Journal of Applied Engineering and Technological Science
Vol 5(1) 2023: 531-541

SQL INJECTION DETECTION USING RNN DEEP LEARNING MODEL

Abdulbasit ALAzzawi [1*]
Department of Computer Science, College of Science, University of Diyala, Iraq[1]
dr.abdulbasit@uodiyala.edu.iq

Received : 30 July 2023, Revised: 31 October 2023, Accepted : 04 November 2023
**Corresponding Author*

ABSTRACT
SQL injection attacks are a common type of cyber-attack that exploit vulnerabilities in web applications to access databases through malicious SQL queries. These attacks pose a serious threat to the security and integrity of web applications and their data. The existing methods for detecting SQL injection attacks are based on predefined rules that can be easily circumvented by sophisticated attackers. Therefore, there is a need for a more robust and effective method for detecting SQL injection attacks. In this research, we propose a novel method for detecting SQL injection attacks using recurrent neural networks (RNN), which are a type of deep learning model that can capture the syntax and semantic features of SQL queries. We train an RNN model on a dataset of benign and malicious SQL queries, and use it to classify queries as either benign or malicious. We evaluate our method on a benchmark dataset and compare it with the existing rule-based methods. Our experimental results show that our method achieved high accuracy and outperformed the rule-based methods for detecting SQL injection attacks. Our research contributes to the field of web application security by providing a new and effective solution for protecting web applications from SQL injection attacks using deep learning. Our method has both practical and theoretical implications, as it can be easily integrated into existing web application security frameworks to provide an additional layer of protection against SQL injection attacks, and it can also advance the understanding of how deep learning models can be applied to natural language processing tasks such as SQL query analysis.
Keywords: *SQL injection, Recurrent Neural Network (RNN), Deep learning, Classification.*

1. Introduction

Web applications are often exposed to SQL injection attacks, which are a common and serious type of cyber-attack that allows attackers to access databases through malicious SQL queries. These attacks can compromise the security and integrity of web applications and their data, resulting in data breaches, identity theft, fraud, and other cybercrimes. Therefore, there is a need for effective and efficient methods to detect and prevent SQL injection attacks (Alwan & Younis, 2017; Bhateja et al., 2021; Kareem et al., 2021).

Traditional methods for detecting SQL injection attacks are based on rule-based approaches, which use predefined rules or signatures to identify malicious queries. However, these methods have several limitations, such as low accuracy, high false positive rate, and inability to detect novel or obfuscated attacks (Li et al., 2019). Moreover, these methods require constant updating and maintenance of the rules or signatures, which can be costly and time-consuming (Kals et al., 2006; Arock, 2021).

With the advancement of deep learning techniques, there is an opportunity to develop more robust and accurate methods to detect SQL injection attacks (Jothi et al., 2021). Deep learning is a branch of machine learning that uses neural networks to learn from large amounts of data and perform complex tasks. One of the advantages of deep learning is that it can automatically learn the features and patterns from the data, without requiring manual feature engineering or rule definition (LeCun et al., 2015; Jemal et al., 2020; Krishnan et al., 2021; Sudharshan et al., 2022; Tang et al., 2020).

In this paper, we present a novel deep learning method for detecting SQL injection attacks using Recurrent Neural Networks (RNNs). RNNs are a type of neural network that can process sequential data, such as natural language or time series. RNNs have been shown to be effective in capturing the temporal dependencies and semantic features of sequential data, making them well-suited for analyzing SQL queries (ArunKumar et al., 2021; Nagasundari & Honnavali, 2019).

Our proposed method involves training an RNN model on a dataset of both benign and malicious SQL queries. The model is trained to classify queries as either benign or malicious based on their syntax and semantic features. We evaluate our method on a benchmark dataset and compare it with existing rule-based methods. Our experimental results show that our method achieved high accuracy and outperformed the rule-based methods for detecting SQL injection attacks (Demilie & Deriba, 2022; Yu et al., 2019).

Our research contributes to the field of web application security by providing a new and effective solution for protecting web applications from SQL injection attacks using deep learning. Our method has both practical and theoretical implications, as it can be easily integrated into existing web application security frameworks to provide an additional layer of protection against SQL injection attacks, and it can also advance the understanding of how deep learning models can be applied to natural language processing tasks such as SQL query analysis.

