The Journal of Supercomputing (2025) 81:608
https://doi.org/10.1007/s11227-025-07109-w

Sql injection detection algorithm based on Bi-LSTM and integrated feature selection

Qiurong Qin[1] · Yueqin Li[1] · Yajie Mi[1] · Jinhui Shen[1] · Kexin Wu[1] · Zhenzhao Wang[1]

Accepted: 21 February 2025
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2025

Abstract

SQL injection attacks represent a significant web security threat. However, due to their diversity and variability, existing detection methods often suffer from high false alarm rates and low accuracy. To address these challenges, this paper proposes an efficient and lightweight SQL injection detection model, SQLLS, based on a bidirectional long short-term memory network. Initially, the term frequency-inverse document frequency algorithm is employed to convert SQL statements into numerical feature vectors, enabling the extraction of key information and enhancing the model's ability to characterize the input data. Subsequently, an integrated feature selection method, GFC, is presented, which combines multiple techniques to improve both the accuracy and robustness of feature selection. Specifically, gradient boosting regression trees are used to evaluate the importance of each feature, identifying those most significant for classification; Fisher score filters out features that can effectively distinguish between SQL injections and non-injections based on statistical significance; and the chi-square test further evaluates the relevance of the features with respect to the target label, ensuring that the selected features are highly correlated with SQL injection detection. After feature selection, a mixed precision training technique is utilized to reduce memory consumption and enhance training efficiency. To reduce the complexity of the bidirectional long short-term memory model and improve its computational efficiency, this paper introduces a pruning technique that minimizes computational overhead by removing unimportant weight connections, thereby improving the model's operational efficiency. Experimental results demonstrate that the SQLLS model achieves an accuracy of 100%, a low false alarm rate of 0.154%, and significantly shorter running times compared to existing models.

Keywords SQL injection · TF-IDF algorithm · Integrated feature selection · Bi-LSTM network

Extended author information available on the last page of the article

Published online: 12 March 2025

1 Introduction

Cyber-attacks cost the economy nearly $50 billion annually, with SQL injection attacks accounting for more than one-fifth of that total [1–4]. Therefore, accurate detection of SQL injection attacks is an important task in the field of network security [5–7].

Existing detection methods for SQL injection are typically categorized into three main groups: traditional rule-based approaches [8–11], traditional machine learning-based methods [12], and deep learning-based techniques [5–7, 13]. While traditional methods can be effective in certain scenarios, they often struggle with dynamically generated SQL statements [14] and face difficulties in addressing the increasing sophistication of injection attacks [15, 16]. Traditional machine learning-based methods, though robust [17], encounter significant limitations in feature engineering, large-scale dataset management, and handling complex attack patterns [18, 19].

In response to these challenges, deep learning-based SQL injection detection methods have gained significant attention in recent years [20, 21]. Alghawazi et al. [22] proposed a recurrent neural network-based detection method to address the complexity of attack patterns, which was evaluated using the Kaggle dataset, achieving an accuracy of 94% and an F1 score of 92%. However, its performance still lags behind state-of-the-art approaches. Zhang et al. [23] introduced an SQL injection detection model based on deep neural networks, which achieved 96% precision using word vector representations and a multilayer neural network structure, combined with ReLU and Dropout techniques. However, the model was only tested on a single dataset, and its robustness needs further validation. A Paul et al. [24] proposed a hybrid CNN-LSTM model that achieved high accuracy in SQL injection detection but did not incorporate feature selection, potentially introducing noise that could affect performance. In addition, the model employs a multilayer convolutional and LSTM structure, which enhances accuracy. However, due to its complexity, the training process requires substantial computational resources. This issue is particularly pronounced when applied to large-scale datasets, where the model may encounter high computational time and significant cost. Overall, while these deep learning-based approaches have made progress in improving the accuracy of SQL injection detection [25], they continue to face challenges in terms of performance and computational efficiency.

To address the above challenges, this paper presents the SQLLS detection model based on a bidirectional long short-term memory (Bi-LSTM) network. The model begins by extracting features from SQL statements using the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm, effectively capturing key information. Next, the integrated feature selection method GFC is employed to identify the most influential features for the classification task. Meanwhile, mixed precision is employed to train the SQLLS model, reducing memory consumption during training. Finally, to enhance computational efficiency, this paper minimizes computational overhead by removing unimportant weights through the pruning technique. The contributions of this paper are as follows:

(1) An efficient and lightweight SQL injection detection model based on a Bi-LSTM network: This paper proposes a SQL injection detection model that leverages the Bi-LSTM network to capture nonlinear relationships in SQL statements through deep learning, enabling the accurate identification of complex SQL injection attack patterns. Compared to traditional deep learning models, this model achieves high accuracy while maintaining a lightweight design by incorporating pruning techniques. These techniques significantly reduce computational and memory overhead, making the model more suitable for resource-constrained environments. According to our survey, most people tend to focus on the performance of the model when researching cybersecurity and seldom consider the lightness of the model.
(2) Innovative integrated feature selection method, GFC: This paper introduces an integrated feature selection strategy, GFC, which combines three feature selection methods: Gradient boosting regression trees (GBRT), Fisher score, and the chi-square test. GBRT evaluates feature importance and identifies key features by integrating decision trees(DT). Fisher score selects features that can effectively distinguish between SQL injection and non-SQL injection based on statistical significance. The chi-square test measures the correlation between features and target labels to ensure the selected features are highly relevant to the classification task. These methods complement each other, collectively improving the accuracy and robustness of feature selection.
(3) Cross-dataset validation and construction of an XSS dataset: To comprehensively evaluate the performance of the proposed model, this paper constructs a dedicated dataset for XSS attacks. This additional dataset is specifically designed to assess the model's adaptability and practicality in handling diverse attack scenarios.

2 Analyzing the characteristics of SQL injection statements

2.1 SQL injection and its detection

SQL injection is a malicious technique [26] that exploits vulnerabilities in applications that fail to properly validate, filter, or escape user input, enabling attackers to execute harmful SQL queries on the application's database [27]. By injecting malicious SQL code, attackers can deceive the database system into performing unauthorized operations, thereby bypassing authorization and access controls and retrieving unauthorized data that may lead to database modification. Currently, several major types of SQL injection attacks exist, including Authentication Bypass and Blind SQL Injection. Each type of attack can have severe consequences, potentially causing significant harm to users, applications, and databases. Therefore, detecting SQL injection attacks has become a critical issue that must be addressed urgently in the field of network security.

SQL injection detection is a classification problem aimed at distinguishing between SQL-injected statements and non-SQL-injected statements within a dataset. It can also be refined to classify specific types of SQL injection attacks. The

core objective is to accurately predict potential threats by identifying malicious SQL injection statements. Through classifier training, the model learns the features and patterns of SQL injection attacks, enabling it to determine whether a given SQL statement is likely to be an SQL injection attack. This, in turn, improves the overall security of the system.

2.2 Analysis of SQL injection statement characteristics

According to current research, SQL injection statements typically include key symbols such as (),|, *, +,$,?,!,@,::,[],%, and more. These statements also include important keywords such as: 'where', 'union', 'like', 'select', 'update', 'and', 'or', 'set', 'values', 'all', 'alter', 'as', 'create', 'deny', 'convert', 'char', 'ASCII', 'any', 'count', 'false', 'then', 'from', 'true', and 'null'.

On the other hand, conventional SQL statements involve the regular tasks that individuals commonly come across in their everyday interactions with databases. These procedures involve many duties, such as the creation, querying, and modification of tables. Common SQL queries often consist of terms such as 'create', 'select', 'insert', 'alter', 'drop', 'delete', 'union', 'set', 'database', 'and', 'or', 'information_schema', 'hex', and 'ASCII'. Sometimes it also contains some dangerous keywords similar to those in SQL injection statements: '–', '/','||', 'or', 'all', 'union', 'select', 'alter', etc. This phenomenon underscores the challenge of identifying SQL injection statements, which often lurk in the middle of normal SQL code and evade easy detection.

To gain a more comprehensive understanding of the attributes of SQL injection statements, this study begins with an analysis of 15 keywords within Kaggle's massive 'biggest-sql-injection' dataset, which contains 148326 data entries. The results of this analysis are shown in Fig. 1 and reveal a striking distinction, in which

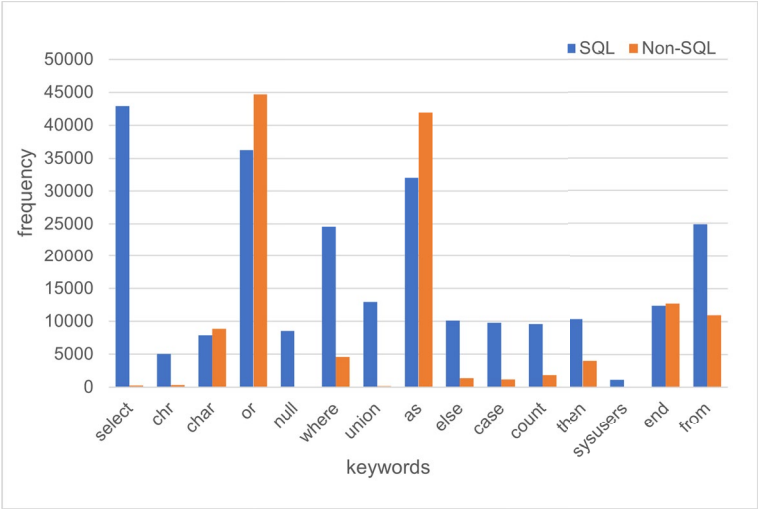

Fig. 1 Frequency of 15 keywords in SQL injection statement and Non-SQL injection statement

keywords such as 'select', 'null', 'where', and 'union' have a significantly higher prevalence within SQL injection statements compared to conventional SQL expressions. This observed phenomenon is primarily due to the characteristics of SQL injection attacks, which often involve database queries, and thus, these keywords are included more frequently.

