIA but also ensures low false negative and false positive rates.

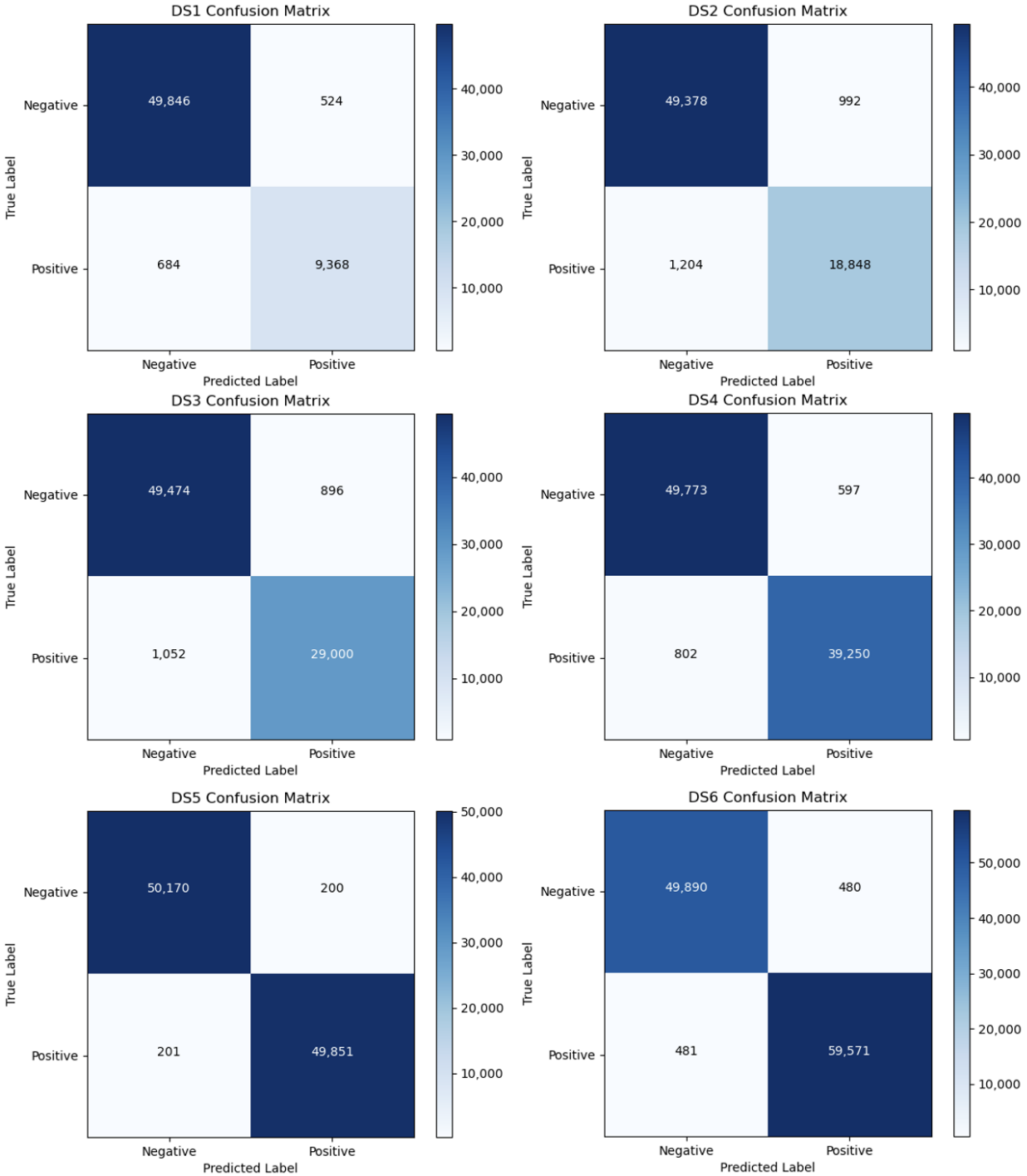

Figure 10. Confusion matrix of our method on different datasets.