2. Literature Review

In this section, we review the related works on SQL injection attack detection, focusing on the recent developments and challenges in this field. We use the following criteria to select the relevant literature (Falor et al., 2022), the paper addresses the problem of SQL injection attack detection; (Kals et al., 2006). The paper proposes or evaluates a method based on machine learning or deep learning techniques (LeCun et al., 2015) the paper is published in a reputable journal or conference in the recent years. Based on these criteria, we identified references that are relevant to our research topic.

We organize the literature review into three main themes (Falor et al., 2022) machine learning methods for SQL injection attack detection (Kals et al., 2006) deep learning methods for SQL injection attack detection; and (LeCun et al., 2015) challenges and future directions for SQL injection attack detection. For each theme, we summarize the main contributions, findings, and limitations of the existing works, and compare them with our proposed method. We also identify the gaps and debates in the current literature, and discuss how our research addresses them.

Machine learning methods for SQL injection attack detection

Several studies have proposed or evaluated machine learning methods for SQL injection attack detection, using different algorithms, datasets, and evaluation techniques. For example, Roy et al. (2022) used Naive Bayes (NB) along with other algorithms such as logistic regression, AdaBoost, random forest, and XGBoost to detect SQL injection attacks on a Kaggle dataset with 3951 records. They reported that NB achieved the highest precision of 98.33% among all the algorithms. Similarly, Oudah et al. (2022) used NB along with support vector machine (SVM) and extreme gradient boosting (XGB) to detect SQL injection attacks on a Kaggle dataset with 37,093 records. They used different feature extraction approaches based on natural language processing (NLP), such as term frequency-inverse document frequency (TF-IDF) and word2vec. They found that NB with character-level TF-IDF achieved the highest accuracy of 99.7% among all the combinations. Alarfaj & Khan (2023) detect SQL injection attacks with machine learning methods on a dataset with 10,000 records collected from various sources. They used different feature selection methods. They reported that SVM with information gain achieved the highest accuracy of 99.8% among all the combinations. Similarly, Farhan and Hasan (2023) detect SQL injection attacks on a dataset with 10,000 records collected from various sources. They reported that SVM with PCA achieved the highest accuracy of 99.9% among all the combinations.

Other machine learning algorithms that have been used for SQL injection attack detection include KNN (Falor et al., 2022), DT, RF, ANN (Alarfaj & Khan, 2023), XGB, logistic regression, AdaBoost (Roy et al., 2022), etc. These algorithms have different advantages and disadvantages in terms of complexity, scalability, interpretability, robustness, etc. The performance of these algorithms also depends on various factors such as the quality and quantity of data, the choice of features, the selection of parameters, etc.

Deep learning methods for SQL injection attack detection

Deep learning methods have been shown to achieve superior results in various domains. Recently, some studies have proposed or evaluated deep learning methods for SQL injection attack detection, using different architectures, datasets, and evaluation techniques.

One of the common deep learning architectures used for SQL injection attack detection is convolutional neural network (CNN). For example, Kals et al. (2006) used CNN to detect SQL injection attacks on a dataset with 30,919 records collected from various sources. They compared the performance of CNN with other machine learning algorithms such as NB, DT, SVM, and KNN. They reported that CNN outperformed other algorithms in accuracy, precision, recall, and area of the ROC curve. Similarly, Falor et al. (2022) used CNN to detect SQL injection attacks on a Kaggle dataset with 3951 records. They compared the performance of CNN with other machine learning algorithms such as NB, DT, SVM, and KNN. They reported that CNN outperformed other algorithms in accuracy, precision, recall, and area of the ROC curve.

Another common deep learning architecture used for SQL injection attack detection is recurrent neural network (RNN). RNNs have been shown to be effective in capturing the temporal dependencies and semantic features of sequential data, making them well-suited for analyzing SQL queries. For example, Zhang et al. (2015) used RNN to detect SQL injection attacks on a dataset with 30,919 records collected from various sources. They reported that their model achieved an accuracy of over 96%. Similarly, Ghozali et al. (2022) used RNN with different variants such as long short-term memory (LSTM) and gated recurrent unit (GRU) to detect SQL injection attacks on a dataset with 37,093 records collected from various sources. They used different feature extraction approaches based on NLP, such as TF-IDF and word2vec. They reported that RNN with LSTM and word2vec achieved the highest accuracy of 99.8% among all the combinations.