Furthermore, our analysis of the dataset revealed that different forms of SQL injection attacks rely on different key statements to achieve their respective goals. These key statements are closely related to the chosen attack strategy and target. They can be categorized as follows:

(1) Blind SQL injection often leverages key statements such as 'and 1=1–', 'and 1=2–', 'sleep(5)', or 'if(1=1, sleep(5), 0)–'. These statements utilize Boolean logic or time delays to gradually infer the database structure or sensitive information. Even when the system does not directly return query results, attackers can indirectly obtain the desired information by analyzing changes in the system's response.
(2) Authentication bypass attacks exploit key statements such as 'or 1=1','or 1=1 and 1>0','or 1=1 or a=a '. By altering the authentication logic in SQL queries to ensure the condition is always true, attackers can bypass the authentication mechanism and gain unauthorized access to the system.
(3) False logical queries include phrases such as: 'wait for delay 0:0:10', 'and 1=cast(1 as XML)-',' and 1=cast(1 as int)-', 'select * from non_existent_table-', and more. These statements are often designed to cause logical errors, trigger error messages, or cause anomalous application behavior. Attackers use these errors to infer critical details such as database structure, field names, and data types and then strategize targeted attacks.
(4) Alternative encodings rely on symbols such as '%20','%3B','%3D', '%3A','%2C' and 'base64encode('payload')' to encode malicious SQL code. This manipulation is used to bypass the application's security filters.

There are striking differences between SQL injection statements and conventional SQL statements in the use of keywords and key statements. Each type of attack uses different keywords to achieve its goals. Thus, the use of word frequency-based features can skillfully identify and capture these differences, enabling effective detection and differentiation of SQL injections.

3 SQL injection detection model framework

3.1 Detection model framework

This paper presents an SQL injection detection model called SQLLS, which is constructed to effectively identify SQL injection statements by focusing on data text features extracted from SQL statements. The model is based on the Bi-LSTM network architecture. It includes three key stages: data processing, model training, and model

evaluation, as shown in Fig. 2. During the data processing stage, a series of preprocessing steps are applied to the data text, including stop word removal, word segmentation, and lowercase conversion, to ensure data quality and consistency. Next, the TF-IDF technique is employed to extract textual characteristics from SQL injection statements through the conversion of the textual data of SQL statements into numerical feature vectors.

Building upon this foundation, we employ the integrated feature selection method, GFC, to identify the most influential features for the classification task. This approach also eliminates features that have a minimal impact on classification performance, thereby achieving dimensionality reduction and enhancing the model's performance and efficiency. In the model training phase, we develop a lightweight SQLLS model based on the Bi-LSTM network and utilize mixed precision training to reduce memory consumption during model training. For model evaluation, we use six performance metrics: F1 score, recall, precision, accuracy, false positive rate (FPR), and misclassification, to provide a comprehensive assessment of the model's effectiveness.

3.2 TF-IDF algorithm

To better extract key features from SQL injection statements, we use the TF-IDF algorithm, whose calculation formulas are shown in formulas (1)–(3):

$$\mathrm{TF}(w_i, s) = \frac{\text{The number of occurrences of the lexical item } w_i \text{ in the data text } s}{\text{Total lexical items in text } s} \tag{1}$$

$$\mathrm{IDF}(w_i) = \log\left(\frac{N}{1 + \mathrm{DF}(w_i)}\right) \tag{2}$$

$$\text{TF-IDF}(w_i, s) = \mathrm{TF}(w_i, s) \times \mathrm{IDF}(w_i) \tag{3}$$

The term 'TF' refers to the frequency of vocabulary items appearing in the data text, which indicates the importance of the vocabulary item 'w_i' in the current data text.

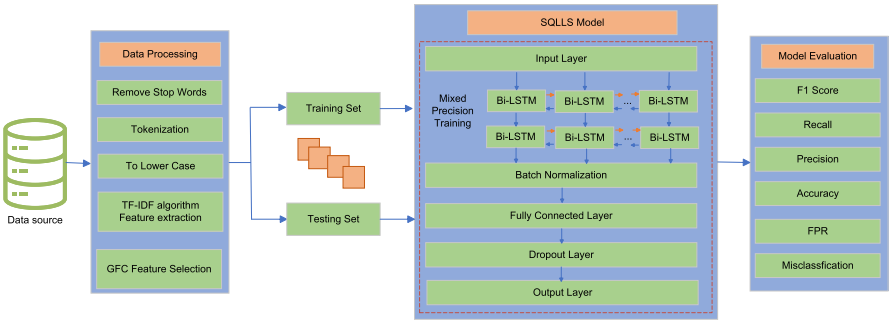

Fig. 2 The framework of the model

The term 'IDF' refers to the inverse document frequency, which measures how common a specific lexical entry is within the entire dataset. Its purpose is to diminish the significance of common lexical items prevalent across all data texts and to emphasize lexical items that occur frequently in the current data text but have a lower frequency across the entire dataset. Specifically, 'N' represents the total number of data texts within the dataset, while 'DF(w_i)' represents the count of data texts containing the lexical item 'w_i'.

'TF-IDF' represents a significant weighting of lexical items within the text data, derived from the product of TF and IDF values. This method effectively captures the frequency of lexical items within the current data text and their distribution across the entire textual dataset. As a result, it yields more distinct and discriminative feature vectors.

In the data processing part of this paper, the following steps are required:

(1) Input: Dataset $\mathrm{X} = \{\mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3, \ldots, \mathrm{X}_\mathrm{n}\}$, label $\mathbf{y} = \{y_1, y_2, y_3, \ldots, y_n\}$.
(2) Remove stop words: Stop words are words that occur frequently in data text but usually have no real meaning, such as 'the' and 'is'. Removing these stop words can help reduce noise and improve the quality of the data text.
(3) Word segmentation: Divide text data into vocabulary items for further processing.
(4) Normalization: Normalize lexical items to eliminate case differences, remove special characters or symbols, and handle inflection. This helps to treat lexical items with the same meaning but different forms as the same feature.
(5) Build a vocabulary: Collect lexical items from all documents to build a vocabulary. Each lexical item in the vocabulary corresponds to a unique index that identifies a column of the feature matrix. For example, the lexical item 'hello' in the vocabulary might be mapped to index 0 and the token 'how' to index 1. Each unique lexical item in the vocabulary corresponds to a feature.
(6) Calculate the TF-IDF value using formulas (1), (2), and (3).
(7) To construct a feature matrix, TF-IDF converts the data text into a feature matrix, where each row represents a data text, each column represents a vocabulary item, and each element in the matrix represents the TF-IDF value of the corresponding word. In this way, the feature matrix D is obtained.

The resulting feature matrix D is as follows:

$$D = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}$$

Among them, n means that there are n samples and k means that k vocabulary items (features) are extracted.

3.3 Feature selection

To reduce feature redundancy and enhance the effectiveness of SQL injection detection, this study combines three feature selection methods—GBRT, Fisher score, and the chi-square test [28, 29]—to propose an integrated feature selection approach called GFC. In this method, GBRT evaluates feature importance by leveraging a DT model, Fisher score selects features based on statistical significance, and the chi-square test further refines the feature set by measuring the correlation between features and true labels. This ensures that the final selected feature set possesses strong classification capability. The proposed method aims to fully utilize the learning potential of the integrated model to improve the accuracy and robustness of feature selection.

The feature selection process for GFC consists of the following steps. First, GBRT is used to rank the candidate features based on information gain, and the top m_1 most important features are selected. Next, Fisher score is applied to calculate the ratio scores of the features, and the top m_2 features are selected according to their rankings. Then, the feature sets obtained from GBRT and Fisher score are combined to form a unified feature set. Finally, the combined feature set is further refined using the chi-square test, and the top q features with the highest classification capability are selected as the final feature set.

3.3.1 GBRT feature selection

GBRT [30] feature selection evaluates the importance of each feature by leveraging a DT model that progressively optimizes the model's prediction accuracy using a gradient boosting algorithm [31]. This method excels at capturing complex nonlinear relationships between features, as each DT is adapted to the specific structure of the data, enabling more accurate modeling of complex dependencies. Compared to traditional linear models, GBRT effectively handles interactions between nonlinear features and enhances the model's generalization capability. In this study, the GBRT feature selection method calculates feature importance based on the integrated DT model and selects the top m_1 features with the highest contributions to SQL injection detection.

3.3.2 Fisher score feature selection

For the high-dimensional feature matrix D obtained through TF-IDF feature extraction in the previous section, this study first applies the Fisher score algorithm for feature selection. The primary objective of the Fisher score algorithm is to identify a feature subset in which the distance between data points of different categories is maximized, while the distance between data points within the same category is minimized [32]. The Fisher score is calculated as shown in Eq. (4).

$$F(X_i) = \frac{\sum_{j=1}^{c}(\mu_j^i - \mu^i)^2}{\sum_{j=1}^{c} n_j(\sigma_j^i)^2}, \tag{4}$$

Here, μ_j and σ_j represent the mean and standard deviation of the feature vector for the j-th class, respectively, while n_j denotes the sample size of the j-th class. After calculating the Fisher score for each feature, the top m_2 features with the highest scores are selected to form the output matrix Z, which has dimensions $m_2 \times n$.

Given an input matrix $X \in \mathbb{R}^{k \times n}$, where k represents the feature dimensionality and n denotes the number of samples, the Fisher score is used to compute a discriminative score for each feature. The k-dimensional features are then reduced to m_2 dimensions, and the resulting output matrix $Z \in \mathbb{R}^{m_2 \times n}$ contains the most discriminative m_2 features.

