Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.063319

ARTICLE

Research on SQL Injection Detection Technology Based on Content Matching and Deep Learning

Yuqi Chen[1,2], **Guangjun Liang**[1,2,3,*] **and Qun Wang**[1,2,3]

[1]Department of Computer Information and Cyber Security, Jiangsu Police Institute, Nanjing, 210031, China
[2]Jiangsu Electronic Data Forensics and Analysis Engineering Research Center, Nanjing, 210031, China
[3]Jiangsu Provincial Public Security Department Key Laboratory of Digital Forensics, Nanjing, 210031, China

*Corresponding Author: Guangjun Liang. Email: lianggjun@126.com

Received: 11 January 2025; Accepted: 03 April 2025; Published: 09 June 2025

ABSTRACT: Structured Query Language (SQL) injection attacks have become the most common means of attacking Web applications due to their simple implementation and high degree of harm. Traditional injection attack detection techniques struggle to accurately identify various types of SQL injection attacks. This paper presents an enhanced SQL injection detection method that utilizes content matching technology to improve the accuracy and efficiency of detection. Features are extracted through content matching, effectively avoiding the loss of valid information, and an improved deep learning model is employed to enhance the detection effect of SQL injections. Considering that grammar parsing and word embedding may conceal key features and introduce noise, we propose training the transformed data vectors by preprocessing the data in the dataset and post-processing the word segmentation based on content matching. We optimized and adjusted the traditional Convolutional Neural Network (CNN) model, trained normal data, SQL injection data, and XSS data, and used these three deep learning models for attack detection. The experimental results show that the accuracy rate reaches 98.35%, achieving excellent detection results.

KEYWORDS: SQL injection; network security; deep learning; convolution neural network

1 Introduction

In the current era of accelerated digital transformation, web applications have become the primary platforms for various activities—including enterprise operations, social interactions, and financial transactions. As network technology advances at a remarkable pace, the security threats faced by web applications are growing increasingly complex and diverse. Structured Query Language (SQL) injection attacks, one of the most common and damaging network attack methods, pose a serious threat to the security and stability of web applications. The principle of SQL injection attacks is that attackers take advantage of the vulnerabilities in the user input validation mechanisms of Web applications to insert malicious SQL code into the interaction process between the application and the database, thereby achieving illegal data access, tampering, or destruction [1,2]. For example, in 2023, a well-known online education platform was attacked by SQL injection, and a large number of students' learning records, test scores, and personal identity information were leaked. This not only violated the privacy of students but also caused great damage to the reputation of the platform, leading to a large number of user losses. The platform had to invest huge amounts of funds in data restoration and security reinforcement. The frequent occurrence of such incidents highlights the urgency and importance of strengthening the security protection of Web applications, especially improving SQL injection detection technology.

URLs are often used by attackers to carry out various network attacks. SQL injection and Cross-Site Scripting (XSS) attacks accounted for more than 80% of the attacks using URLs. Some researchers have used machine learning methods to detect security threats. Bobade et al. [3] focus on SQL injection in the realm of cyber security. They conduct a detailed study of SQL injection and its various types. This work explores how attackers execute malicious code to manipulate SQL databases, thereby conducting attacks. To counter this threat, the authors assess prevention and detection techniques for SQL injection. They conduct an extensive review of relevant literatures and, in the results section, present a comparative analysis of classical SQL injection (SQLi), advanced SQLi, and deep-learning-based approaches. Anu et al. [4] analyzes the technical details of SQL injection attack problems and conducts a comprehensive literature review. They develop a machine learning paradigm-based solution for detecting SQL injection attacks. By leveraging the Kaggle dataset, various machine learning techniques are employed, including the K-Nearest Neighbors (KNN) Classifier, Random Forest, Voting Classifier, and Logistic Regression, to identify such attacks. Mohanraj et al. [5] focus on how malicious users input carefully crafted SQL queries into the input fields of web applications. When these queries are executed by the application, unauthorized operations on the database can be carried out, leading to data security risks and undermining users' trust in web applications. Additionally, the authors explore hashing as an effective preventive measure, which transforms plain text into a fixed length string (hash value or digest), and study how to apply hashing technology to protect the data stored in databases from SQL injection attacks. Okesola et al. [6] demonstrate how Parameterized Queries can be used to defend against SQL Injection attacks. By using prepared statements in Java and employing LoginController as the Servlet to control the application login process, Parameterized Queries are successfully integrated into an e-commerce application. The implementation results show that SQL injection is no longer possible because the input is set as data, and data is treated differently from codes, preventing attackers' codes from being executed. Abdullah et al. [7] explore the operating mechanisms of SQL injection attacks and highlight the role of port scanning in such attacks. The research reveals that web applications vulnerable to SQL injection can be exploited by attackers using port scanning to identify open or closed ports, such as MySQL's default port 3306. Then, malicious SQL code can be injected into user input fields to modify database queries and endanger the security of the application. As SQL injection attacks predominantly stem from inadequate input validation and improper parameter handling in SQL queries, the authors stress the indispensability of implementing effective security measures such as input validation and parameterization. These measures can reduce the risk of SQL injection attacks and improve the overall security of applications. Guan et al. [8] propose a deep-learning-based model for detecting SQL injection and XSS attacks. This model can fuse local and global features, capture long-term dependencies in sequential data, and focus more on the relevant parts of the input data. Experimental results show that this model outperforms other mainstream methods on multiple datasets. Zhang et al. [9] employ word-level convolutional neural networks to classify URLs. Specifically, they parse URLs in the dataset based on special characters to construct a corpus. This research leverages deep-learning techniques to develop detection and recognition models, achieving high accuracy. However, current studies rely on constructing word vectors to generate training data, which makes the processing procedure rather cumbersome. Moreover, as the volume of data grows, the retrieval performance in large-scale data deteriorates accordingly. Gandhi et al. [10] proposed CNN-BiLSTM model achieves a remarkable accuracy of 98% and shows superior performance compared to other machine learning algorithms. Additionally, the authors conduct a comparative study of different types of machine learning algorithms used for SQLI attack detection. They demonstrate the performance of various algorithms in terms of accuracy, precision, recall, and F1 score relative to the proposed CNN-BiLSTM model in detecting SQL injection attacks. Bouafia et al. [11] adopt a combined approach by utilizing the Acunetix Web Vulnerability Scanner for detection, employing Burp Suite to capture HTTP requests with parameters vulnerable to SQL injection, and using SQLMAP as an automated SQL injection operation tool. The authors carry out practical tests using the real-world Damn Vulnerable Web Application (DVWA). They simulate SQL injection attack scenarios across all available security levels: low, medium, and high. The test results indicate that even though security mechanisms are strengthened at different security levels, this solution exhibits extremely high performance throughout these levels.

This work proposes a detection method for SQL injection attacks and XSS attacks in Uniform Resource Locators (URLs). Compared with similar work, the main contributions and innovations of this paper are as follows: (1) After the data of the dataset and the retrograde preprocessing, the word segmentation based on the content matching method can avoid the loss of effective information. (2) Word2Vec is used to transform the data set in vector form, and the data based on content matching is used as the training object to ensure the detection effect of SQL injection. (3) The traditional CNN model is meticulously optimized and adjusted. Normal data, SQL injection data, and XSS data are utilized for training. Furthermore, attack detection is accomplished by employing three distinct models, namely the CNN model, the CNN-RNN model, and the CNN-LSTM model. (4) Through extensive experimentation, our proposed method has been proven to effectively detect SQL injection attacks and reduce the number of false positives to a certain extent compared with the classical machine learning algorithm.

2 Related Work

The implementation of an SQL injection attack consists of several factors. On one hand, the process of writing and developing web applications is highly intricate. Owing to the varying programming proficiency levels and relatively weak security awareness among programmers, some vulnerabilities emerge during the program development stage. On the other hand, after a web application is deployed, the data entered by users is not rigorously filtered, and the server side fails to conduct strict verification. Consequently, attackers are able to construct malicious statement inputs, execute malicious commands within the database, and achieve the goal of obtaining sensitive database information and even gaining control over the entire server. Injection attacks can be divided into out-of-band injection, in-band injection and logical inference injection. Existing SQL injection attack detection methods predominantly rely on the approach of assessing the validity of SQL statements to detect attacks. Based on their detection principles, these methods can be roughly classified into dynamic analysis techniques, machine learning-based methods, and deep learning approaches.