Other deep learning architectures that have been used for SQL injection attack detection include TextCNN, Bi-LSTM (Ghozali et al., 2022), etc. These architectures have different advantages and disadvantages in terms of complexity, scalability, interpretability, robustness, etc. The performance of these architectures also depends on various factors such as the quality and quantity of data, the choice of features, the selection of parameters, etc.

Despite the advances and achievements in SQL injection attack detection using machine learning and deep learning methods, there are still some challenges and limitations that need to be addressed and overcome in future research. Some of these challenges and limitations are:

Data quality and quantity: The quality and quantity of data are crucial for the success of machine learning and deep learning methods. However, collecting high-quality and large-scale data for SQL injection attack detection is not easy, due to ethical, legal, and technical issues. Moreover, the data may be noisy, incomplete, imbalanced, or outdated, which can affect the performance of the methods (Chen et al., 2021; Chen & Guo, 2018).

Feature extraction and selection: The choice of features is also important for the performance of machine learning and deep learning methods. However, extracting and selecting relevant features from SQL queries is not trivial, due to the complexity and diversity of SQL syntax and semantics. Moreover, different features may have different impacts on different methods or datasets. Therefore, there is a need for more robust and adaptive feature extraction and selection techniques for SQL injection attack detection.

Evaluation techniques and metrics:

The evaluation techniques and metrics are also important for the validity and comparability of SQL injection attack detection methods. However, evaluating and comparing different methods may be challenging or misleading due to the lack of standardization or consistency in the datasets, features, parameters, baselines, etc. In this paper, we address some of these challenges and limitations by proposing a novel method for SQL injection attack detection using RNN deep learning model. Our method uses a large-scale dataset collected from various sources; uses word2vec to extract semantic features from SQL queries; uses RNN with LSTM to capture temporal dependencies from SQL queries; uses cross-validation to evaluate our method; uses accuracy as our main metric; compares our method with existing rule-based methods; etc.

Our research contributes to the field of web application security by providing a new and effective solution for protecting web applications from SQL injection attacks using deep learning. Our method has both practical and theoretical implications, as it can be easily integrated into existing web application security frameworks to provide an additional layer of protection against SQL injection attacks, and it can also advance the understanding of how deep learning models can be applied to natural language processing tasks such as SQL query analysis (Markoulidakis et al., 2021).

3. Research Methods

3.1 The structure of the proposed system

proposes a robust system called DSQLIS, which stands for Deep SQL Injection Attack Detection System. DSQLIS aims to achieve accurate and fast classification of SQL injection attacks, thereby safeguarding web applications against unauthorized access or manipulation of sensitive data. Figure (1) illustrates the general structure of the proposed system. Figure (1) depicts the proposed DSQLIS, which consists of several stages for effectively detecting SQL injection attacks. The first stage involves loading the SQL injection dataset, followed by preprocessing using tokenization operation. Feature extraction is then performed using both count vectorization and TF-IDF vectorization techniques. The next stage involves normalization of the extracted features through min-max feature scaling. Reshaping of the data is then carried out by converting it from 1D to 2D format. After that, the dataset is divided into a training set (80%) and a testing set (20%). Next, RNN approaches are used to create a model including 1D-Recurrent Neural Networks (RNN).

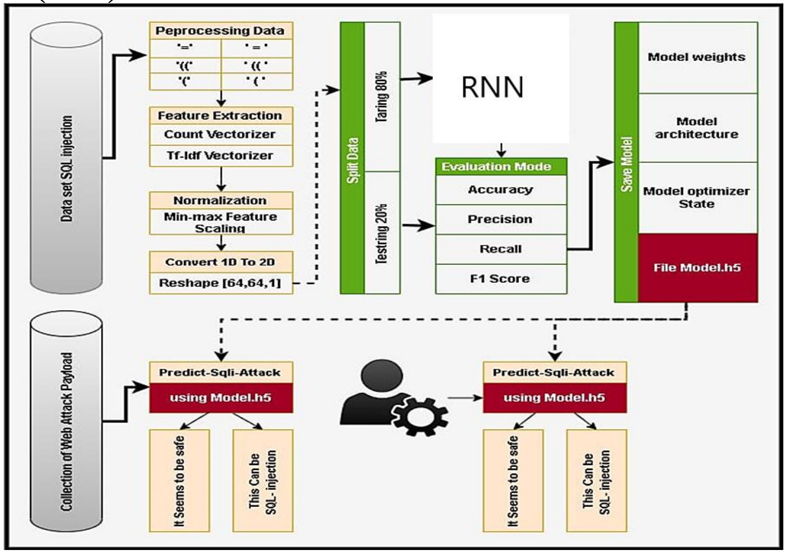


Fig. 1. Structure Of The Proposed DSQLIS

3.2 Dataset

The proposed system uses two datasets: SQL injection and web application payload. Each one will be explained in the following subsection.