3.3.3 Chi-square feature selection

After applying Fisher score and GBRT for feature selection, the selected features from both methods are concatenated to form the final feature set. To further enhance the significance of the feature matrix and achieve dimensionality reduction, this study introduces the chi-square feature selection technique [33]. The formula for calculating the chi-square value of features is presented in Eq. (5).

$$X^2(\mathfrak{t}, \mathfrak{c}) = \sum_{e_t \in 0,1} \sum_{e_c \in 0,1} \frac{\left(N_{e_t e_c} - E_{e_t e_c}\right)^2}{E_{e_t e_c}} \tag{5}$$

Among them, t: term means that a certain feature has (1) or not (0), and c: class means category 1 or 0. N is the actual value, and E is the observed value.

The expected value calculation is shown in formulas (6)–(8) (the expected frequency is based on the assumption of independence between features and categories, and there is no correlation between the occurrence of features and categories):

$$E_{e_t e_c} = n \times p_{(e_t)} \times p_{(e_c)} \tag{6}$$

$$p_{(e_t)} = \frac{n_{e_t}}{n} \tag{7}$$

$$p_{(e_c)} = \frac{n_{e_c}}{n} \tag{8}$$

where n is the total number of samples, $e_t \in \{0, 1\}$ indicates whether feature t appears, and e_c indicates category 1 or 0.

The basic principle of chi-square feature selection is to evaluate the degree of association between a feature and its authentic label using the chi-square test. This evaluation plays a central role in determining whether a particular feature merits selection as a valuable classification attribute. The essence of this approach is to contrast the degree of divergence between the observed frequency and the expected

frequency of the feature and the accurate label. Specifically, the term 'observed frequency' refers to the actual category distribution within the sample data when subjected to given features and labels. Conversely, the 'expected frequency' is derived from statistical inference under the assumption of independence by projecting the expected category distribution. Calculating the discrepancy between the observed and expected frequencies yields the chi-square value, which serves as an indicator. A larger chi-square value indicates a higher correlation between the feature and the label, which means the feature is more relevant in classification efforts.

The steps of chi-square feature selection:

Step 1: Construct a contingency table for each feature, as shown in Table 1.

'n_{11}' is the number of samples in the total sample that have feature t_1 and are SQL injections.'n_{01}' is the number of samples in the total sample that have no feature t_1 but are SQL injections.'n_{10}' is the number of samples in the total sample that have feature t_1 but are not SQL injections.'n_{00}' is the number of samples in the total sample that have no feature t_1 and are not SQL injections.

Total sample size: $n = n_{11}+n_{10}+n_{10}+n_{00}$.

Step 2: Use formula (4) and formula (5) to compute the chi-square value for feature t_1.

Step 3: Iterate through steps 1 and 2 to compute the chi-square value for t_2 and continue this process until t_k is computed.

Step 4: Select the first q features (where q is less than k) with the largest chi-square value as the selected features. A larger chi-square value indicates a stronger correlation between the feature and the label. Therefore, selecting features with higher chi-square values helps to extract information-rich features while achieving dimensionality reduction goals. Feature selection is based on the chi-square value, resulting in the feature matrix D':

$$D' = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nq} \end{bmatrix} \quad \text{where } q < k$$

At this stage, the dimension q of the feature matrix D' is reduced compared to the original dimension k of the feature matrix D_0. This marks the end of the feature selection process.

Table 1 Contingency table

	Characteristic t_1	Featureless t_1
SQL injection (labeled 1)	n_{11}	n_{01}
Non-SQL injection (labeled 0)	n_{10}	n_{00}

Overall, GFC integrates multiple feature selection methods to obtain a more comprehensive and reliable feature subset. The primary objective is to extract hidden key information from SQL statement data and enhance detection accuracy.

4 SQLLS model design

After the data processing stage, we get the feature D'. To improve the accuracy of SQL injection statement detection, this study transforms the feature matrix D' into the required three-dimensional format compatible with the Bi-LSTM network. Subsequently, this adapted matrix is fed into the newly developed SQLLS model for training purposes.

4.1 The specific structure of the model

Figure 3 illustrates the architecture of the SQLLS model, which consists of several layers. These include an input layer, two bidirectional long short-term memory (LSTM) layers, a batch normalization layer, a fully connected layer, a dropout layer,

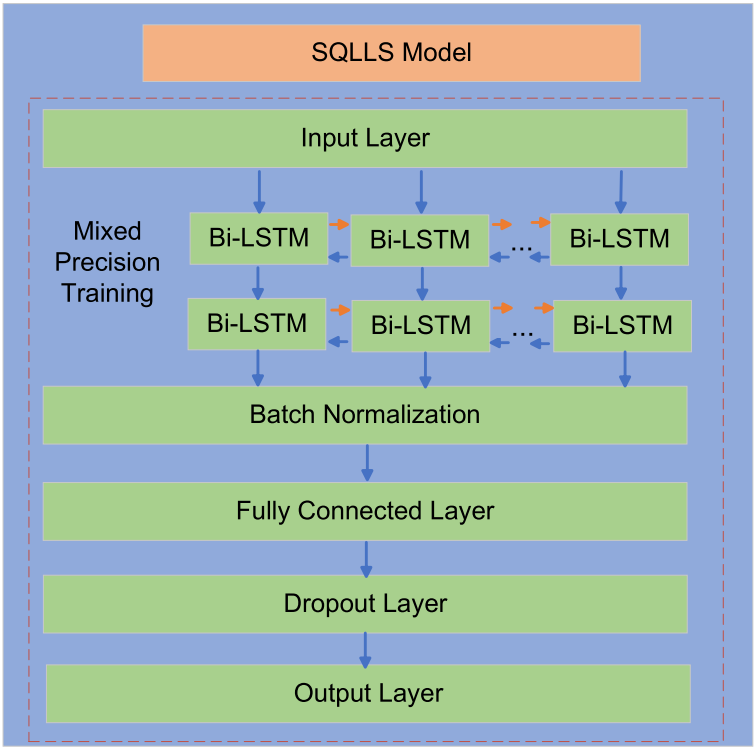

Fig. 3 SQLLS model

and an output layer. The first bidirectional LSTM layer contains 128 LSTM units, followed by a second layer of 64 LSTM units. Within each LSTM unit, there are different components: the forgetting gate, the input gate, and the output gate. These components regulate the flow and retention of information. To achieve this regulation, the outputs of these components are passed through a sigmoid activation function, constraining their values between 0 and 1. At the same time, the tanh activation function operates within the LSTM unit to compute new candidate memory states. The tanh function maps input data to the range $[-1, 1]$, facilitating the generation of new memory candidates. By synergizing the sigmoid and tanh functions, these bidirectional LSTM layers efficiently capture extensive dependencies in sequence data. The significance of the bidirectional LSTM lies in its ability to consider preceding and subsequent information in the sequence, effectively capturing patterns and dependencies for accurate predictions.

The role of the batch normalization layer is to normalize the output of the LSTM layer to improve model stability and generalization. It normalizes the data from each batch, providing more stable activation values even as the data distribution changes. In this model, placing the batch normalization layer after the bidirectional LSTM layer ensures normalized data distribution before entering the fully connected layer, optimizing the training process. This approach helps mitigate the vanishing gradient problem during training and accelerates model convergence.

The fully connected layer consists of 32 neurons, with connections between the output of the previous layer and other neurons established through a weight matrix, forming a dense, fully connected network structure. Specifically, this layer receives as input the output of all the neurons in the previous layer. Each neuron performs a weighted summation of these inputs using learned weights and then undergoes a nonlinear mapping through the sigmoid activation function. This process promotes interaction and integration of information between different neurons, thereby facilitating the capture of higher-level features and relationships.

The purpose of the drop-out layer is to randomly drop the output of certain neurons during the training phase. This practice minimizes co-adaptation among neurons, thereby reducing the risk of overfitting to specific subtleties of the training dataset. As a result, this mechanism enhances the model's ability to generalize to previously unseen data, contributing to increased stability and reliability.

The output layer consists of a single unit, and the activation function chosen is the sigmoid function. This choice allows the output of the model to be transformed into a probability value between 0 and 1. This probability value can then be compared to a threshold (set to 0.5 in this paper). Outputs above the threshold are classified as SQL injection statements, while outputs below the threshold are classified as non-SQL injection statements.

During the model training process, the cross-entropy function serves as the loss function. Its primary purpose is to quantify the dissimilarity between the model's predictions and real-world outcomes. By minimizing this cross-entropy loss, we force the model's predictions to align as closely as possible with actual observations, allowing the model to learn and predict with increased accuracy. Incorporating the cross-entropy loss function guides the model in parameter adjustment during training, resulting in a well-defined and plausible distribution of expected probabilities

across classes. As a result, the model becomes more adept at distinguishing between different categories during the learning phase, resulting in improved classification performance. In addition, we employ mixed precision for model training, a strategy designed to reduce computational and memory requirements while maintaining model accuracy.

4.2 Construction of the LSTM network

Due to the demonstrated ability of LSTM [34] to capture long-term memory information in sequence data, it excels in tasks that require remembering long-term dependencies. Therefore, in this paper, the LSTM network is used as one of the core networks of this model.

LSTM is a special type of recurrent neural network (RNN) [35]. Similar to other RNN units, LSTM has recurrent connections but introduces interacting input gates, output gates, and forgetting gates instead of single neuron gates. These three gate mechanisms regulate the input and output of information in cells, allowing the LSTM network to determine what information to remember, forget, or output based on past states, stored memories, and current inputs. The LSTM network effectively solves the gradient disappearance problem that can occur in traditional RNNs and can better capture long-range dependencies. Therefore, LSTMs are very effective in handling classification, processing, and prediction tasks based on sequence data [29, 30].