Dynamic analysis technology falls under the category of black-box testing methods. It involves scanning the system with dynamic detection attacks while the system is in operation. This approach is typically adopted during the system acceptance phase or the online operation phase. Subsequently, based on the scanning results, it determines whether the system harbors SQL injection vulnerabilities. Ray et al. [12] have developed an analysis model applied to the system. This model can determine whether there is SQL injection through stain analysis. Extract the constructed SQL statement and analyze whether there is dirty data in the syntax tree structure tree according to the SQL statement parsing principle [13]. Due to this technology needs to analyze the source code and cannot find some attacks that occur at runtime, the detection efficiency is not high. Most of these methods require code rewriting technology or precompiling after modifying the program engine to load the detection components, which is highly coupled with the original application, and it is difficult to conduct mass detection.

The method of machine learning to detect SQL injection is generally to collect a large number of URLs, extract important features from them, and conduct modeling and analysis through clustering algorithm or classification algorithm. Common classification algorithms include SVM, decision tree, logical regression, etc. Using a preprocessed database of 46,392 SQL queries, Thalji et al. [14] propose a novel optimized approach called the Autoencoder network (AE-Net) for automatic feature engineering. The proposed AE-Net extracts new high-level deep features from SQL textual data and then inputs them into machine learning models for performance evaluation. Extensive experimental evaluation shows that the extreme gradient boosting classifier outperforms existing studies in SQL injection detection, with an impressive k fold accuracy score of 0.99. Moreover, the authors further enhance the performance of each applied learning approach through hyperparameter tuning and validate it via k fold cross validation. They also apply statistical t test analysis to assess performance variations.

Now deep learning has been used by many scholars to study SQL injection detection. Deep learning is to build a deep representation learning structure and neural network based on the shallow neural network, and then use the deep nonlinear structure to better fit the complex function, greatly enhancing its own learning ability. Automatic feature extraction is a major advantage of deep learning compared to traditional machine learning. Deep learning does not require manual marking of sample categories. Instead, it can adaptively extract appropriate schemes from massive amounts of data through the integration of the feature extraction component and the representation component. Cao et al. [15] used the BERT pre training model from the Tramsformers offline model. The bert model structure used attention mechanism to generate word vectors, achieving a high detection rate. Li et al. [16] proposed a method of data enhancement to alleviate the over-fitting of the model by expanding the SQL injection attack samples. However, this method cannot cope with the new SQL injection attack types, and the quality deep learning methods that rely on prior knowledge and generate samples mostly use word embedding to code the SQL samples. The above scholars have achieved good results, but there are still several problems: (1) Based on character segmentation, some unique key words in the sentence will lose valid information. (2) Unable to obtain the distribution and type of special characters and the relationship with surrounding words. (3) The accuracy and testing speed of the deep learning model still need to be verified. Ming et al. [17] introduce a novel deep learning based SQL detection system named Bidirectional LSTM CNN based on Multi. The proposed method implements a pre-processing step that generates multiple views from SQL data by semantically encoding SQL statements into their corresponding SQL tags. By utilizing two different main layers which are bidirectional LSTM and CNN, the proposed method learns a joint latent space from multi view representations. Liu et al. [18] propose a novel deep learning framework that integrates Bidirectional Encoder Representations from Transformers (BERT) and Long Short Term Memory (LSTM) networks to enhance the detection of SQL injection attacks. By 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, and then generates embedding vectors that can effectively identify malicious SQL query patterns. Fathi et al. [19] present two advanced models for SQL injection (SQLI) detection, using a LSTM neural network as a deep learning model and other traditional Machine Learning classifiers. A key challenge tackled in this study is data imbalance, a common problem in cybersecurity datasets where the number of malicious instances is far fewer than that of benign ones. This imbalance can cause Machine Learning models to be biased towards the majority class. To address this, the research uses a variety of data preprocessing techniques that significantly improve model performance. Janet et al. [20] explore the possibility of constructing effective SQLi detectors through machine learning. Specifically, we investigate the impacts of contextualized and non-contextualized embedding methods when converting SQL queries into vector space. Our research results demonstrate the superiority of the contextualized embedding method. Across various classification algorithms, its accuracy consistently remains above 99%, and it reduces the model training time by 31 times. Souza et al. [21] propose an SQLi detection solution that combines Regular Expressions (RegEx) and Machine Learning (ML), named the Two Layer approach of SQLi Detection (2LD SQLi). The RegEx serves as the first layer of filtering to protect against SQLi inputs, improving the response time of 2LD SQLi through RegEx filtering. After this filtering, it is analyzed by an ML model to detect SQLi, increasing the detection accuracy. How to aptly adapt to the idiosyncrasies of different databases constitutes the cardinal challenge. Lakhani et al. [22] propose a novel approach that uses Natural Language Processing (NLP) and BERT for feature extraction. This approach can adapt to SQLI variants, achieving an accuracy of 97%, a false-positive rate of 0.8%, and a false-negative rate of 5.8%. Marinelli et al. [23] take SQL injection, which poses risks to production systems, as a base case for the research. By using causal tracing techniques, they determine that the capabilities of SQL injection knowledge are most prominent in the last layers of the network. Moreover, through training a K-Nearest Neighbors (KNN) classifier, they confirm that there may be some shared representation of SQL injection knowledge across models. When training the classifier on the activations of GPT-Neo and attempting to classify GPT-2, they find that the accuracy reaches 52%. Huang et al. [24] use prompt engineering and instruction fine-tuning techniques to develop a specialized large language model for SQL injection attack detection. They analyze the impact of the number of iteration rounds, the quantity of fine-tuning samples, and inference parameters on the model's performance to enhance the detection ability of the large language model. They utilize the powerful semantic understanding ability of the large language model to significantly reduce the false positive rate. The authors conduct an experimental analysis on the proposed specialized large language model using the Kaggle dataset. The model achieves an accuracy rate of over 99.85%, a false alarm rate of less than 0.2%, and an F1 score of 0.999. Compared with the current state-of-the-art methods for SQL injection attack detection, its detection performance shows a significant improvement.

3 SQL Injection Detection Framework for Deep Learning

Traditional SQL injection detection techniques have issues such as delayed rule updates, lack of context information, and poor adaptability to new type attacks. Therefore, by combining deep learning methods, we propose a deep learning based SQL injection detection framework. Within this framework, to prevent the loss of effective information, during the feature extraction process, the data in the dataset undergoes preprocessing and reverse processing, and word segmentation is performed based on content matching. This approach enables clear marking of key information, which facilitates the neural network classifier in extracting more representative features. At present, the more suitable models for detection are CNN, long-short term memory (LSTM) network, recurrent neural network (RNN) and the hybrid and improved models of the above networks. Indeed, word embedding methods represented by BERT can achieve better performance as they can take context relationships into account. However, BERT has high requirements for computing resources, is overly reliant on data, and has poor interpretability. Since SQL injection detection places more emphasis on attack tracing, this paper focuses on the Content Matching method.

3.1 SQL Injection Detection Framework

The framework is meticulously divided into four distinct modules, as vividly illustrated in the Fig. 1. To commence, the framework initiates its operation by leveraging the data acquisition module. This crucial component scours through a vast array of sources, be it databases, log files, or network traffic captures, with the sole aim of gathering all relevant data. Once the raw data is amassed, it then passes through the data preprocessing module. Here, a series of intricate operations are carried out. Noise reduction techniques are applied to eliminate any spurious or erroneous entries that could potentially skew the results. Outlier detection and removal procedures ensure that extreme values that do not conform to the general data pattern are weeded out. Data normalization is also implemented to bring all the data to a common scale, facilitating smoother processing downstream. Through these combined efforts of the data acquisition and preprocessing modules, high-quality data sets, which serve as the bedrock for the subsequent model training and testing, are successfully constructed.

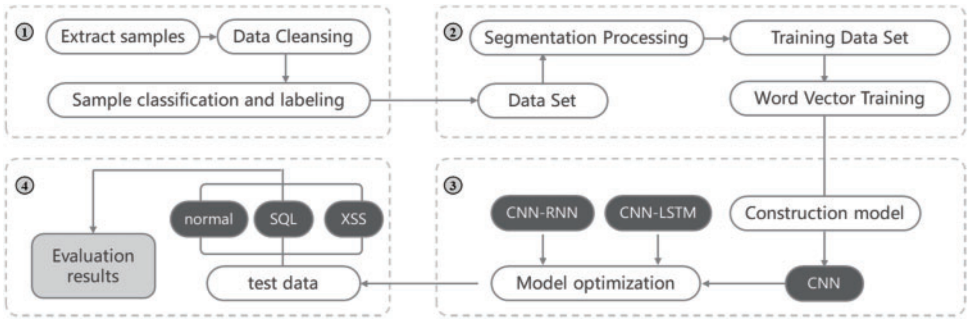

Figure 1: SQL injection detection framework

Subsequently, the model training module takes center stage. It takes the refined data sets and subjects them to an elaborate training regimen. Multiple iterations of training are conducted, with different combinations of hyperparameters being tested and tweaked. This exhaustive process aims to identify the optimal configuration that will yield the model with the best performance. The training process involves not only adjusting the weights and biases of the neural network but also evaluating various performance metrics such as accuracy, precision, recall, and F1 score at each iteration. By closely monitoring these metrics, the training can be fine-tuned to converge towards the most effective model.