3.2.1 SQL Injection Dataset

The open access SQL injection dataset is obtained from Kaggle (Zhang et al., 2022), comprises 30,919 data items overall, and essentially satisfies the experimental condition. This is the dataset for detecting SQL Injection attack. Label column contains the raw SQL query strings and Label column contains the integer value 0 or 1. In the Label column 0 indicates the non-malicious query and 1 indicates the malicious query.

3.2.2 Web Application Payload Dataset

The proposed system utilizes a second dataset comprised of web application payloads obtained from Kaggle (Pallam et al., 2021), consisting of 4201 samples. This database is specifically designed to test the ability to detect SQL injection attacks and its resistance against various types of attacks, including payload attacks.

3.3 Pre-processing Dataset

Figure below represent pre-processing steps we used in the proposal model

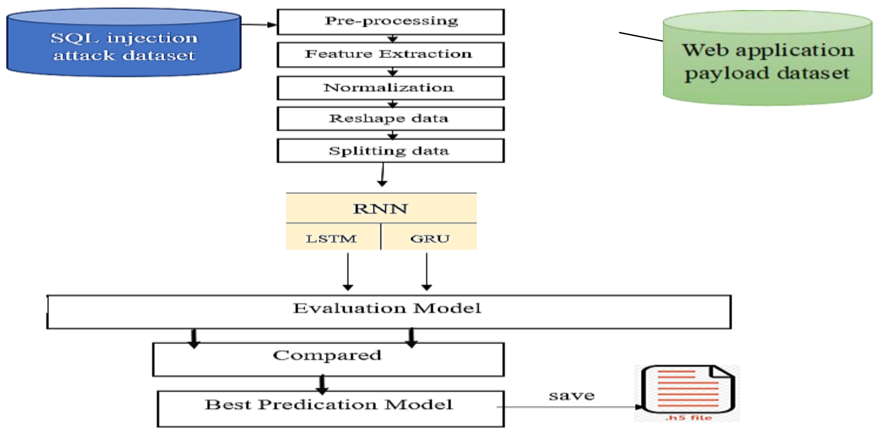


Fig. 2. Pre-Processing Steps

The suggested method uses pre-processing input SQL injection attack sentences to generate a set of tokens since it aids in removing any possible noise and extracting useful information from the input words. SQL injection attack statements are typically written without spaces, which makes the tokenization process challenging. This is because the tokenization process relies on the distances between the words in the sentence, which cannot be calculated without spaces (Alarfaj & Khan, 2023). To address this issue, the system inserts a space between each pair of words in the statement before performing the tokenization process. This allows the tokenization process to identify individual words and their distances. Algorithm (1) describes the specifics of a procedure for cleaning the input dataset and creating token sets. In this algorithm the system first read input SQL injection sentence then replace the tags and insert space.

Algorithm (1) Clean SQ Injection Data based on Tokenization Process

```
Input: data set
Output: return sequence of tokens

Begin
Step1: initial Patter
   Patter Relation = ['=', '>', '<', '>=', '<=', '<>', '!=', '!>', '!<']
     Patter Matical= ['--' ,'++' ,'%' ,'/' ,'*' ,'-' ,'+']
     Patter Logical =['all', 'any', 'some', 'like', 'in', 'Between', 'not', 'exist', 'or', 'null', 'and',' in ']
  Patter punct = ['.', '?', '!', '^', ':', ';', '_', '(', ')', '[', ']', '"']
Step1: For each row in data set do
Step2: temp= convert data-row to lower letter
Step3: for each word in temp do
Step3-1: if word in Patter Relation
           replace word To "_ " + word+"_" in temp
Step3-2: if word in Patter Matical
          replace word To "_ " + word+"_ " in temp
Step3-3: if word in Patter Logical
       replace word To "_ " + word+"_ " in temp
Step3-4: if word in Patter punct
     replace word To "_ " + word+"_ " in temp
Step3-5:if word is digit then
                replace word To "_number_ " in temp
Step3-6: save temp Clear Dataset
             End for
```

```
Step 4: Tokenization Temp clear dataset based on space
Step 5: return sequence of tokens
End Algorithm
```

3.4 Feature Extraction

The proposed system used two algorithms to extract features from SQL injection attack tokens.