According to the data characteristics of SQL injection detection in this paper, the LSTM structure built is shown in Fig. 4, and the specific design of each specific link is introduced as follows:

(1) Forgotten Gate

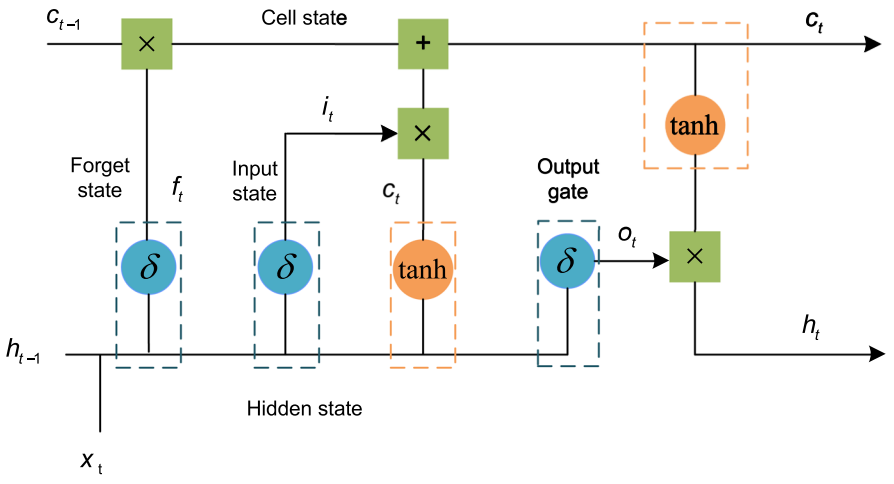

Fig. 4 LSTM network architecture

The forgotten gate's task is to decide which information should be discarded by the cell state. The calculation formula used is shown in formula (9), where f_t represents the information to be retained (close to 1) and the information to be forgotten (close to 0).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{9}$$

The feature vector of the t-th time step of the input sequence is denoted as x_t, and $[h_{t-1},x_t]$ denotes the concatenation. The feature vector of the t-th time step of the input sequence is denoted as x_t, and $[h_{t-1}, x_t]$ denotes the concatenation of the hidden state h_{t-1} from the previous time step and the output feature x_t from the current time step. In this context, W_f represents the forget gate's weight matrix, b_f represents the forget gate's bias vector, and σ denotes the sigmoid function responsible for mapping values to the range $[-1, 1]$.

(2) Input gate

In this stage, the previous cell information C_{t-1} is updated to generate the new cell information C_t. The updating process consists of two steps: In this stage, the previous cell information C_{t-1} is updated to generate the new cell information C_t. The updating process consists of two steps: selectively forgetting part of the old cell information using the forget gate and incorporating part of the candidate cell information $\tilde{C}_t$ through the input gate to generate the updated cell information C_t. The mathematical formulas used for these calculations are shown in Eqs. (10)–(12).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{10}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{11}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{12}$$

W_i and W_C, respectively, represent the weight matrix of the input gate and the weight matrix of the candidate cell state, and b_i and b_c represent the bias vector of the input gate and the candidate cell state, respectively. Tanh(.) is a hyperbolic tangent function used to map the weighted input to a range between -1 and 1.

(3) Output gate

In this phase, the state characteristics of the output cell are determined based on the input h_{t-1} and x_t. To achieve this, the input is passed through the output gate's sigmoid layer, yielding the attenuation coefficient o_t. This coefficient is then multiplied by the scaled memory state C_t to derive the output information at time t, called the hidden state h_t. The detailed calculation is shown in Eqs. (13) and (14).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{13}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{14}$$

where W_o and b_o are the weight matrix and bias vector of the output gate, respectively.

$\sigma(.)$ is the logistic sigmoid function used as the activation function of the gate, as shown in Eq. (15).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{15}$$

Tanh(.) is a hyperbolic tangent function that is used as the activation function of the input and output modules, as shown in Eq. (16).

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{16}$$

4.3 Building the Bi-LSTM network

As LSTM can only fit sequence-related data from one direction, it cannot adapt to complex nonlinear models. To more effectively capture the sequential relationships and features within textual data, we employ the Bi-LSTM network. The Bi-LSTM network is a sophisticated fusion of forward and backward LSTMs. Its primary strength lies in its ability to interweave information from these two directions, thereby facilitating bidirectional information transfer and processing. The calculation formula for Bi-LSTM is presented below. The output calculation is shown in Eq. (17):

$$y_t = \delta(W_{\underset{h_y}{\rightarrow}} \overrightarrow{h}_t) + \delta(W_{\underset{h_y}{\leftarrow}} \overleftarrow{h}_t) \tag{17}$$

Among them, $W_{\underset{h_y}{\rightarrow}}$ and $W_{\underset{h_y}{\leftarrow}}$ denote the weight matrices of the forward and backward LSTMs at that moment in time, respectively, which are used to map the forward and backward hidden states to the output space, and b_y is the bias vector, which is used to adjust the offset of the linear combination.

The forward hidden state calculation formula (18) is shown as:

$$\overrightarrow{h}_t = \mathrm{LSTM}(x_t, \overrightarrow{h}_{t-1}) \tag{18}$$

LSTM(.) represents the LSTM network, and $\vec{h}_{t-1}$ represents the forward hidden state of the previous time step. $\vec{h}_t$ is the forward LSTM, which calculates the hidden state at each time step that contains the sequence information before the current time step. The backward hidden state calculation is shown in formula (19):

$$\overleftarrow{h}_t = \mathrm{LSTM}(x_t, \overleftarrow{h}_{t+1}) \tag{19}$$

$\overleftarrow{h}_t$ is the backward LSTM, which calculates the hidden state at each time step and contains information about the sequence after that time step.

Through this two-way computational method, Bi-LSTM forms a global 'context window' when modeling sequence data, which can simultaneously consider the past and future of the sequence to more accurately capture the correlation and pattern in the sequence [31, 32, 35] (Fig. 5).

4.4 Lightweight Bi-LSTM

To reduce the complexity and enhance the computational efficiency of the Bi-LSTM model, we apply a pruning technique. The fundamental concept of pruning is to remove unimportant weights from the network, thereby reducing both computation and memory consumption, while retaining as much predictive power as possible. After pruning, the network parameters θ' become sparse, with the sparsity ratio controlled by a parameter p_{weight}, which determines the proportion of nonzero weights retained. Specifically, the relationship between the number of nonzero weights $|m|$ in the pruned network and the total number of original network parameters $|\theta|$ is expressed by the following equation:

$$|m| = p_{\text{weight}} \cdot |\theta| \tag{20}$$

where $|m|$ represents the number of nonzero weights retained after pruning, and $|\theta|$ denotes the total number of weights in the original network. This relationship allows

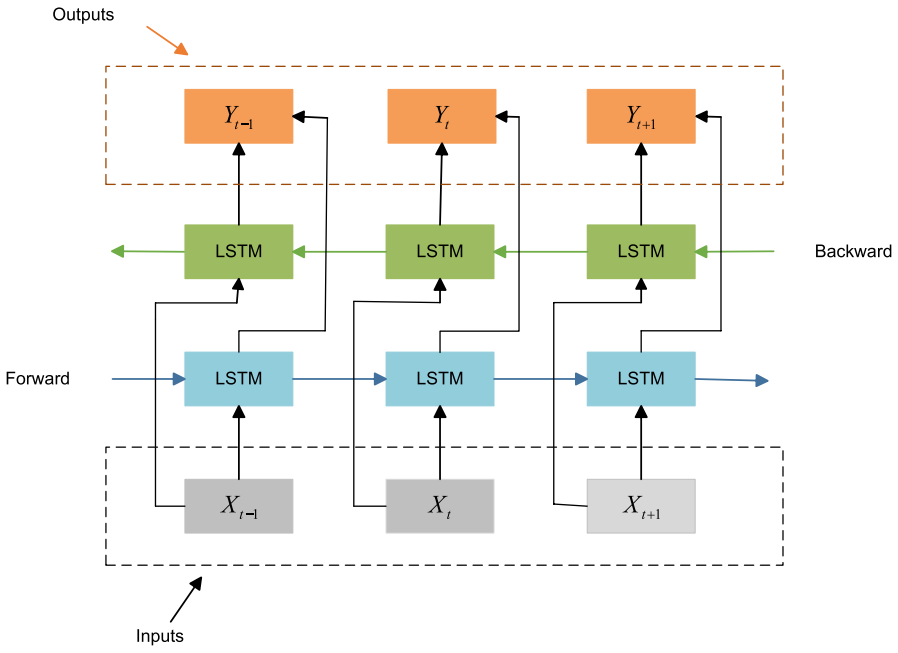

Fig. 5 Bi-LSTM architecture

the pruning technique to control the proportion of retained weights, thereby effectively reducing computational resource consumption.

The computation process of the pruned Bi-LSTM retains the same structure as the original Bi-LSTM; however, only the retained weights after pruning are used in the computation. Consequently, pruning does not alter the fundamental operations of the network but reduces the computational workload at each time step. In both the forward LSTM and backward LSTM, the computational formulas are as follows:

$$\vec{h}_t = \text{LSTM}(x_t, \vec{h}_{t-1}, \theta') \tag{21}$$

$$\overleftarrow{h}_t = \text{LSTM}(x_t, \overleftarrow{h}_{t+1}, \theta') \tag{22}$$

where θ' represents the weights after pruning, x_t is the input at the current time step, and $\vec{h}_t$ and $\overleftarrow{h}_t$ denote the forward-hidden state at the previous time step and the backward-hidden state at the subsequent time step, respectively. In this way, the computation of the model after pruning continues to rely on the information from both the previous and subsequent time steps, but the computational load is significantly reduced.

Through the pruning technique, we make the Bi-LSTM model more lightweight, thereby reducing computational overhead, which is particularly efficient in environments with limited memory and computational resources. Despite pruning, the model's performance does not degrade significantly, making the lightweight Bi-LSTM especially suitable for resource-constrained scenarios.

4.5 Loss function design

During the model training process, cross-entropy is adopted as the loss function, as it is particularly well-suited for the SQL injection detection task. The cross-entropy loss function quantifies the difference between the model's predicted values and the true labels, guiding the model to iteratively optimize its parameters during training to enhance prediction accuracy. The mathematical formulation of the cross-entropy loss function is presented in Eq. (23).

$$L = -\frac{1}{N}\Sigma_{i=1}^{N} y_i \cdot log\hat{y_i} + (1 - y_i) \cdot log(1 - \hat{y_i}) \tag{23}$$

Here, L symbolizes the loss function, while N is the total number of training samples. The vector $y = [y_1, y_2 \ldots y_n]^T$ denotes the expected output values of the samples, where y_i denotes the expected output value of the ith sample, essentially representing its true label.