Finally, when the model with the peak performance has been obtained, it is put to the test using the test samples. These test samples, carefully selected to represent a diverse range of real-world scenarios, are fed into the trained model. The model then makes predictions, and the results are compared against the ground truth to assess its effectiveness in accurately detecting SQL injection attacks.

In the main process, the initial chaos of raw data is first methodically sorted. This sorting operation is not a simple rearrangement but rather a strategic organization. It groups the data based on various factors such as source, timestamp, or data type to form a comprehensive dataset that is suitable for training, verification, and detection purposes. After this initial organization, we turn our attention to the training data set. Here, we perform a series of preprocessing steps, with word segmentation processing being a key aspect. Using advanced natural language processing techniques, we break down the text data into meaningful words or tokens. This not only simplifies the subsequent analysis but also helps in highlighting the semantic components of the data.

Once the word segmentation is complete, the processed data is then utilized as the input of Word2Vec. Word2Vec, a powerful tool in the field of natural language processing, maps each word to a vector in a high-dimensional space. This vector representation captures the semantic and syntactic relationships between words. As the processed data is fed into Word2Vec, it extracts rich features and forms dense word vectors that encapsulate the essence of the text.

Following this, the constructed convolutional neural network model, which has been carefully designed with multiple layers of convolutional filters and pooling operations, is trained using the word vectors. The training process is a computationally intensive task that requires significant computing resources. However, through continuous optimization and iteration, the model gradually learns the patterns and characteristics associated with SQL injection attacks. Once trained, we obtain the test model, which is a refined version of the initial model, capable of making more accurate predictions.

After that, the relatively complete model is deployed to detect various statements. Whether it's a simple SQL query or a complex database operation statement, the model scrutinizes them all. The detection results are then presented in a highly intuitive percentage format. This percentage not only indicates the likelihood of a statement being a SQL injection attack but also provides a clear measure of the model's efficiency. For example, if the model predicts a 90% probability of a statement being malicious, it gives a strong indication that further investigation is warranted.

Moreover, using the above model, an additional practical application comes into play. You can also input a statement for vehicle inspection. This could potentially be relevant in scenarios where vehicle-related databases are involved, and the need to detect any malicious SQL injections in queries related to vehicle information is crucial. Once the statement is input, the model processes it and returns the result to the terminal in a timely manner, providing immediate feedback to the user.

In the scenario of SQL injection detection, it can convert the words in SQL statements into vectors, which facilitates subsequent processing by machine learning or deep learning models. However, in some cases, it fails to capture the subtle nuances of context. Specifically, there are the following four points:

(1) Fixed word vector representation

Word2Vec learns a fixed vector representation for each word, and this vector does not depend on the specific context in which the word is located. In SQL injection, a word may have completely different meanings and functions in different contexts. For example, the keyword "SELECT" is used for normal operations of retrieving data from the database in a normal SQL query statement. However, in a malicious SQL injection statement, it may be used to construct an illegal data query to bypass the application's security checks. Word2Vec assigns a fixed vector to "SELECT" and cannot dynamically adjust its representation according to the context in the specific SQL statement. Therefore, it is difficult to capture such subtle semantic differences.

(2) Lack of ability to handle long-distance dependencies

In complex SQL statements, there may be long-distance dependencies between words, and these dependencies are very important for determining whether it is an SQL injection. For example, in a nested SQL query or a query with multiple conditions, a certain keyword or condition in the front may affect the semantics of the subsequent part. Word2Vec essentially learns word vectors based on local context information (usually a fixed-size window), and it cannot effectively handle long-distance dependencies. So, when an SQL injection attack bypasses detection by cleverly constructing long-distance semantic associations, Word2Vec has difficulty identifying this attack pattern because it cannot capture such subtle context differences.

(3) Inability to handle polysemy

There are some words or symbols with multiple meanings in the SQL language. For example, the "+" symbol in SQL can be used for both numerical addition operations and, in some cases, string concatenation. Word2Vec cannot distinguish these different meanings according to the specific context and will uniformly represent "+" as a fixed vector. In SQL injection detection, if an attacker constructs a malicious statement using this polysemy feature, Word2Vec cannot accurately understand its semantics in a specific context, thus affecting the detection accuracy.

(4) Inability to adapt to dynamically changing SQL syntax

With the development of database technology and the continuous innovation of attackers' techniques, SQL syntax and injection methods are constantly changing. Word2Vec learns word vectors based on static training data, and once the training is completed, its vector representation is fixed. When new SQL syntax structures or injection methods appear, Word2Vec cannot adapt to these changes in a timely manner, making it difficult to capture the subtle context differences in these new situations and resulting in a decline in the detection ability for new types of SQL injection attacks.

3.2 Feature Extraction Process Based on Content Matching

Due to the fact that in previous research endeavors, scholars predominantly resorted to deep learning techniques for detecting SQL injection attacks by means of syntax parsing and word embedding coding, a series of notable drawbacks emerged. When employing syntax parsing, the intricate syntactic structures of SQL statements were dissected. However, in the process, certain key features that held pivotal significance in accurately discerning malicious intents could inadvertently be concealed. This was primarily because the parsing might focus too narrowly on the grammatical aspects, overlooking the underlying semantic cues that could signal a potential injection attack. Meanwhile, word embedding coding, while attempting to represent words in a vector space to facilitate computational processing, often introduced an unwanted byproduct-noise. The very act of mapping words to vectors could distort or dilute the original semantic essence, leading to a situation where the model was inundated with extraneous or misleading information.

This paper, in a bid to rectify these shortcomings, proposes an innovative and improved SQL injection detection method firmly grounded in content matching. Content matching is a detection method based on rules or templates. It compares the input content by pre-defining a series of rules, patterns, or features. In the context of SQL injection detection, it involves matching the user's input content with the characteristics of known malicious SQL statements, keywords (such as the special usage patterns of "SELECT", "DROP", "INSERT" in malicious constructs), and combinations of dangerous characters. If the input content matches the pre-defined rules, it is determined that there may be an SQL injection attack.

As meticulously illustrated in Table 1, a comprehensive analysis of the diverse array of SQL injection attacks was carried out. By delving deep into the very fabric of these malicious attempts, it became evident that SQL injection attacks frequently manifested with distinct and recognizable content features. For instance, the telltale signs of "or=", a common operator misused in injection scenarios to manipulate query logic; "union", often exploited to combine data from disparate sources in an unauthorized manner; "drop", a dangerously potent command that can obliterate crucial database components like tables; and the quintessential "login" and "password" fields, which are prime targets for attackers seeking to breach authentication mechanisms.

Table 1: List of attack keywords

Attack mode	Attack character
Tautology	Username = 'abc' or 1 = 1 –'pwd = '111'
Federated query	Username = 'abc', pwd = '123' union select count(∗) from users
Compound query	Username = "; drop table users –"pwd = '111'
Logic error	Username = 'abc', pwd = '123'
Inferential formula	Username = 'abc' and 1 = 2 'pwd = '111'
Constructor	Pwd = SYSTEM_USER()

In contrast, when we scrutinize the traditional deep learning approach in the preprocessing phase of feature extraction, a fundamental flaw becomes apparent. Since the traditional method typically adheres to a fixed total number of characters for extraction, it inadvertently truncates or distorts the very information it aims to capture. When dealing with words and characters that are integral to detecting SQL injection, such as those aforementioned crucial terms, the rigid character limit means that the effective information embedded within them is lost. Consider a scenario where a SQL injection attempt cleverly embeds malicious code within a longer string that contains the word "password". If the feature extraction is constrained by a fixed character count, it might chop off the relevant part of the word or fail to fully capture its context, rendering the neural network classifier bereft of the essential cues it needs. Consequently, simply dividing sentences based on characters is an inadequate strategy. It fails to provide the neural network classifier with comprehensive and detailed information that can be gleaned from URL strings. Notably, URL strings serve as the primary vectors for SQL injection attacks.