6. Conclusions

This paper proposes a deep learning-based approach for SQL injection attack detection, addressing the challenges of diverse attack patterns and the difficulty of extracting effective features. The proposed method leverages the fusion of TF-IDF and Word2Vec embeddings to vectorize SQL samples while preserving maximum information. It employs an improved TextCNN to automatically extract important local information from the vectorized SQL

samples. Additionally, a Bi-LSTM network is utilized to capture the sequential information in the samples. Finally, an ATTENTION mechanism is applied to enhance the weights of important features in the samples, reducing the impact of noise and improving the detection accuracy. Finally, a SQL injection attack detection system is designed according to the proposed scheme, and comparative experiments are carried out on multiple datasets. In comparison to the effectiveness of classical machine learning methods (such as SVM, KNN, Decision Tree, NB, and RF), the experimental results show that the network structure designed in this paper to detect SQL injection attacks improves accuracy and reduces the rate of false negatives and false positives. Additionally, the timeliness of discrimination also shows good performance, with excellent comprehensive detection results.

The main focus of this paper is to detect and evaluate traditional first-order SQL injection attacks. Due to the complexity and specific characteristics of second-order SQL injection attacks, it is currently challenging to achieve automated detection or high detection accuracy. Therefore, the next step of our research will concentrate on developing effective detection techniques for novel second-order SQL injection attacks under complex conditions. Furthermore, there exist adversarial attack methods [33] that pose challenges to the robustness of deep learning detection models. Therefore, further research is needed to investigate the robustness of the proposed methods, aiming to enhance the model's adversarial resilience and stability. This will be a crucial evaluation criterion in the future for the field of SQL injection detection.

Author Contributions: Conceptualization, H.S. and Q.L.; methodology, Q.L.; software, H.S.; validation, H.S. and Q.L.; formal analysis, Y.D. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (62172055, U20B2045), the Natural Science Foundation of Ningxia (2022AAC03620), the Project (2021XD-A09).

Institutional Review Board Statement: Not applicable, this study did not require ethical approval.

Informed Consent Statement: Not applicable, this study did not involve information about other people.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to the data source involves private or sensitive information.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Uwagbole, S.O.; Buchanan, W.J.; Fan, L. Applied machine learning predictive analytics to SQL injection attack detection and prevention. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 1087–1090.
2. Krishnan, S.S.A.; Sabu, A.N.; Sajan, P.P.; Sreedeep, A.L. SQL injection detection using machine learning. *Rev. Geintec-Gest. Inov. Tecnol.* **2021**, *11*, 300–310.
3. Farooq, U. Ensemble machine learning approaches for detection of sql injection attack. *Teh. Glas.* **2021**, *15*, 112–120. [CrossRef]
4. Adebiyi, M.O.; Arowolo, M.O.; Archibong, G.I.; Mshelia, M.D.; Adebiyi, A.A. An SQL injection detection model using chi-square with classification techniques. In Proceedings of the 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa, 9–10 December 2021; pp. 1–8.
5. Zhang, Y.; Wu, J.; Zhou, C.; Cai, Z. Instance cloned extreme learning machine. *Pattern Recognit.* **2017**, *68*, 52–65. [CrossRef]
6. McWhirter, P.R.; Kifayat, K.; Shi, Q.; Askwith, B. SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel. *J. Inf. Secur. Appl.* **2018**, *40*, 199–216. [CrossRef]
7. Wang, Y.; Wang, D.; Zhao, W.; Liu, Y. Detecting SQL vulnerability attack based on the dynamic and static analysis technology. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung, Taiwan, 1–5 July 2015; pp. 604–607.
8. Gould, C.; Su, Z.; Devanbu, P. JDBC checker: A static analysis tool for SQL/JDBC applications. In Proceedings of the 26th International Conference on Software Engineering, Edinburgh, UK, 28 May 2004; pp. 697–698.
9. Wassermann, G.; Gould, C.; Su, Z.; Devanbu, P. Static checking of dynamically generated queries in database applications. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2007**, *16*, 14-es. [CrossRef]