To optimize the loss function, the gradient descent algorithm and its variant, the Adam optimizer, are employed. During each iteration, the optimization algorithm calculates the gradient of the loss function and updates the model weights accordingly to progressively reduce prediction error. Over multiple iterations, the loss function converges, and the model parameters are iteratively refined, enabling the

final model to achieve higher accuracy on the training data and better generalization to new data.

5 Experimental verification and analysis of results

To validate the effectiveness of the proposed model, extensive experiments were conducted on four datasets. The SQLLS model was compared with four classical algorithms: K-nearest neighbors (KNN), Naive Bayes (NB), logistic regression (LR), and gradient boosting (GB). Comparative analyses were also performed against state-of-the-art models to further evaluate the performance and advantages of the SQLLS model. Additionally, ablation experiments were conducted to verify the validity of the innovative components within the model. Through these experiments and comparative analyses, the effectiveness and sophistication of the proposed model were comprehensively validated.

5.1 Datasets

To verify the validity and reliability of the proposed SQLLS model, four representative public datasets were utilized in this study. Datasets 1 and 2, both sourced from the Kaggle website, were employed to evaluate the model's performance in general scenarios. Dataset 3, a dedicated dataset for cross-site scripting (XSS) attacks, was specifically constructed to comprehensively assess the model's adaptability and practicality across different scenarios. Dataset 4 contains multiple complex types of SQL injection attacks, designed to evaluate the model's ability to identify various attack types and their complexities. The details of these datasets are presented in Tables 2, 3, 4 and 5.

Table 2 SQL injection detection dataset 1

Label	Description	Count	Proportion (%)
1	SQL injection statement	77750	52.418
0	Non-SQL injection statement	70576	47.582

Table 3 SQL injection detection dataset 2

Label	Description	Count	Proportion (%)
1	SQL injection statement	11382	36.812
0	Non-SQL injection statement	19537	63.188

Table 4 XSS attack dataset 3

Label	Description	Count	Proportion (%)
1	XSS attack statement	1028	50.516
0	Non-XSS attack statement	1007	49.484

Table 5 SQL injection detection dataset 4

Label	Description	Count	Proportion (%)
0	Auth_Bypass	248	0.446
1	Blind_SQLi	16039	28.793
2	DoS	320	0.576
3	In_Band_SQLi	15474	27.768
4	Normal_SQL	37710	67.660

5.2 Evaluation indicators

Following the analysis, we evaluate the performance of our method using several evaluation metrics, including accuracy, precision, recall, F1 score, FPR, and misclassification rate. The confusion matrix, which includes True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN), was computed to provide an overview of the results. The detailed presentation is shown in Table 6.

Accuracy can be defined as the ratio of accurately identified classes to the overall number of classifications.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{24}$$

Precision measures the proportion of correctly predicted positive instances among all predicted positive instances. The accuracy calculation is represented by the formula (25).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{25}$$

Recall, alternatively referred to as the true positive rate, sensitivity, or detection probability, denotes the proportion of accurately anticipated positive instances about the total number of real positive instances. The recall rate can be determined using the formula (26).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{26}$$

The F1 score, a measure of accuracy, assesses precision and recall simultaneously. It represents the harmonic mean of precision and recall, and its calculation is demonstrated in Eq. (27).

Table 6 Confusion matrix table

True labels	Predicted labels	
	0	1
0	TN	FP
1	FN	TP

$$\mathrm{F1} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} = \frac{2 \times \mathrm{TP}}{2 \times \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{27}$$

FPR represents the ratio of samples incorrectly classified as positive instances to the total number of actual negative instances in the classification model. The calculation formula for FPR is provided in Eq. (28).

$$\mathrm{FPR} = \frac{FP}{FP + TN} \tag{28}$$

The misclassification rate, also known as the classification error, is the proportion of incorrect predictions to the total number of samples. The formula for calculating the misclassification rate is provided in Eq. (29).

$$\mathrm{Misclassification} = \frac{FP + FN}{TP + TN + FP + FN} \tag{29}$$

5.3 Experimental setup

In this study, all four datasets were systematically divided into training, validation, and test sets, with the training set comprising 80% of the total data, the validation set 10%, and the test set 10%. This division was designed to comprehensively evaluate the performance of the model. Additionally, the Adam optimizer was employed during model training to ensure both efficiency and stability in the optimization process. The detailed hyperparameter configurations for each dataset are presented in Table 7.

The hyperparameter configurations in Table 7 were chosen to balance model performance and computational efficiency. The Bi-LSTM neuron counts and dropout rates were tuned to prevent overfitting while capturing temporal dependencies. Batch sizes and learning rates were selected to optimize gradient stability and convergence speed. The number of epochs was adjusted based on dataset complexity, and activation functions were chosen to suit the nature of the classification tasks. These choices ensure the model performs reliably across different datasets.

Table 7 Hyperparameterization

Datasets	First dataset	Second dataset	Third dataset	Fourth dataset
Neurons in Bi-LSTM	128/64	128/64	128/64	128/64
Dropout	0.3	0.3	0.4	0.3
Batch size	32	32	16	32
Learning rate	0.0001	0.0001	0.0006	0.0006
Epoch	100	30	300	100
Activation function	Sigmoid	Sigmoid	Sigmoid	Softmax

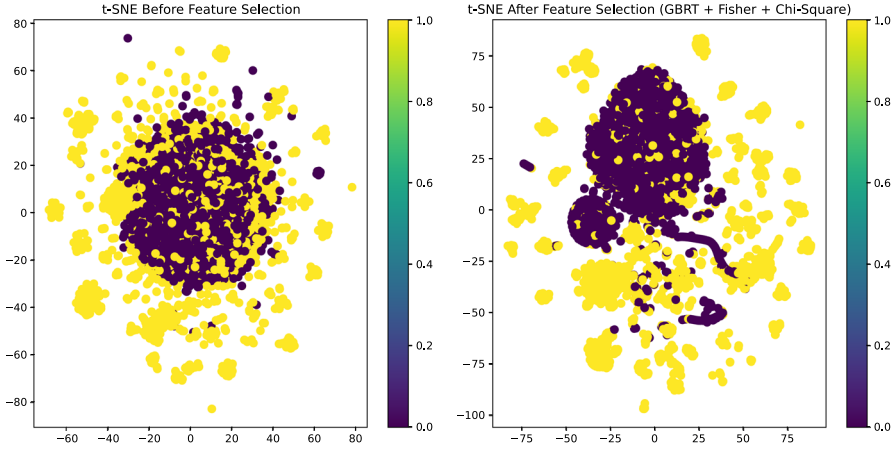

Fig. 6 T-SNE dimensionality reduction classification results before and after GFC feature selection for dataset 1

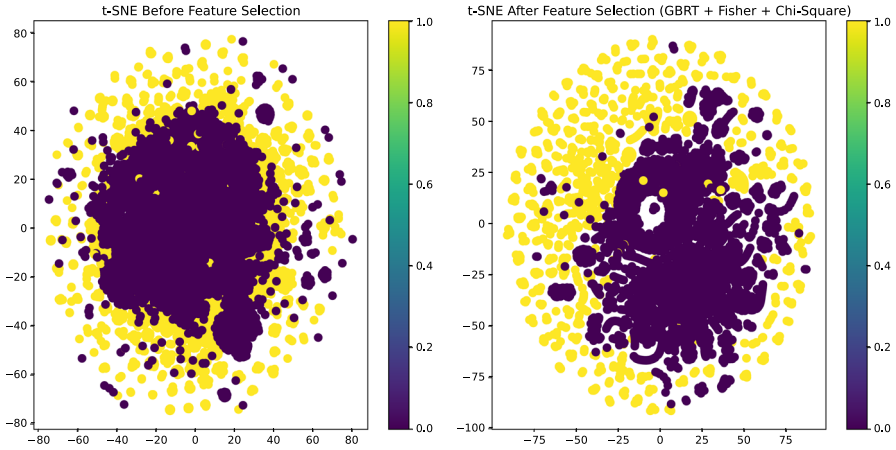

Fig. 7 T-SNE dimensionality reduction classification results before and after GFC feature selection for dataset 2

5.4 Analysis of results

5.4.1 t-SNE visualization

To verify the effectiveness of GFC feature selection in SQL injection attack detection, the t-SNE algorithm was employed to reduce the high-dimensional feature vectors to two dimensions. The t-SNE algorithm preserves the local structure of the data while revealing important global relationships. Figures 6, 7, 8 and 9 illustrate the effects of dimensionality reduction before and after GFC feature selection for the four datasets.