In SQL injection detection, the content matching method has significant advantages over syntax analysis and word embedding techniques. Syntax analysis requires a deep understanding of the complex syntax rules of SQL and the construction of lexical, syntactic, and semantic analysis processes. It has high technical difficulty and consumes a lot of resources, and is likely to become a performance bottleneck when processing large-scale SQL traffic. In contrast, the content matching method only needs to define simple rules for keywords and character combinations, is easy to implement, has low requirements for technicians, and consumes less resources. In terms of real-time performance, the complex analysis of syntax analysis can cause detection delays, while content matching can respond quickly. Moreover, in the face of various SQL syntax variants and extensions, syntax analysis needs to be customized for different databases, while content matching does not rely on precise syntax analysis and has stronger adaptability. Word embedding technology is based on machine learning to convert words into vector representations. Its decision-making process is complex and difficult to understand intuitively and maintain. The rules of content matching are clear, which is convenient for security personnel to operate. Additionally, word embedding requires a large amount of text data for training to learn semantic relationships. When there are challenges in obtaining labeled data, content matching has low data dependence. Furthermore, the vector calculation and model inference of word embedding have high computational complexity and are time-consuming and resource-intensive when processing large-scale data. Content matching mainly performs string comparison and matching, has low computational complexity, and can complete the detection task quickly.

The improved method uses the feature extraction method in the preprocessing process and the feature atomization of lexical analysis tags to generate more detailed and specific feature vectors of tag sequences through content matching. The preprocessing extraction process is shown in Fig. 2, including data collection, digital generalization, URL replacement, word segmentation, attack keyword processing based on content matching and word vector training.

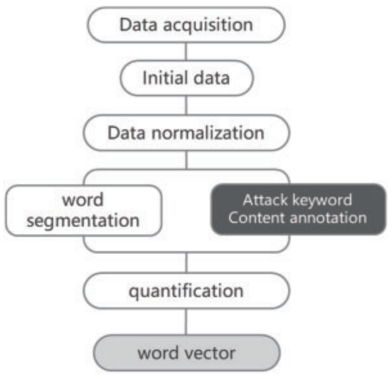

Figure 2: Feature extraction process

The main tasks of data processing include:

(1) Digital generalization

Through URL parsing, when operating on database data in the server, a series of conditional statements need to be carried. for example, http://www.baidu.com/num=10000&id=888888. This statement is a normal query statement, but the input of irrelevant numbers will interfere with the experimental results. Therefore, the number "0" is used instead of the content. The above statement can be generalized as http://www.baidu.com/num=0&id=0.

(2) URL replacement

The general URL statement is as follows: https://www.baidu.com. Simplify and decompose the complex URL format, replacing the domain name with an English letter 'u'. The number in the protocol, domain name, and SQL statement does not affect the essence of the SQL injection statement. Therefore, number generalization and corresponding letter replacement can still retain the main features of the attacked statement, reducing the difficulty of model learning through these two steps and making detection more convenient.

(3) Word segmentation processing

In addition to numbers and "http://" format, regular URLs also contain special characters such as " ", "+", "?", "<>". In order to divide the string sequence, spaces are usually used for segmentation, but among the sequence characters, the association between characters is not large, so it is necessary to retain some keywords and associate them before and after segmentation. Pre-segmentation processing can standardize character formats through normalization, preventing differences in vector representation arising from case-sensitivity. Noise removal enables focus on key SQL semantic information, while synonym replacement reduces interference from diversity Post-segmentation processing can capture semantic relationships between words through word-vector fusion, forming more representative vectors. Feature selection and dimensionality reduction can reduce computational load and highlight key features. Additionally, supplementing context information helps the model better understand the intent of SQL statements.

(4) Attack keyword processing based on content matching

During content matching, in order to ensure that some attack keywords will not lose some effective information, first convert the SQL statement into a feature vector that can be recognized by the neural network. Its embedded matrix representation $\mathrm{S} \rightarrow \mathrm{X} \in \mathrm{R}^{\mathrm{L}\times\mathrm{K}}$. Make the matrix X contain a group of adjacent components x_i $(i = 1, 2, \cdots, L)$, where x_i is the vector representation of characters or words in the URL, and $x_i \in R^K$ is the K-dimension vector.

Common attack keywords include malicious strings and characters, as shown in Table 2. Therefore, dividing URLs according to characters is not enough to get more comprehensive information. Distinguish keywords in Table 2 from other characters as a whole to improve accuracy.

Table 2: List of attack keywords

Type	Attack keywords
Character string	Select, update, insert, delete, declare, exec, drop, create, or, cmd, script, alert
Character	=, ', ;, %, −, +, //

According to the mapping table, assign unique codes to characters and sensitive words to build a coding matrix, as shown in Eq. (1):

$$S' = (u'_1, u'_2, \cdots) \tag{1}$$

u'_i is the encoding of characters or words in the URL. Then the matrix S' is converted into a two-dimensional dense matrix X containing semantic information through the word embedding layer, as the input of the convolution layer, as shown in Eq. (2), x_i is a 64-dimensional column vector.

$$X = (x_1, x_2, \cdots) \tag{2}$$

(5) Word vector training

After the above pretreatment, word segmentation is performed to form a sentence vocabulary dictionary. In order to convert into feature vectors, Word2Vec method is used to train model parameters, and the similarity of word vectors between single words is calculated by cosine similarity. Finally, the calculated word vector is stored [25].

3.3 Optimization of Convolution Neural Network

Deep learning is a type of machine learning technology based on artificial neural networks. It can automatically learn features and patterns from a large amount of data. In SQL injection detection, a deep learning model receives a large amount of normal and malicious SQL input data as training samples. Through the learning and training of a multi-layer neural network, it automatically extracts the deep level features and patterns from the input data, thereby determining whether new input is an SQL injection attack. Among the vast and diverse array of algorithms that constitute the realm of deep learning technology, the convolution neural network truly stands out as one of the most prevalently utilized and quintessentially representative ones. In the context of single-channel CNN classification, a meticulously designed and highly crucial process unfolds. It commences by taking the word vector, which is painstakingly obtained through the embedding of each individual word present in the sentence. This word vector serves as the fundamental input that kick-starts the operation of the CNN. It is then seamlessly transferred to the convolution layer, a core component that holds the key to unlocking hidden patterns and features. Each convolution layer within this intricate architecture employs a convolution kernel, which is, in essence, a precisely defined matrix, often taking the form of a 33 or 55 configuration [26]. These matrices are not just arbitrary arrangements; they are carefully crafted tools that enable the convolution layer to delve deeper into the analysis of each and every convolution kernel within the neural network. By doing so, it aims to extract and synthesize higher-level, more abstract features that encapsulate the essence of the data being processed. This process is akin to a skilled detective sifting through layers of evidence to uncover the most critical clues that would otherwise remain hidden. Deep learning models can learn the complex patterns and features of SQL injection attacks and have a good detection effect on some malicious SQL statements that have been obfuscated or deformed. For example, attackers may use encoding, comments, etc. to bypass simple content-matching rules. However, deep learning models can identify these deformed attack patterns through learning from a large amount of data.

The pooling layer, another integral part of the CNN architecture, plays a distinct yet equally vital role. Its primary objective is to further streamline and optimize the network by reducing the number of nodes in the final full connection layer. This reduction is not a haphazard act; it is a strategic maneuver that serves to curtail the overall parameters in the entire neural network. By minimizing the complexity in this manner, the network becomes more efficient, less prone to overfitting, and better able to generalize its learned patterns to new, unseen data. It's like pruning a complex tree to enhance its overall health and productivity.

The full connectivity layer, as the name suggests, acts as the grand integrator. Its main function revolves around amalgamating the diverse and fragmented local features that have been painstakingly extracted from both the convolution layer and the pooling layer. Through a series of complex computations and connections, it weaves together these disparate pieces of information to form a cohesive and comprehensive overall feature set. This holistic view of the data then paves the way for classification, which is ultimately achieved through the final Softmax output layer. The Softmax layer, with its probabilistic nature, assigns probabilities to each possible class, allowing the network to make informed decisions and predictions with a degree of certainty. In essence, the entire CNN architecture, with its coordinated efforts of each layer, functions like a well-oiled machine, transforming raw data into meaningful insights and classifications.

In order to further optimize the above model, the rule activation function is used in the convolution layer. After each convolution layer, a pool layer with a pool core size of $3 * 3$ is connected. Build three similar structures to classify the output results. The convolution layer is set as Table 3.