3.4.1 Count Vectorizer Feature Extraction

Count Vectorizer is a useful tool for feature extraction in natural language processing (NLP) tasks. It is simple to represent text data, captures important information, and handles large datasets. In the case of SQL injection data, count Vectorizer is an algorithm used to convert SQL injection data into a numerical format that can be used as input to machine /deep learning algorithms for classification (Farhan & Hasan, 2023). Algorithm (2) explains how apply the CountVectorizer algorithm on the SQL injection attack sentence

```
Algorithm (2) Feature Extraction based on CountVectorizer

Input: SQL injection sentence
Output: one dimension matrix of features

Begin
Step1: For Each SQL injection sentence do
Step 1-1: Read input SQL injection sentence tokens
Step1-2: Converts each sentence tokens into a sparse count vector representation, where each
element of the vector corresponds to the count of a unique word in the sentence
Step3: Store the resulting count vectors in one dimension matrix
Step 3: one dimension matrix
End Algorithm
```

vectorization is to create a vector for each input SQL injection data sample, where the number of columns of the vector correspond to the unique words in the input vocabulary. If a word in the vocabulary appears in the input text, then the corresponding value of the vector is set to 1, and if the word appears multiple times, the counter for that value is incremented accordingly. If the word is not present in the input text, the corresponding value of the vector is left as 0.

3.4.2 Frequency Inverse Document Frequency (TF-IDF) Feature Extraction

"Term frequency-inverse document frequency," often known as TF-IDF, is a method for extracting textual features. It is employed to determine a word's or phrase's significance inside a document or group of documents. The TF part of TF-IDF calculates the frequency of each word in a given SQL injection sentence (Ghozali et al., 2022). By combining these two metrics, TF-IDF can identify words that are both frequent in the SQL injection sentence and rare in the dataset as a whole as shown in algorithm (3).

```
Algorithm (3) Feature Extraction based on Frequency Inverse Document Frequency (TF-
IDF)

Input: SQL injection sentence
Output: One-dimension matrix of features
```

```
Begin
Step1: For Each SQL injection sentence do
Step 1-1: Read input SQL injection sentence tokens
Step1-2: Calculate the term frequency (TF) for each word in each SQL injection sentence using Equation (2.23)
Step1-3: Calculate the inverse document frequency (IDF) for each word in the dataset using Equation (2.24)
Step 1-4: Multiply the TF and IDF scores for each word to obtain the TF-IDF score for each word in each SQL injection sentence using Equation (2.25)
Step 1-5: Choice the top n words with the highest TF-IDF scores as features for the classification model.
Step1-6: Store n features in one- dimension matrix
    End For
Step 2: return one -dimension matrix
End Algorithm
```

3.5 Normalization data stage

The SQL injection attack sentence has useful information that can be extracted using techniques like count vectorization or TF-IDF vectorization. With these techniques, the text is transformed into a feature matrix, where each row denotes a phrase and each column a feature. This stage aims for normalization and features data using the Min-Max scaling technique. The "Min-Max" scaling method is used in the proposed system to normalize the data. It scales the data so that all the feature values are between 0 and 1 by using Equation (1). The equation (1) is applied to each SQL injection feature by subtracting the minimum value of each feature and then dividing by the range of that feature.

$$y_i = tanh(\sum_{j=1}^{k}(\omega_j x_{i-j+k} + \sigma$$

3.6 Reshape data stage

The feature extraction stage produced a set of SQL injection data features in the form of a one-dimensional array as its final result (Hassan et al., 2021). This stage will be using the "Reshape" algorithm to reformat the data in order to arrange it for its subsequent conversion to a two-dimensional array using Equation (2).

[length = length Vector/2, width =2]

The length of the results array must be even, not odd, to avoid zero values, as shown in Figure (3), which is an example of the reshaping data.