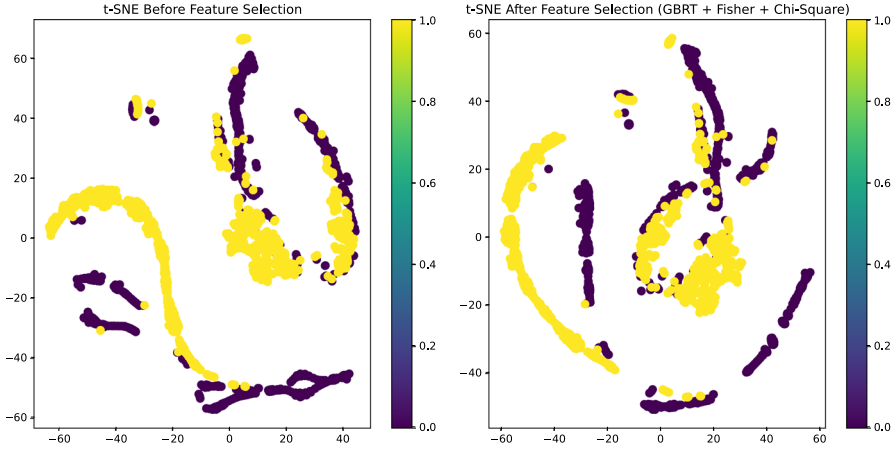

Fig. 8 T-SNE dimensionality reduction classification results before and after GFC feature selection for dataset 3

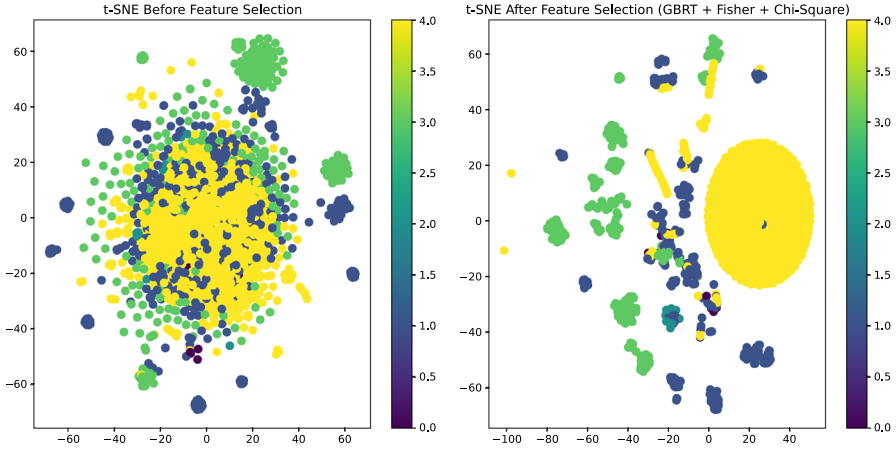

Fig. 9 T-SNE dimensionality reduction classification results before and after GFC feature selection for dataset 4

Before feature selection, the sample points are distributed chaotically in the two-dimensional space, with unclear category boundaries. This indicates that the original data contain numerous redundant, noisy, or irrelevant features, which reduce the differentiation between samples. After applying GFC feature selection, the dimensionality of the feature matrix was significantly reduced by filtering out the features most relevant to the target variable. In the two-dimensional space, the distribution of sample points after the dimensionality reduction is more distinct, and the category boundaries are clearer. This demonstrates that GFC feature selection effectively removes redundant and irrelevant features, resulting in the

optimized feature matrix D', which enhances the differentiation between samples and contributes to improved classification performance.

5.4.2 Loss function and accuracy validation results

As illustrated in Figs. 10, 11 and 12, the model demonstrates the advantage of fast convergence across all experiments, achieving optimal performance within 20–50 epochs. During this process, the training and validation losses remain consistent with accuracy, without signs of overfitting, indicating that the model exhibits strong

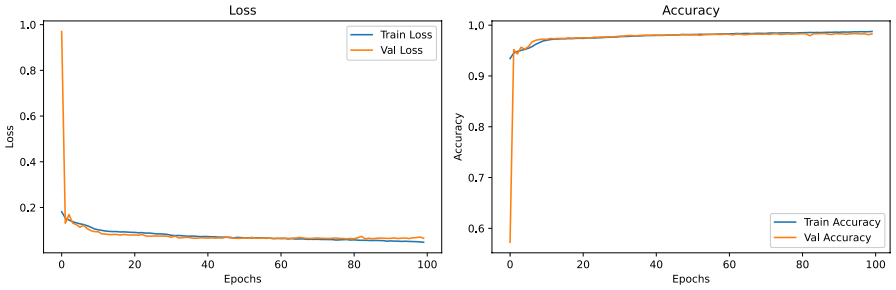

Fig. 10 Loss function for training and test data for dataset 1 on the left. Accuracy curves for training and test data for the right dataset 1

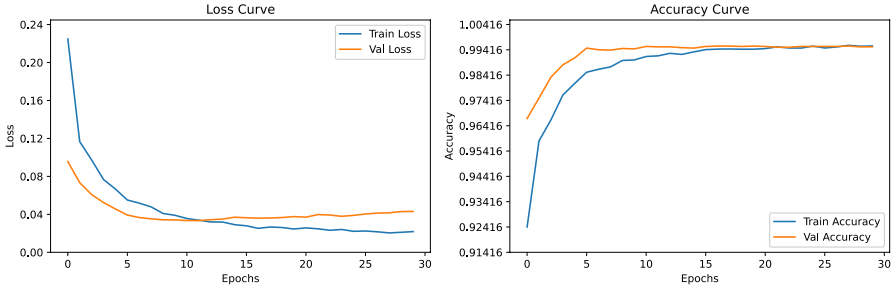

Fig. 11 Loss function for training and test data for dataset 2 on the left. Accuracy curves for training and test data for the right dataset 2

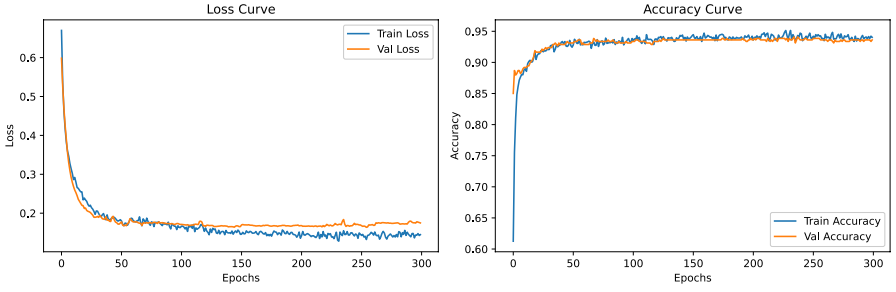

Fig. 12 Loss function for training and test data for dataset 3 on the left. Accuracy curves for training and test data for the right dataset 3

stability and generalization ability. Additionally, the validation loss is slightly lower than the training loss, suggesting that the model fits the validation set better, thereby achieving overall high robustness and accuracy.

5.4.3 Comparison of binary classification results

The results of the proposed SQLLS model are presented in Tables 8, 9 and 10, as well as Figs. 13, 14 and 15, where it is compared with KNN, NB, LR, GB, and the proposed SQLLS model using six evaluation metrics. Across both Dataset 1 and Dataset 2, the SQLLS model outperforms the other four models in all six metrics, achieving low false alarm and misclassification rates. Similarly, on Dataset 3, the SQLLS model performs equally well. For this dataset, the SQLLS model achieves an F1-Score of 94.954%, recall of 96.275%, precision of 93.665%,

Table 8 Comparison of classifier metrics for dataset 1

Metric	KNN (%)	NB (%)	LR (%)	GB (%)	SQLLS (%)
F1-Score	87.672	87.692	94.318	96.646	98.416
Recall	89.748	90.238	90.818	95.728	98.118
Precision	85.690	85.286	98.100	97.583	98.716
Accuracy	86.796	86.749	94.276	96.525	98.348
FPR	16.442	17.078	1.930	2.601	1.399
Misclassification	13.204	13.251	5.724	3.475	1.652

Table 9 Comparison of classifier metrics for dataset 2

Metric	KNN (%)	NB (%)	LR (%)	GB (%)	SQLLS (%)
F1-Score	99.078	89.319	95.069	98.573	99.321
Recall	98.472	81.580	91.314	97.992	99.909
Precision	99.691	98.680	99.147	99.161	99.735
Accuracy	99.321	92.772	96.491	98.949	99.499
FPR	0.180	0.642	0.462	0.488	0.154
Misclassification	0.679	7.228	3.509	1.051	0.598

Table 10 Comparison of classifier metrics for dataset 3

Metric	KNN (%)	NB (%)	LR (%)	GB (%)	SQLLS (%)
F1-Score	85.210	59.951	63.391	92.383	94.954
Recall	89.767	56.952	61.620	91.441	96.275
Precision	81.092	99.070	81.395	94.419	93.665
Accuracy	83.538	72.326	70.140	92.906	94.585
FPR	18.824	83.958	56.875	9.896	7.292
Misclassification	16.462	40.049	36.609	7.617	5.406

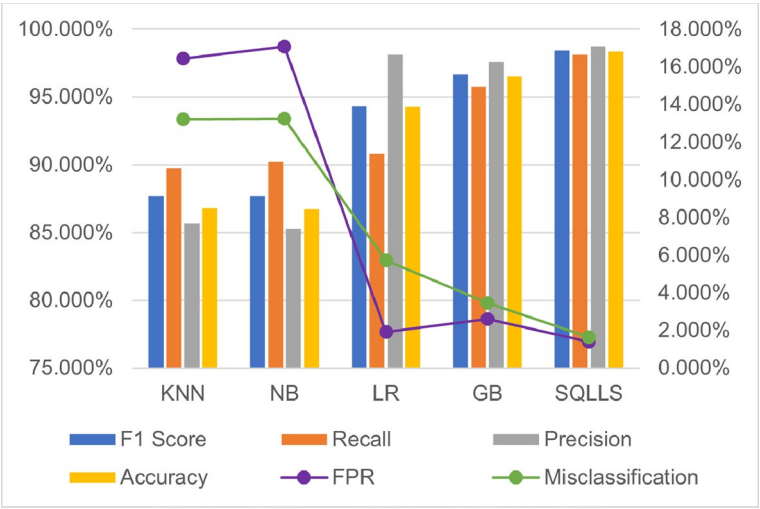

Fig. 13 Comparison chart of classifier indicators-dataset 1

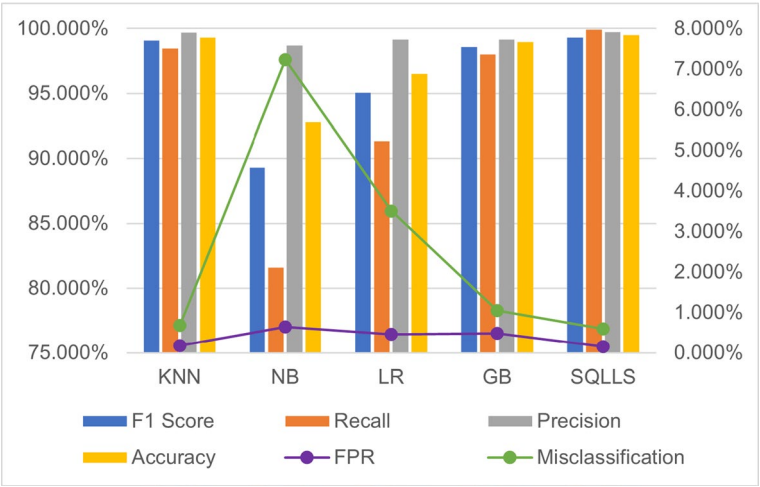

Fig. 14 Comparison chart of classifier indicators-dataset 2

and accuracy of 94.585%, significantly surpassing traditional algorithms such as KNN, NB, LR, and GB, while maintaining low false positive and misclassification rates. These results demonstrate that the SQLLS model excels not only in SQL injection attack detection but also in other types of attack detection, showcasing its strong robustness and efficiency.