Table 3: Details of volume layer settings

Convolution layer	Number of convolution kernels	Convolution kernel size	Convolution mode	Activation function
Convolution layer 1	16	$3 * 3$	SAME	Rule
Convolution layer 2	32	$4 * 4$	SAME	Rule
Convolution layer 3	64	$5 * 5$	SAME	Rule

Two important functions are involved in model building:

(1) Activation Function. As a kind of piecewise linear function, Rule function often uses one-sided consistent operation, that is, when the input value is less than or equal to 0, the output is 0. When the input value is greater than 0, the output value remains unchanged,as shown in Eq. (3). Therefore, the gradient can be maintained without attenuation, which alleviates the problem of gradient disappearance. However, as the rule is trained, some weights cannot be updated, resulting in neuron death.

$$f(x) = \begin{cases} x & if \quad x > 0 \\ 0 & if \quad x \le 0 \end{cases} \tag{3}$$

(2) Classification function. The softmax function can normalize the original data of the upper layer into a value between [0,1], which can be used as a probability distribution to serve as the target prediction value of multiple categories. Softmax function is generally used as the last layer of neural network, which accepts the input value from the upper layer network, and then converts it into probability. The specific function expression is as shown in Eq. (4):

$$S_i = e^{a_i} / \sum_{k=1}^{i} e^{a_k} \tag{4}$$

In Eq. (4): a is an array, a_i is the i-th number in array a, and a_k is the k-th number in array.

To ensure that the model does not overfit, the following measures can be taken. First, use regularization methods such as L1 and L2 regularizations. By adding regularization terms to the loss function, the magnitude of the model parameters is restricted, preventing the model from becoming overly complex and thus reducing the risk of overfitting. Second, perform data augmentation. Carry out operations like rotation, scaling, and adding noise to the training data to increase data diversity. This enables the model to learn a wider range of features and improves its generalization ability. Third, adopt an early-stopping strategy. During the model training process, monitor the performance of the validation set. When the performance of the validation set stops improving, halt the training early to prevent the model from over-learning on the training set. In addition, feature selection can also be carried out. Remove redundant and irrelevant features to reduce the input dimensions of the model and lower the possibility of overfitting.

3.4 Mixing and Improvement of Depth Model

In the cutting-edge research field of RNN, there exists a sophisticated and intricate operating mechanism. Each neuron within the network undertakes the crucial task of meticulously processing the incoming information. Such input information covers a wide range, including continuous text segmented into discrete time steps, time-series data, or audio waveforms and other diverse data forms. After the neuron has absorbed and deeply analyzed the input information, it immediately generates an output result represented as. The unique charm of RNN lies in the seamless flow of information between neurons. This information transfer process is far from random or chaotic. Instead, it resembles a meticulously choreographed and sophisticated program, enabling the network to continuously self-construct and optimize based on the knowledge and insights accumulated over time. Analogously, it's like a scrupulous scholar gradually refining their knowledge system as research unfolds. As new attack data is steadily incorporated into the training set, deep-learning models can continuously learn and evolve, adaptively adjusting their detection strategies. This empowers them to tackle the ever-changing SQL injection attack methods and exhibit strong generalization capabilities.

In fact, a deep analysis of the essence of the RNN network architecture reveals that there is no fundamental difference between it and the traditional ordinary neural network. It can be regarded as an advanced version of the ordinary neural network, constructed by adding several special modules. These modules serve as the key channels for information conduction, ensuring that information can be transmitted orderly from one neuron to the next one, and continuously between different time steps. This endows RNN with certain advantages, enabling it to exhibit outstanding pattern-capturing capabilities when dealing with tasks such as speech recognition with a small number of words or sentence recognition scenarios with concise text content. It can accurately extract the inherent internal patterns and the dependencies among data in short-sequence data, and deeply understand the context information through the recurrent connection mechanism.

However, in the process of technological development, as real-world application scenarios become increasingly complex, limitations have emerged. When RNN is applied to actual situations where the span of information before and after continues to increase, its performance shortcomings become prominent, and its once remarkable performance shows a downward trend. This bottleneck severely restricts the further promotion and wide application of RNN. The root cause lies in the fact that RNN has difficulty maintaining effective long-term memory during the processing of long sequences and cannot accurately capture the complex dependencies among long-span information. As the information gap gradually widens, the utilization efficiency of RNN for past context information is greatly reduced, resulting in a decrease in prediction accuracy.

Fortunately, there is currently a series of strategies to alleviate the above difficulties. By organically integrating the advantages of CNN and RNN in text classification tasks, it is expected to open up a new situation for optimizing model performance. In the model training stage, a key operation is to finely adjust the input word vectors. That is, according to strict linguistic rules and data characteristics, carefully calibrate the vector representation forms of words, striving to accurately capture the subtle semantic and syntactic differences contained in words, so as to help the model deepen its understanding of the processed text. In addition, with its unique convolutional kernel structure, CNN can slide and scan on text data like a precise detector, efficiently extracting spatial features and local short-term features. After refinement and concentration, the output feature vector of the CNN pooling layer can serve as the core input data source for the LSTM, laying a solid foundation for subsequent in-depth processing.

LSTM stands out in the field of deep learning with its innovative memory cell structure, replacing the traditional hidden layer architecture of ordinary neural networks. As shown in Fig. 3, the core key to the performance of LSTM lies in the internal state management of each neuron. The information transfer process between neurons is precisely regulated by three unique gating mechanisms embedded within the network [27]. These three gates, namely the input gate, the output gate, and the forgetting gate, are by no means simple binary switches; instead, they are composed of a series of strictly calibrated complex functions. Each function equation accurately reflects the different states of the network, finely determining the rules for retaining, discarding, and incorporating new knowledge.

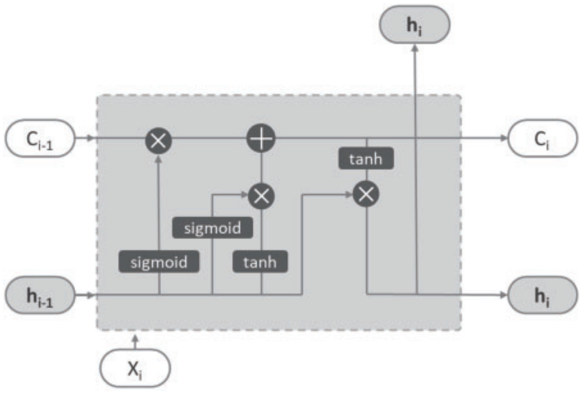

Figure 3: LSTM network structure

Specifically, this process begins with the generation of the vector inside the neural unit. This vector is an organic fusion product of the current input information and the existing internal state of the neuron. Subsequently, it interacts and combines with the historical information stored in the forgetting gate. The forgetting gate is like an intelligent filter, making a precise judgment based on the current real-time situation to selectively allow or block the transfer of past memory information. Once the fusion step is completed, a new state will be generated inside the input gate, which integrates the key relevant information of the previous moment. At the same time, the input gate determines whether to incorporate new information into the unit memory according to the built-in rules. Finally, the output gate is like a searchlight focusing strong light, accurately locking onto the most valuable information, outputting the current state, and orderly transmitting it to the next neuron or time step, driving the continuous operation of the entire network. This approach is intended to enhance the comprehensiveness and accuracy of the detection process, contributing to the development of a more robust and reliable security solution in the context of data-driven threat identification.

The process begins with the generation of the vector within the neural unit. This vector is a culmination of the current input and the internal state of the neuron. It then combines with the information held within the forgetting gate. This gate acts as a sieve, selectively allowing or blocking the passage of past memories based on the current context. Once this combination occurs, a new state, which encapsulates the relevant information from the previous moment, is generated in the input gate. This gate decides whether new information should be added to the cell's memory. Finally, the output gate, like a spotlight, focuses on the most pertinent information and outputs the current state, which is then passed on to the next neuron or time step.

This paper ventures further into the uncharted territory of optimizing the deep learning model. The proposed approach is to first utilize the CNN network to its fullest potential. By deftly extracting spatial features and local short-term features, it lays a solid foundation. The output feature vector of the CNN pooling layer, rich in distilled information, is then channeled as the input for the LSTM. The LSTM, in turn, takes over the reins and delves into learning long-term global temporal features. It's like a historian piecing together a chronicle of events, connecting the dots over long periods. This combination, known as the CNN-LSTM model, is a powerhouse. The word vector of each word in the sentence, after undergoing the transformation of word embedding, is fed into the CNN. The features extracted by the CNN are then passed on to the LSTM as its nourishment. Finally, through the full connection layer, which acts as a grand integrator, and the Softmax layer, which assigns probabilities to different classes, the ultimate classification result is achieved. This end-to-end process transforms raw text into meaningful insights, paving the way for more accurate predictions and a deeper understanding of sequential data.