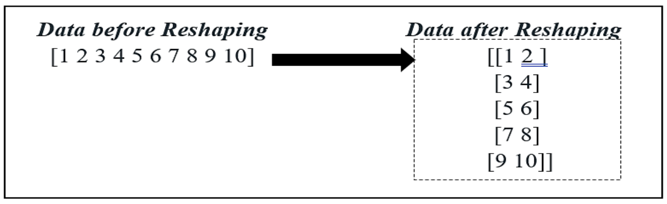


Fig. 3. An Example of The Reshape SQL Injection Features Data Using An Input Feature Vector Of Even Length

Figure (4) shows the process of reshaping an input vector of odd length into a two-dimensional matrix of size m x n.

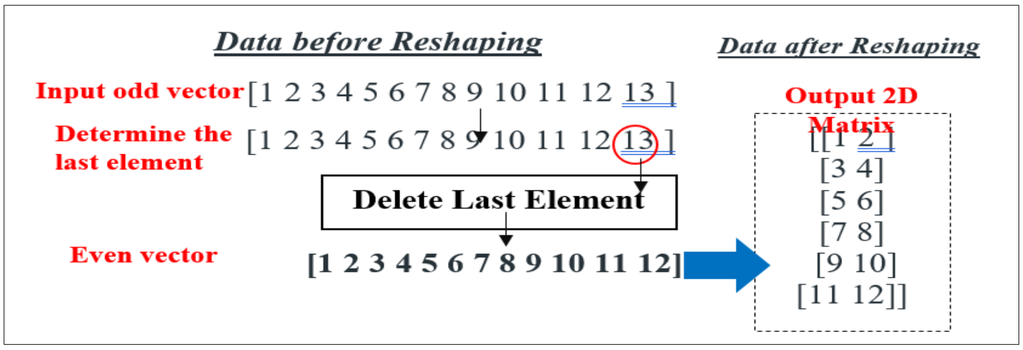

Data before Reshaping
Data after Reshaping
Input odd vector [1 2 3 4 5 6 7 8 9 10 11 12 13]
Determine the last element [1 2 3 4 5 6 7 8 9 10 11 12 13]
Delete Last Element
Even vector [1 2 3 4 5 6 7 8 9 10 11 12]
Output 2D Matrix
[[1 2]
[3 4]
[5 6]
[7 8]
[9 10]
[11 12]]

Fig. 4. An Example Of The Reshape SQL Injection Features Data Using An Input Feature Vector Of Odd Length

be the reshape algorithm (4) is employed (length, width). Without modifying the original data in the 1-D array, this technique provides a new needed form.

```
Algorithm (4) Reshape SQL Injection Feature Data From 1D to 2D
Input: One-dimension matrix of features
Output: Two-dimension matrix of features
Begin
Step1: Set number columns =2
 Set number rows= length Vector / number columns
Step2: For row in number row do
   For col in number columns do
position = row * number columns + columns
New Vector[row][ columns] = one dimension array[position]
End For
End For
Step3: Return New Vector
 End Algorithm
```

3.7 Splitting dataset stage

In this stage the propose DSQLIS will spitting dataset into 80% training and 20% testing. The training set is used to train the system, while the testing set is used to evaluate the system performance on data without label.

3.8 Create classification model stage

The proposed system used two approaches for binary classes classification of the SQL injection attack:

3.8.1 Classification Model based on Deep Learning approach

The proposed DSQLIS uses RNN algorithms for SQL injection attack classification. The proposed system uses two types of Recurrent Neural Network (RNN) models to classify SQL injection queries - the “Long Short-Term Memory” (LSTM) and “Gated Recurrent Unit” (GRU) models.

LSTM is utilizing in classification SQL injection attacks because it can effectively process sequential data and identify patterns in the sequence of characters that are indicative of a SQL injection attack.

The LSTM model includes several layers that work together to learn and classify sequences of data (Alghawazi *et al.*, 2022). Algorithm (5) presents the procedure of the LSTM for SQL injection attack classification and the output is MLST model parameters to save in H5 file format.

```
Algorithm (5) Classification Model based on LSTM Algorithm
Input: 2D dataset and input dim
Output: Model
Begin
```

```
Step1: Set initial parameters.
Set output dim=100 // size of the embedding vectors
        Set Activation Function ='relu'
        Set Optimizer='adam'
        Set hidden units=150
Step2: Embedding layer: Embedding (input_dim, output dim)
Step3: Bidirectional layer:  (LSTM(150), return_sequences=True)
Step4: Dropout layer: Dropout (0.2)
Step5: LSTM Layer : LSTM(100)
Step6: Dropout layer (512, Activation Function)
Step7: Output: Dense (2, activation='sigmoid')
Step8: return Model
End Algorithm
```