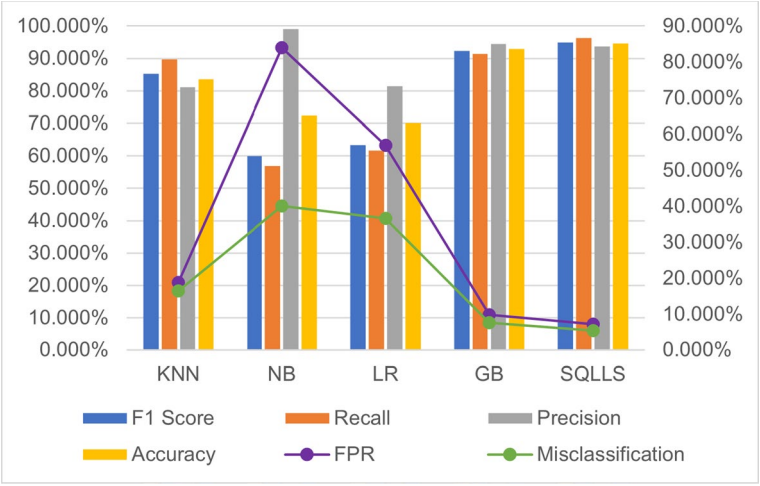

Fig. 15 Comparison chart of classifier indicators-dataset 3

5.4.4 Comparison of multi-classification results

In the field of SQL injection detection, addressing multi-classification tasks often involves significant complexity. Many existing models struggle to simultaneously handle multiple types of SQL injection attacks effectively. However, the proposed SQLLS model demonstrates the ability to efficiently and accurately identify various SQL injection attacks and classify them into their corresponding attack types. In this study, the performance of the SQLLS model is compared with that of classical machine learning methods.

As shown in Table 11, the SQLLS model demonstrates significant performance advantages on the same dataset, particularly when handling multiple types of SQL injection attacks. It effectively distinguishes between different attack types and normal benign operations. Compared to traditional machine learning algorithms, such as KNN and NB, the SQLLS model achieves superior performance across several key metrics.

The primary reason for this advantage is the use of the GFC feature selection method. This method integrates three feature selection techniques—GBRT, Fisher score, and the chi-square test—to effectively identify the most influential features for the detection task while minimizing noise and redundant information. By leveraging this approach, the SQLLS model more accurately captures the features of different types of SQL injection attacks, thereby enhancing both the accuracy and robustness of detection.

Table 11 Performance comparison of different algorithms

Algorithm	Class	Precision (%)	Recall (%)	F1-Score (%)
KNN	Benign payloads	94.118	47.059	62.745
	Authentication bypass	95.136	97.630	96.367
	Blind SQL injection	97.321	99.091	98.198
	Denial of service	98.908	99.505	99.206
	Classic SQL injection	99.020	97.964	98.489
NB	Benign payloads	95.455	30.882	46.667
	Authentication bypass	98.296	88.297	93.028
	Blind SQL injection	92.982	96.364	94.643
	Denial of service	96.115	92.700	94.377
	Classic SQL injection	93.762	99.480	96.537
LR	Benign payloads	76.471	76.471	76.471
	Authentication bypass	97.177	92.408	94.732
	Blind SQL injection	100.000	90.909	95.238
	Denial of service	99.603	97.158	98.365
	Classic SQL injection	96.540	99.586	98.039
GB	Benign payloads	75.342	80.882	78.014
	Authentication bypass	94.484	84.417	89.167
	Blind SQL injection	93.162	99.091	96.035
	Denial of service	99.885	93.368	96.516
	Classic SQL injection	92.550	99.066	95.697
SQLLS	Benign payloads	75.522	97.001	84.156
	Authentication bypass	100.000	99.999	99.992
	Blind SQL injection	99.625	100.000	100.000
	Denial of service	100.000	99.986	100.000
	Classic SQL injection	99.995	99.985	100.000

5.4.5 Deep Learning algorithm performance and runtime comparison

To assess the impact of pruning, we compared SQLLS with the unoptimized Bi-LSTM, LSTM, and the recently proposed CNN+BiLSTM model [24], with the results presented in Table 12. The analysis reveals that CNN+Bi-LSTM achieves an accuracy of 99.91% in the Benign Payloads category, which is impressive, but attains a lower accuracy of 91.75% in the Authentication Bypass category. In contrast, LSTM achieves an accuracy of only 70.551% in the Benign Payloads category, which is substantially lower than that of CNN+Bi-LSTM and the pruning-optimized SQLLS. The optimized SQLLS maintains near 100% accuracy across most categories, particularly excelling in attack categories such as Blind SQL Injection and Denial of Service. With respect to recall, CNN+Bi-LSTM achieves 99.93% recall in the Benign Payloads category, but demonstrates a lower recall of 89.90% in the Authentication Bypass category. LSTM achieves 95.436% recall in the Benign Payloads category, while Bi-LSTM exceeds 99% recall in

Table 12 Performance comparison and runtime of deep learning algorithms

Algorithm	Class	Precision (%)	Recall (%)	F1-Score (%)	Run time
CNN+LSTM[24]	Benign payloads	99.91	99.93	99.92	31 min
	Authentication bypass	91.75	89.90	90.82	
	Blind SQL injection	99.89	99.95	99.92	
	Denial of service	100.00	100.00	100.00	
	Classic SQL injection	99.98	99.95	99.97	
LSTM	Benign Payloads	70.551	95.436	86.364	24 min 4 s
	Authentication bypass	100.000	99.279	99.821	
	Blind SQL injection	99.624	99.561	97.592	
	Denial of service	98.120	97.000	100.000	
	Classic SQL injection	99.773	99.209	99.378	
BiLSTM	Benign payloads	75.000	97.436	84.298	32 min 17 s
	Authentication bypass	99.686	99.624	99.655	
	Blind SQL injection	99.702	99.935	100.000	
	Denial of service	100.000	99.990	100.000	
	Classic SQL injection	99.686	99.982	100.000	
SQLLS (Light weight)	Benign payloads	75.522	97.001	84.156	20 min 39 s
	Authentication bypass	100.000	99.999	99.992	
	Blind SQL injection	99.625	100.000	100.000	
	Denial of service	100.000	99.986	100.000	
	Classic SQL injection	99.995	99.985	100.000	

several attack categories, including Denial of Service and Blind SQL Injection. The pruning-optimized SQLLS maintains 100% recall in the Blind SQL Injection category, showcasing superior detection capabilities. In terms of F1-Score, CNN+Bi-LSTM achieves 99.92% in the Benign Payloads category, but attains a lower F1-Score of 90.82% in the Authentication Bypass category. LSTM's F1-Score in other categories is generally lower than that of CNN+Bi-LSTM and SQLLS, which achieve nearly 100% F1-Score in attack detection tasks. In terms of training time, CNN+Bi-LSTM requires 31 min, resulting in high computational overhead, while LSTM takes 24 min and 4 s, and BiLSTM requires 32 min and 17 s. In contrast, the pruning-optimized SQLLS requires only 20 min and 39 s, significantly enhancing computational efficiency, reducing redundant calculations, and maintaining high accuracy. In summary, the pruning-optimized SQLLS performs exceptionally well in terms of accuracy, recall, and F1-Score, while also significantly reducing training time. This makes it particularly suitable for real-time detection and identification of SQL injection attacks in resource-constrained environments, establishing it as a more efficient and powerful tool.

5.4.6 Ablation experiments

To evaluate the independent contributions of different feature selection methods to the performance of SQL injection attack detection models and verify their

Table 13 Performance comparison across datasets and feature selection methods

Dataset	Feature Selection	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Dataset 1	Remove the GBRT	97.492	99.544	95.644	97.555
	Remove the Fisher score	96.851	99.444	94.510	96.914
	Remove the Chi-square	97.907	99.567	96.417	97.967
	GFC	98.348	98.716	98.118	98.416
Dataset 2	Remove the GBRT	98.609	98.805	97.424	98.110
	Remove the Fisher score	97.833	99.224	94.893	97.010
	Remove the Chi-square	97.995	99.049	95.504	97.244
	GFC	99.499	99.735	98.909	99.321
Dataset 3	Remove the GBRT	92.875	90.789	96.279	93.454
	Remove the Fisher score	92.138	90.667	94.884	92.727
	Remove the Chi-square	92.383	90.351	95.814	93.002
	GFC	94.585	93.665	96.275	94.954

significance, ablation experiments were conducted (as shown in Table 13). The experimental results demonstrate that different feature selection methods significantly influence the performance of the detection model, with GBRT and Fisher score being critical for improving accuracy and reducing the false alarm rate.