3.5 Applicability of the Content Matching and Deep Learning Methods

The applicability of the methods based on Content Matching and Deep Learning varies when dealing with eight common types of cyber-attacks, namely SQL injection, cross-site scripting (XSS), denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks, brute-force attacks, man-in-the-middle (MitM) attacks, cross-site request forgery (CSRF), password reset attacks, and file inclusion attacks. The following is a detailed analysis:

(1) SQL injection attacks

- Content Matching method: Highly applicable. SQL injection attacks typically contain specific SQL keywords (such as SELECT, DROP, UNION, etc.) and combinations of dangerous characters (such as single quotes ' and semicolons ;). By pre-defining these rules and patterns, the input content from users can be matched. Once a match is found, it can be determined that there may be an SQL injection attack. This method is simple and direct, capable of quickly detecting common SQL injection patterns.
- Deep Learning method: Also applicable. Deep learning models can learn the features and patterns of a large number of normal and malicious SQL statements. They can achieve good detection results even for obfuscated and deformed SQL injection statements. These models can automatically extract deep-level features from data and discover potential attack patterns that are difficult for humans to identify.

(2) Cross-site scripting (XSS) attacks

- Content Matching method: Relatively applicable. XSS attacks usually contain specific HTML tags (such as <script>) and JavaScript code snippets. By pre-defining these rules and matching the user input content, most obvious XSS attacks can be detected. However, for some encoded and deformed XSS attacks, the content matching method may fail.
- Deep Learning method: Applicable. Deep learning models can learn the complex patterns and features of XSS attacks and handle encoded and deformed XSS attacks. They can learn potential attack patterns from a large amount of normal and malicious HTML and JavaScript code, thereby improving the detection accuracy.

(3) Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks

- Content Matching method: Partially applicable. For some DoS/DDoS attacks based on specific protocol features, such as UDP flood attacks, they can be detected by matching specific UDP packet features (such as port numbers and packet sizes). However, for some complex DDoS attacks that use multiple protocols or forge source IP addresses, the content matching method may not work well.
- Deep Learning method: Applicable. Deep learning models can learn the normal and abnormal patterns of network traffic. Through learning a large amount of network traffic data, they can identify traffic anomalies caused by DDoS attacks. These models can handle complex network traffic features and have good detection capabilities for different types of DDoS attacks.

(4) Brute-force attacks

- Content Matching method: Not very applicable. Brute-force attacks mainly involve continuously trying different password combinations, making it difficult to match through pre-defined rules and patterns. Since the password combinations used by attackers are random, there are no obvious features for direct matching.
- Deep Learning method: Somewhat applicable. Deep learning models can learn the patterns of normal user login behaviors, such as login time and login frequency. By learning a large amount of login data, when there is an abnormal login attempt frequency, the model can identify a possible brute-force attack. However, the accuracy of this method may be affected by factors such as changes in users' normal login habits.

(5) Man-in-the-middle (MitM) attacks

- Content Matching method: Not very applicable. MitM attacks mainly involve intercepting and tampering with communication data, making it difficult to match through pre-defined rules and patterns. Attackers can carry out attacks without changing the data format, making it hard for content matching to detect these attacks.
- Deep Learning method: Somewhat applicable. Deep learning models can learn the features and patterns of normal communication data. Through learning a large amount of communication data, when there are abnormal communication behaviors (such as abnormal data transmission paths or abnormal data content), the model can identify a possible MitM attack. However, this method requires a large amount of normal and abnormal communication data for training, and it is still difficult to detect some advanced MitM attacks.

(6) Cross-site request forgery (CSRF) attacks

- Content Matching method: Not very applicable. CSRF attacks involve using users' identity information to initiate unintended requests on target websites, making it difficult to match through pre-defined rules and patterns. Attackers can construct normal request formats and only take advantage of users' identities, making it hard for content matching to detect these attacks.
- Deep Learning method: Somewhat applicable. Deep learning models can learn the features and patterns of normal user requests. By learning a large amount of user request data, when there are abnormal request behaviors (such as abnormal request sources or abnormal request frequencies), the model can identify a possible CSRF attack. However, this method needs to consider the diversity of users' normal request behaviors, and the detection accuracy may be affected to some extent.

(7) Password reset attacks

- Content Matching method: Not very applicable. Password reset attacks mainly involve bypassing the verification mechanism of the password reset process, making it difficult to match through pre-defined rules and patterns. Attackers can take advantage of various vulnerabilities and means, and there are no obvious features for direct matching.

- Deep Learning method: Somewhat applicable. Deep learning models can learn the features and patterns of the normal password reset process. By learning a large amount of password reset data, when there are abnormal password reset behaviors (such as abnormal verification information or abnormal request sources), the model can identify a possible password reset attack. However, this method requires a large amount of normal and abnormal password reset data for training, and it is still difficult to detect some new-type password reset attacks.

(8) File inclusion attacks

- Content Matching method: Relatively applicable. File inclusion attacks usually contain specific file paths, file names, and other features. By pre-defining these rules and matching the user input content, most obvious file inclusion attacks can be detected. However, for some deformed and encoded file inclusion attacks, the content matching method may fail.
- Deep Learning method: Applicable. Deep learning models can learn the complex patterns and features of file inclusion attacks and handle deformed and encoded file inclusion attacks. They can learn potential attack patterns from a large amount of normal and malicious file inclusion requests, thereby improving the detection accuracy.

In summary, the content matching method has good detection results for attacks with obvious rules and patterns (such as SQL injection, XSS, and file inclusion attacks), but it performs poorly for complex and changeable attacks that are difficult to match through rules (such as brute-force attacks, MitM attacks, CSRF, and password reset attacks). The deep learning method has stronger adaptability and generalization ability and can handle various complex attack patterns, but it requires a large amount of training data and high-performance computing resources. In practical applications, these two methods can be combined to improve the accuracy and efficiency of cyber-attack detection.

4 Simulation Experiment of SQL Injection Detection for Deep Learning

4.1 Data Feature Processing

In the main process of the experiment, the data is first sorted to form a dataset for training, verification and detection. Before training, it is necessary to collect enough normal statement data and attack class statements as sample data. The data set used in this experiment is from the data shared mainly from: https://github.com/client9/libinjection/ (accessed on 01 January 2025) and using Wireshark's HTTP protocol to capture websites, intercepting datasets composed of various web payloads from GET and POST requests. This algorithm can be applied to real web application environments for issues such as SQL injection detection. In order to accelerate the running speed of the model and reduce the consumption of computing resources, we have adopted a strategy: extracting data packets containing keywords and importing them into the model for recognition and detection. In order to improve the contrast between positive and negative samples during the subsequent model training process, this paper removed normal scripts without keywords to obtain pure normal original samples. By removing irrelevant data, we can better distinguish between normal and abnormal behavior, thereby improving the accuracy and robustness of the model.

Table 4 presents the statement data set, which is partitioned into three distinct subsets: the training data set, the validation data set, and the test data set. This partitioning is crucial for the development and evaluation of models in the context of statement classification, especially for detecting malicious statements like SQL injection and XSS. The data set encompasses three types of statements: normal statements, SQL injection statements, and XSS statements. For normal statements, there are 24,000 samples in the training data set, which serves as the foundation for a model to learn the patterns and characteristics of legitimate statements. The validation data set contains 10,000 normal statements, and it is used to fine-tune the model's hyperparameters and prevent overfitting. The test data set has 4000 normal statements, providing an unbiased evaluation of the model's performance on unseen normal data. Regarding SQL injection statements, the training data set consists of 25,000 samples. These data help the model recognize the malicious patterns and techniques used in SQL injection attacks. Similar to the normal statements, there are 10,000 samples in the validation data set and 4000 samples in the test data set for assessing the model's ability to detect SQL injection attacks accurately.

Table 4: Statement data set

Statement type	Training data set	Validation data set	Test data set
Normal statement	24,000	10,000	4000
SQL injection statement	25,000	10,000	4000
XSS statement	25,000	10,000	4000

In this experiment, we employ the high-level Python programming language, the TensorFlow deep learning framework, and the Androguard Android app reverse engineering tool. The experimental platform's hardware configuration consists of an Intel Core i5 12400 processor, 32 GB of DDR4 3200 MHz memory, and an NVIDIA GeForce GTX 1080Ti with 11 GB of memory. On the software side, we use Windows 11 Professional Edition 22H2, the NVIDIA (R) Cuda compilation tools (release 11.8, V11.8.89), Cudnn V8.8.0, Python 3.9.13, TensorFlow gpu2.10.0, and Android 3.3.5. The training data sets are processed to reduce the complexity of the sentences in the training data sets. Complex statements will increase the difficulty of algorithm implementation and reduce the effect of model training, so proper data processing is very necessary in model training. The sample data used for training and detection are decoded statements, as shown in the Table 5.