10. Yi, W.; Zhoujun, L.; Tao, G. Literal tainting method for preventing code injection attack in web application. *J. Comput. Res. Dev.* **2012**, *49*, 2414–2423.
11. Appiah, B.; Opoku-Mensah, E.; Qin, Z. SQL injection attack detection using fingerprints and pattern matching technique. In Proceedings of the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 24–26 November 2017; pp. 583–587.
12. Bisht, P.; Madhusudan, P.; Venkatakrishnan, V.N. CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2010**, *13*, 1–39. [CrossRef]
13. Halfond, W.G.J.; Orso, A. AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks. In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, Long Beach, CA, USA, 7–11 November 2005; pp. 174–183.
14. Xiao, Z.; Zhou, Z.; Yang, W.; Deng, C. An approach for SQL injection detection based on behavior and response analysis. In Proceedings of the 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), Guangzhou, China, 6–8 May 2017; pp. 1437–1442.
15. Choi, J.; Kim, H.; Choi, C.; Kim, P. Efficient malicious code detection using N-gram analysis and SVM. In Proceedings of the 2011 14th International Conference on Network-Based Information Systems, Tirana, Albania, 7–9 September 2011; pp. 618–621.
16. Lei, X.; Qu, J.; Yao, G.; Chen, J.; Shen, X. Design and implementation of an automatic scanning tool of SQL injection vulnerability based on Web crawler. In *Security with Intelligent Computing and Big-Data Services: Proceedings of the Second International Conference on Security with Intelligent Computing and Big Data Services (SICBS-2018)*; Springer International Publishing: Cham, Switzerland, 2020; pp. 481–488.
17. Komiya, R.; Paik, I.; Hisada, M. Classification of malicious web code by machine learning. In Proceedings of the 2011 3rd International Conference on Awareness Science and Technology (iCAST), Dalian, China, 27–30 September 2011; pp. 406–411.
18. Akcay, S.; Kundegorski, M.E.; Willcocks, C.G.; Breckon, T.P. Using deep convolutional neural network architectures for object classification and detection within X-ray baggage security imagery. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 2203–2215. [CrossRef]
19. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
20. Zhou, T.; Sun, X.; Xia, X.; Li, B.; Chen, X. Improving defect prediction with deep forest. *Inf. Softw. Technol.* **2019**, *114*, 204–216. [CrossRef]
21. Zhuo, Z.; Cai, T.; Zhang, X.; Lv, F. Long short-term memory on abstract syntax tree for SQL injection detection. *IET Softw.* **2021**, *15*, 188–197. [CrossRef]
22. Dawadi, B.R.; Adhikari, B.; Srivastava, D.K. Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks. *Sensors* **2023**, *23*, 2073. [CrossRef]
23. Gandhi, N.; Patel, J.; Sisodiya, R.; Doshi, N.; Mishra, S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In Proceedings of the 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 17–18 March 2021; pp. 378–383.
24. Li, Q.; Li, W.; Wang, J.; Cheng, M. A SQL injection detection method based on adaptive deep forest. *IEEE Access* **2019**, *7*, 145385–145394. [CrossRef]
25. Alarfaj, F.K.; Khan, N.A. Enhancing the Performance of SQL Injection Attack Detection through Probabilistic Neural Networks. *Appl. Sci.* **2023**, *13*, 4365. [CrossRef]
26. Kim, Y. Convolutional Neural Networks for Sentence Classification. *arXiv* **2014**, arXiv:1408.5882.
27. Li, Q.; Wang, F.; Wang, J.; Li, W. LSTM-based SQL injection detection method for intelligent transportation system. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4182–4191. [CrossRef]
28. SQL Injection Attack Dataset. Available online: https://github.com/client9/libinjection/ (accessed on 15 September 2022).
29. Sqlmap Tool. Available online: https://github.com/sqlmapproject/sqlmap (accessed on 24 October 2022).
30. Alkhathami, J.M.; Alzahrani, S.M. Detection of Sql Injection Attacks Using Machine Learning in Cloud Computing Platform. *J. Theor. Appl. Inf. Technol.* **2022**, *100*, 5446–5459.
31. Ross, K.; Moh, M.; Moh, T.S.; Yao, J. Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. In Proceedings of the ACMSE 2018 Conference, Richmond, KY, USA, 29–31 March 2018; pp. 1–8.
32. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5998–6008.
33. Guan, Y.; He, J.; Li, T.; Zhao, H.; Ma, B. SSQLi: A Black-Box Adversarial Attack Method for SQL Injection Based on Reinforcement Learning. *Future Internet* **2023**, *15*, 133. [CrossRef]