When GBRT feature selection is removed, the accuracy and false alarm rate fluctuate significantly across all datasets, particularly in scenarios with limited data, where the false alarm rate increases markedly. Similarly, removing Fisher score has a considerable impact on both the recall and false alarm rate, highlighting its vital role in enhancing the model's ability to detect positive classes. In contrast, removing chi-square feature selection has a relatively smaller effect, but it still contributes to further optimization of the F1 score and false alarm rate.

By integrating GBRT, Fisher score, and chi-square test into the GFC feature selection method, the SQL injection attack detection model effectively exploits classification features, achieving an accuracy of up to 99.466%. These results demonstrate that the GFC feature selection method successfully integrates the advantages of multiple feature selection techniques, significantly improving the accuracy and robustness of SQL injection attack detection.

6 Conclusion

In this paper, we propose an efficient SQL injection detection model, SQLLS, based on Bi-LSTM networks. The core methodology of the model involves using the TF-IDF algorithm to convert SQL statements into numerical feature vectors, thereby extracting key features and enhancing the model's representational capabilities. To improve the accuracy and robustness of feature selection, we introduce an integrated feature selection method, GFC, which combines techniques such as GBRT, Fisher score, and chi-square test to comprehensively evaluate the relevance and importance of features to the target labels, ultimately filtering out the most discriminative

features. To reduce the complexity and enhance the computational efficiency of the Bi-LSTM model, this paper introduces the pruning technique, which optimizes computational overhead by removing unimportant weight connections. This approach makes the model more lightweight and suitable for resource-constrained environments. As a result, the performance of the SQLLS model is significantly improved when compared to classical algorithms such as KNN, NB, LR, and GB. In comparisons with classical algorithms such as KNN, NB, LR, and GB, the SQLLS model outperforms the other four models across six metrics, including F1-Score, recall, precision, accuracy, FPR, and misclassification. The model achieves an accuracy of up to 100.000% with a false alarm rate as low as 0.154%. This model effectively addresses the challenges of high false alarm rates, lack of accuracy, and overfitting observed in existing neural network-based SQL attack detection methods, demonstrating significant potential in SQL injection detection.

7 Future research directions

Future research can focus on four key directions. First, current feature selection methods are static; adaptive feature selection and incremental learning techniques could be explored to dynamically adjust the feature set, enabling the model to cope with changes in attack patterns and continuously adapt to new attack strategies. Second, with the increasing application of deep learning in the security domain, enhancing model interpretability is crucial to ensure the detection process is both accurate and explainable, thereby fostering user trust. Third, improving the robustness of models against adversarial attacks is essential to maintain efficient detection performance, even in the face of sophisticated intelligent attacks. Finally, we plan to collaborate with industry partners or deploy the algorithm in real-world systems to assess its performance under actual operational conditions, helping validate its effectiveness and adaptability in dynamic and complex environments.

Author contributions Q.Q. and Y.L. provided the methodology, did the experiments, and wrote the subject manuscript; Y.M. visualized the data; J.S. did the drawing; K.W. and Z.W. grammar-checked the manuscript; all authors reviewed the manuscript.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Abad G, Ersoy O, Picek S, Urbieta A (2023) Sneaky spikes: Uncovering stealthy backdoor attacks in spiking neural networks with neuromorphic data. arXiv preprint arXiv:2302.06279

2. Campazas-Vega A, Crespo-Martínez IS, Guerrero-Higueras ÁM, Álvarez-Aparicio C, Matellán V, Fernández-Llam C (2023) Analyzing the influence of the sampling rate in the detection of malicious traffic on flow data. Computer Networks 235:109951
3. Huang Y, Li YJ, Cai Z (2023) Security and privacy in metaverse: a comprehensive survey. Big Data Min Anal 6(2):234–247
4. Bringhenti D, Marchetto G, Sisto R, Valenza F (2023) Automation for network security configuration: state of the art and research trends. ACM Comput Surv 56(3):1–37
5. Gowtham M, Pramod HB (2021) Semantic query-featured ensemble learning model for SQL-injection attack detection in IoT-ecosystems[J]. IEEE Trans Reliab 71(2):1057–1074
6. Qu Z, Ling X, Wang T, Chen X, Ji S, Wu C (2024) AdvSQLi: Generating adversarial SQL injections against real-world WAF-as-a-service. IEEE Trans Inf Forens Sec 19:2623–2638
7. Arasteh B, Aghaei B, Farzad B, Arasteh K, Kiani F, Torkamanian-Afshar M (2024) Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. Neural Comput Appl 36(12):6771–6792
8. Wang J, Liu Q, Song B (2022) Blockchain-based multi-malicious double-spending attack blacklist management model. J Supercomput 78(12):14726–14755
9. Ullah K, Rashid I, Afzal H, Iqbal MMW, Bangash YA, Abbas H (2020) SS7 vulnerabilities-a survey and implementation of machine learning vs rule based filtering for detection of SS7 network attacks. IEEE Commun Surv Tutor 22(2):1337–1371
10. Zhang B, Ren R, Liu J, Jiang M, Ren J, Li J (2024) SQLPsdem: A proxy-based mechanism towards detecting, locating and preventing second-order SQL injections. IEEE Trans Software Eng 50(7):1807–1826
11. Dora JR, Hluchý L, Nemoga K (2023) Ontology for blind SQL injection. Comput Inf 42(2):480–500
12. Abdulganiyu OH, Tchakoucht TA, Saheed YK, Ahmed HA (2025) XIDINTFL-VAE: XGBoost-based intrusion detection of imbalance network traffic via class-wise focal loss variational autoencoder. J Supercomput 81(1):1–38
13. Jimoh A, Ahmed MK, Salihu S, Mod B, Salihu MN (2024) Enhancing web security through comprehensive evaluation of SQL injection detection models. Development 23:25
14. Bakhsh SA, Khan MA, Ahmed F, Alshehri MS, Ali H, Ahmad J (2023) Enhancing IoT network security through deep learning-powered intrusion detection system. Internet of Things 24:100936
15. Kumar A, Dutta S, Pranav P (2024) Analysis of SQL injection attacks in the cloud and in WEB applications. Sec Privacy 7(3):e370
16. Yang GY, Wang F, Gu YZ, Teng YW, Yeh KH, Ho PH, Wen WL (2024) TPSQLi: Test prioritization for SQL injection vulnerability detection in web applications. Appl Sci 14(18):8365
17. Souza MS, Ribeiro SE, Lima VC, Cardoso FJ, Gomes RL (2024) Combining regular expressions and machine learning for SQL injection detection in urban computing. J Internet Ser Appl 15(1):103–111
18. Pour MS, Nader C, Friday K, Bou-Harb E (2023) A comprehensive survey of recent internet measurement techniques for cyber security [J]. Comput Sec 128:103123
19. Chakir O, Rehaimi A, Sadqi Y, Krichen M, Gaba GS, Gurtov A (2023) An empirical assessment of ensemble methods and traditional machine learning techniques for web-based attack detection in industry 5.0 [J]. J King Saud Uni Comput Inf Sci 35(3):103–119
20. Liu Y, Dai Y (2024) Deep learning in cybersecurity: a hybrid BERT-LSTM network for SQL injection attack detection. IET Inf Sec 1:5565950
21. Thalji N, Raza A, Islam MS, Samee NA, Jamjoom MM (2023) AE-Net: Novel Autoencoder-Based Deep Features for SQL injection attack detection [J]. IEEE access 11:135507–135516
22. Alghawazi M, Alghazzawi D, Alarifi S (2023) Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model. Mathematics 11(15):3286
23. Zhang W, Li Y, Li X, Shao M, Mi Y, Zhang H, Zhi G (2022) Deep neural network-based SQL injection detection method. Sec Commun Netw 2022:4836289
24. Paul A, Sharma V, Olukoya O (2024) SQL injection attack: detection, prioritization & prevention [J]. J Inf Sec Appl 85:103871
25. Ayad AG, Sakr NA, Hikal NA (2024) A hybrid approach for efficient feature selection in anomaly intrusion detection for IoT networks. J Supercomput 80(19):26942–26984
26. Gu H, Zhang J, Liu T, Hu M, Zhou J, Wei T, Chen M (2019) DIAVA: a traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. IEEE Trans Reliab 69(1):188–202
27. Shar LK, Tan HBK (2012) Defeating SQL injection. Computer 46(3):69–77

28. Song X, Zhang Y, Zhang W, He C, Hu Y, Gong D (2024) Evolutionary computation for feature selection in classification: a comprehensive survey of solutions, applications and challenges. Swarm Evol Comput 90:101661
29. Amin AA, Iqbal MS, Shahbaz MH (2024) Development of intelligent fault-tolerant control systems with machine learning, deep learning, and transfer learning algorithms: a review. Expert Syst Appl 238:121956
30. Mei Z, Zhao T, Xie X (2024) Hierarchical fuzzy regression tree: a new gradient boosting approach to design a TSK fuzzy model. Inf Sci 652:119740
31. Gao K, Xu L (2024) Novel strategies based on a gradient boosting regression tree predictor for dynamic multi-objective optimization. Expert Syst Appl 237:121532
32. Xiao Y, Zou C, Chi H, Fang R (2023) Boosted GRU model for short-term forecasting of wind power with feature-weighted principal component analysis. Energy 267:126503
33. Theng D, Bhoyar KK (2024) Feature selection techniques for machine learning: a survey of more than two decades of research. Knowl Inf Syst 66(3):1575–1637
34. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
35. Yu Y, Si X, Hu C, Zhang J (2019) A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput 31(7):1235–1270

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Qiurong Qin[1] · Yueqin Li[1] · Yajie Mi[1] · Jinhui Shen[1] · Kexin Wu[1] · Zhenzhao Wang[1]

✉ Yueqin Li
xxtyueqin@buu.edu.cn

Qiurong Qin
20221081210209@buu.edu.cn

Yajie Mi
20211081210206@buu.edu.cn

Jinhui Shen
13488896603@163.com

Kexin Wu
15238509174@163.com

Zhenzhao Wang
13562284198@163.com

[1] Smart City College, Beijing Union University, Beijing 100101, China