Table 5: Statement data set encoding

Statement type	Unhandled query statement	Sample data after decoding
Normal statement	id=240&version=62&num=14985	id%3D240%26version%3D62%26num %3D14985
SQL injection statement	A" WHERE 1 = 1 OR NOT (1 = 0)	A%22%20WHERE%201%3D1%20OR% 20NOT%20%281%3D0%29
XSS statement	<script>alert("HELLO")</script>	%3Cscript%3Ealert(%22HELLO%22)%3 C/script%3E

4.2 Test Results of Deep Learning Experiment

The most important part of detecting different sentences based on CNN is the training of the model. When training, Epoch is used to represent the number of training. We can visualize an Epoch as a complete data set and pass it forward and backward in the artificial neural network once, that is, pass and return it once in the neural network. As the number of epochs increases, the number of updates to weights in the neural network also increases, and the curve changes from underfitting to overfitting. Take the training of SQL injection dataset as an example, the measurement results are shown in Fig. 4.

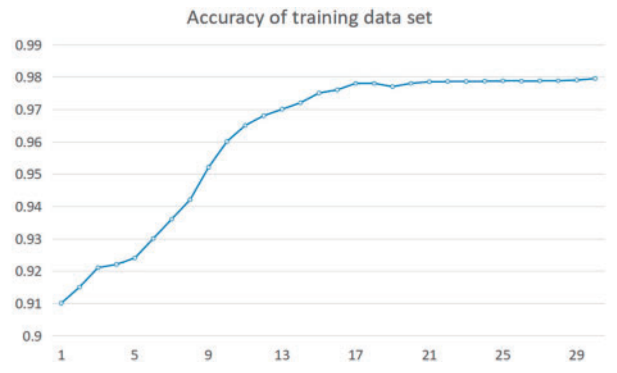

Figure 4: Accuracy of training data set

Through the accuracy of the training data set, we can find that increasing the number of epochs within a certain range can improve the accuracy of the training dataset. Finally, Epoch = 30 is selected as the number of CNN model training. After the model training is completed, use the trained model to test the test dataset, the measurement results are shown in the Table 6.

Table 6: Test results

Statement type	Test type	Normal	sql	xss	Time
Normal statement	Test rate	0.97955	0.0185	0.0075	11.73 s
	Test results	3903	74	23	
SQL injection statement	Test rate	0.01925	0.0025	0.97825	12.25 s
	Test results	77	10	3913	
XSS statement	Test rate	0.0185	0.97975	0.00175	14.09 s
	Test results	3919	74	7	

4.3 Detection Effect of Three Models

In order to verify the effectiveness of various model training segmentation detection methods, the experiment uses accuracy, precision, recall and F1 values as evaluation indicators. Relevant parameters are shown in the Table 7.

Table 7: Test index parameters

Name	Number of convolution kernels
TP	Correct number of matches
FP	False alarm, no matching is incorrect
FN	Misreported, no correct matching number found
TN	Correct number of mismatches
Accuracy	(TP + TN)/(TP + FP + TN + FN)
Recall	TP/(TP + FN)
F1	2 · Precision · Recall/(Precision + Recall)

The model is trained by using three different segmentation methods for data, namely CNN model, CNN-RNN model and CNN-LSTM model, and their detection effect on the test set is observed, as shown in Fig. 5.

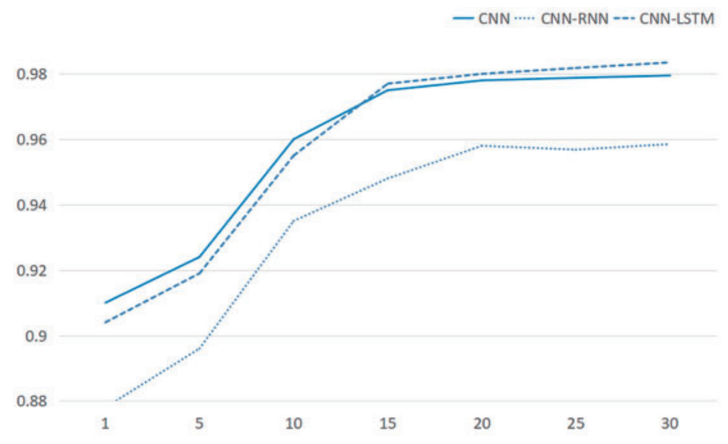

Figure 5: Final test results of all models on the test set

According to Table 8, in the SQL statement data set, the accuracy rate of the CNN detection model is 97.8%, and that of the CNN-LSTM is 98.3%. Both of them have achieved a high accuracy rate. The CNN model has good comprehensive performance. Its accuracy rate is 0.9783, the recall rate is 0.9775, and the F1 score is 0.9789. The indicators are balanced and at a high level. The testing time is 11.73 s, with high calculation efficiency. The CNN-RNN model has an accuracy rate of 0.95802, a recall rate of 0.9594, and an F1 score of 0.9550, all of which are lower than those of the CNN model. The testing time is 16.98 s, with high calculation complexity. The CNN-LSTM model performs excellently. Its accuracy rate is as high as 0.98352, the recall rate is 0.9804, and the F1 score is 0.9886. It is the highest among the three models. The testing time is 14.43 s, achieving a good balance between performance and efficiency. The reason may be that the long time of feature fusion between RNN and CNN and the small size of the data set lead to the slightly lower accuracy rate of the CNN-RNN. When processing data features, the attack keywords can be processed separately to achieve a higher accuracy rate. By combining the indicators of accuracy rate, recall rate and F1 value, the CNN-LSTM can obtain more sufficient features to achieve the highest detection accuracy rate, etc.

Table 8: Final test results of all models on test set

Detection model	Accuracy	Recall	F1	Test time
CNN	0.9783	0.9775	0.9789	11.73 s
CNN-RNN	0.95802	0.9594	0.9550	16.98 s
CNN-LSTM	0.98352	0.9804	0.9886	14.43 s

Finally, CNN-LSTM model is used to observe the detection effect on the test set under the condition of using content matching to mark the data set and directly extracting the original data, as shown in Table 9.

Table 9: Final test results of CNN-LSTM models on test set

CNN-LSTM	Accuracy	Recall	F1	Test time
Feature extraction method for content matching mark	0.98352	0.9804	0.9886	14.43 s
Common feature extraction method	0.96151	0.9672	0.9713	10.43 s

According to Table 9, in the SQL statement dataset, the adoption of the content-matching-based extraction method in the data processing module leads to a relatively long testing time during the experiment. When using the "Feature extraction method for content matching mark", its accuracy rate reaches 0.98352. Table 9 shows that the accuracy rate of this method in the SQL statement dataset is 98.3%. The recall rate is 0.9804, and the F1 score is as high as 0.9886. These indicators demonstrate that this method performs excellently in terms of prediction accuracy, positive-sample recognition ability, and the balance between precision and recall. However, the testing time of this method is 14.43 s, which is relatively long. The accuracy rate of the "Common feature extraction method" is 0.96151. Table 9 shows that its accuracy rate is 96.2%. The recall rate is 0.9672, and the F1 score is 0.9713. Although all the indicators of this method are lower than those of the feature extraction method for content matching mark, the gaps are not very significant. Moreover, its testing time is only 10.43 s, giving it an obvious advantage in computational efficiency. Considering the accuracy rate, recall rate, and F1 score comprehensively, the feature extraction method for content matching mark yields good results, and there is still room for further improvement in its accuracy rate.

Different models need to be selected according to different practical application scenarios. For scenarios with extremely high performance requirements, where more emphasis is placed on comprehensive indicators and the pursuit of the lowest possible false-positive and false-negative rates, such as financial transaction security detection, the CNN-LSTM model is the first choice. Its excellent accuracy, recall rate, and F1 score can provide reliable detection results. If the application scenario has strict requirements for real-time performance, such as real-time network intrusion detection, the CNN model is more suitable. Its short testing time can meet the need for quick response, and its performance is also considerable. When the data has obvious sequential characteristics and the time requirement is not particularly strict, the CNN-RNN model can be considered, but its relatively low performance and long testing time need to be weighed. When applied to large scale datasets, the proposed method faces challenges such as high computational resource requirements, low data processing efficiency, and limited model scalability. For example, it may lead to long training times and difficult parameter tuning. In the case of dynamic datasets, the method needs real time processing and model updating capabilities to adapt to continuous data changes. It also has to address issues like concept drift and maintaining data consistency and quality.

Due to some contextual or implicit relationships, there will indeed be differences in the recognition effects of certain specific types of SQL injection attacks or XSS patterns. On the one hand, we should consider the generalization ability of the model and conduct experimental analyses using different datasets. On the other hand, we should specifically analyze the SQL samples with poor recognition effects and optimize the algorithm's performance.

5 Summary

SQL exhibits characteristics such as low implementation complexity, flexible statement construction, and high threat potential, making it a favored attack method among hackers. To conduct SQL injection detection using deep learning, a content-matching approach is employed for word segmentation, safeguarding against the loss of crucial information. Building on the research of convolutional neural networks, the Word2Vec tool extracts features from the dataset, generating word vectors. These vectors are enhanced with effective information by embedding them with key attack strings and characters. Subsequently, the constructed convolutional neural network models-CNN, CNN-RNN, and CNN-LSTM-are trained. During training, parameters are continuously updated to adjust weights, resulting in a test-ready model. This model analyzes the intent of SQL statements to identify potential attacks. Among these models, CNN-LSTM proves most effective, capturing more comprehensive features and achieving the highest detection accuracy. Looking ahead, solutions leveraging large-language models like GPT will be explored. Notably, the advent of DeepSeek has introduced novel perspectives. Additionally, evaluating this method on larger and more diverse datasets is essential to ensure its robustness and generality. Research into handling obfuscated SQL injection attacks and those employing advanced evasion techniques is not only crucial but also pressing.

Acknowledgement: Thank you to all those who have contributed to this manuscript.

Funding Statement: The work is supported by Jiangsu Higher Education "Qinglan Project", an Open Project of Criminal Inspection Laboratory in Key Laboratories of Sichuan Provincial Universities (2023YB03), Major Project of Basic Science (Natural Science) Research in Higher Education Institutions in Jiangsu Province (23KJA520004), Jiangsu Higher Education Philosophy and Social Sciences Research General Project (2023SJYB0467), Action Plan of the National Engineering Research Center for Cybersecurity Level Protection and Security Technology (KJ-24-004), Jiangsu Province Degree and Postgraduate Education and Teaching Reform Project (JGKT24_B036), and Digital Forensics Engineering Research Center of the Ministry of Education Open Project (DF20-010). Teaching Practice of Web Development and Security Testing under the Background of Industry University Cooperation (241205403122215); Research on Strategies for Combating and Preventing Virtual Currency Telecommunications Fraud (2024SJYB0344).

Author Contributions: Yuqi Chen and Guangjun Liang conceived and designed the experiments; Yuqi Chen and Guangjun Liang performed the experiments; Guangjun Liang and Qun Wang analyzed the data; Guangjun Liang and Yuqi Chen wrote the paper. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data available on request from the corresponding author.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Zhang R, Zhu F, Liu J, Liu G. Depth-wise separable convolutions and multi-level pooling for an efficient spatial CNN-based steganalysis. IEEE Trans Inf Forensics Secur. 2019;15:1138–50. doi:10.1109/TIFS.2019.2936913.
2. Xu R, Zhang N, Li M, Li Z. Research on a high robust detection model for malicious software. Netinfo Secur. 2024;24:1184–95 (In Chinese).
3. Bobade ND, Sinha VA, Sherekar SS. A diligent survey of SQL injection attacks, detection and evaluation of mitigation techniques. In: 2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS); 2024 Feb 24–25; Bhopal, India. p. 1–5. doi:10.1109/SCEECS61402.2024.10481914.
4. Anu P, Ramani G, Mohanapriya D, Ganesh RK, Kalyani N. Mitigation of SQL injection attacks through machine learning classifier. In: 2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS); 2024 Jul 10–12; Coimbatore, India. p. 606–11. doi:10.1109/ICSCSS60660.2024.10625626.
5. Mohanraj A, Madhan Kumar N, Nikhilesh S, Sanjay GS, Sanjay N. Shielding web data: mitigating SQL injection threats via hashing strategies. In: 2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS); 2024 Mar 14–15; Coimbatore, India. p. 2602–7. doi:10.1109/ICACCS60874.2024.10717176.
6. Okesola JO, Ogunbanwo AS, Owoade A, Olorunnisola EO, Okokpuji K. Securing web applications against SQL injection attacks—a Parameterised Query perspective. In: 2023 International Conference on Science, Engineering and Business for Sustainable Development Goals (SEB-SDG); 2023 Apr 5–7; Omu-Aran, Nigeria. doi:10.1109/SEB-SDG57117.2023.10124613.

7. Abdullah AS, Mohapatra P. Detection and analysis of port scanning and SQL injection vulnerabilities with correlating factors in web applications to enhance secure data transmission. In: 2023 International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE); 2023 Nov 1–2; Chennai, India. doi:10.1109/RMKMATE59243.2023.10368777.
8. Guan Y, Zhou W, Wang H, Lin L. Feature fusion-based detection of SQL injection and XSS attacks. In: 2024 5th International Conference on Information Science, Parallel and Distributed Systems (ISPDS); 2024 May 31–Jun 2; Guangzhou, China. doi:10.1109/ISPDS62779.2024.10667632.
9. Zhang M, Xu B, Bai S, Lu S, Lin Z. A deep learning method to detect web attacks using a specially designed CNN. In: Neural Information Processing: 24th International Conference; 2017 Nov 14–18; Guangzhou, China. p. 828–36.
10. Gandhi N, Patel J, Sisodiya R, Doshi N, Mishra S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In: 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE); 2021 Mar 17–18; Dubai, United Arab Emirates. p. 378–83. doi:10.1109/ICCIKE51210.2021.9410675.
11. Bouafia R, Benbrahim H, Amine A. Automatic protection of web applications against SQL injections: an approach based on acunetix, burp suite and SQLMAP. In: 2023 9th International Conference on Optimization and Applications (ICOA); 2023 Oct 5–6; Abu Dhabi, United Arab Emirates. doi:10.1109/ICOA58279.2023.10308827.
12. Ray D, Ligatti J. Defining code-injection attacks. SIGPLAN Not. 2012;47(1):179–90. doi:10.1145/2103621.2103678.
13. Varshney K, Ujjwal RL. LsSQLIDP: literature survey on SQL injection detection and prevention techniques. J Stat Manag Syst. 2019;22(2):257–69. doi:10.1080/09720510.2019.1580904.
14. Thalji N, Raza A, Islam MS, Abdel Samee N, Jamjoom MM. AE-net: novel autoencoder-based deep features for SQL injection attack detection. IEEE Access. 2023;11(3):135507–16. doi:10.1109/ACCESS.2023.3337645.
15. Cao X. Research on SQL injection detection based on deep learning [dissertation]. Nanning, China: Guangxi University; 2020. (In Chinese).
16. Li Q, Wang F, Wang J, Li W. LSTM-based SQL injection detection method for intelligent transportation system. IEEE Trans Veh Technol. 2019;1. doi:10.1109/TVT.2019.2893675.
17. Kakisim AG. A deep learning approach based on multi-view consensus for SQL injection detection. Int J Inf Secur. 2024;23(2):1541–56. doi:10.1007/s10207-023-00791-y.
18. Liu Y, Dai Y. Deep learning in cybersecurity: a hybrid BERT-LSTM network for SQL injection attack detection. IET Inf Secur. 2024;2024(1):5565950. doi:10.1049/2024/5565950.
19. Salah Fathi K, Barakat S, Rezk A. An effective SQL injection detection model using LSTM for imbalanced datasets. Comput Secur. 2025;153(15):104391. doi:10.1016/j.cose.2025.104391.
20. Zulu J, Han B, Alsmadi I, Liang G. Enhancing machine learning based SQL injection detection using contextualized word embedding. In: Proceedings of the 2024 ACM Southeast Conference; 2024 Apr 18–20; Marietta, GA, USA. p. 211–6. doi:10.1145/3603287.3651187.
21. Souza MS, Ribeiro SESB, Lima VC, Cardoso FJ, Gomes RL. Combining regular expressions and machine learning for SQL injection detection in urban computing. J Internet Serv Appl. 2024;15(1):103–11. doi:10.5753/jisa.2024.3799.
22. Lakhani S, Yadav A, Singh V. Detecting SQL injection attack using natural language processing. In: 2022 IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON); 2022 Dec 2–4; Prayagraj, India. doi:10.1109/UPCON56432.2022.9986458.
23. Marinelli R. Causal tracing to identify hacking knowledge in large language models. In: 2024 IEEE International Conference on Big Data (BigData); 2024 Dec 15–18; Washington, DC, USA. p. 8774–6. doi:10.1109/BigData62323.2024.10825125.
24. Huang KJ, Wang J, Chen JY. A large language model based SQL injection attack detection method. Netinfo Security. 2023;23(11):84–93. (In Chinese).
25. Khatua A, Khatua A, Cambria E. A tale of two epidemics: contextual Word2Vec for classifying twitter streams during outbreaks. Inf Process Manag. 2019;56(1):247–57. doi:10.1016/j.ipm.2018.10.010.
26. Zhang W, Li Y, Li X, Shao M, Mi Y, Zhang H, et al. Deep Neural Network-Based SQL injection detection method. Secur Commun Netw; 2022;2022(1):4836289.
27. Zheng ZY, Gu SY. TensorFlow: practical google deep learning framework. Beijing, China: Electronic Industry Press; 2017. p. 141–5. (In Chinese).