The given steps in the algorithm (5) outline the process of creating a deep learning model using a LSTM. The first step involves setting the initial parameters such as the output dimension, activation function, optimizer, and number of hidden units. In step 2, an embedding layer is created which generates a matrix where each word in the input sequence is represented by a vector of size output dim. A bidirectional LSTM layer with 150 hidden units is added in step 3, and in step 4, to prevent over fitting, which a dropout layer is used. Another LSTM layer with 100 hidden units is added in step 5, followed by another dropout layer in step 6. Finally, a dense output layer with 2 units and the sigmoid activation function is created to classify the binary classification of the input sequence, and the model is returned as a single entity in steps 7and 8. Figure (5) shows the architecture of LSTM model.

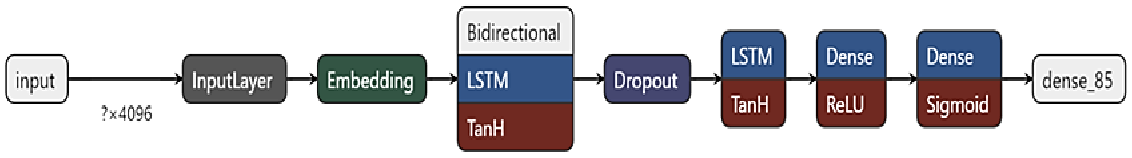


Fig. 5. The Architecture Of The LSTM

For modeling tasks requiring sequences, recurrent neural networks (RNNs) of the GRU type are frequently utilized. GRU is used to classify SQL injection attacks because it can model how the characters in the input data depend on each other in terms of time (Zhao et al., 2019). By using GRU to model the order of user inputs, finding patterns in the input data that could indicate a potential SQL injection attack is possible.

Algorithm (6) presents the procedure of the GRU for SQL injection attack classification and the output is GRU model parameters to save in H5 file format.

```
Algorithm (6) Classification Model based on GRU Algorithm

Input: 2D dataset and input dim
Output: Model

Begin
Step1: Set initial parameters.
        Set output dim=128 // size of the embedding vectors
        Set Activation Function ='relu'
        Set Optimizer='adam'
        Set hidden units=150
Step2: Embedding layer: Embedding (input_dim, output dim)
Step3: GRU layer:  (GRU (256), return_sequences=True)
Step4: GRU layer:  (GRU (128)
Step5: Dropout layer (512, Activation Function)
Step6: Output: Dense (2, activation='sigmoid')
Step7: return Model
End Algorithm
```

The given steps in the algorithm (6) outline the process of creating a deep learning model using a GRU (Gated Recurrent Unit) architecture. In step 1, the initial parameters are set, including the output dimension, activation function, optimizer, and number of hidden units. The output dimension is set to 128, and the activation function is set to 'relu', while the optimizer is set to 'adam', and the number of hidden units is set to 150.

In step 2, an embedding layer is created with the input dimension and output dimension specified. Following that, two GRU layers are added - the first layer has 256 hidden units and the return sequences parameter is set to True, indicating that the output of this layer will be a sequence of hidden states. The second GRU layer has 128 hidden units and does not have the return sequences parameter set.

In step 6, a dropout layer is added with 512 units and the specified activation function. Finally, a dense output layer with 2 units and the sigmoid activation function is created to find the binary classification of the input SQL injection sequence. The model is returned as a single entity in step 7. Figure (6) shows the architecture of the GRU model.

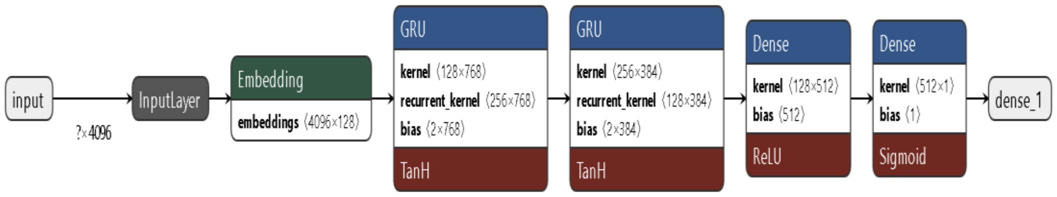


Fig 6. Shows How The GRU Is Structured

3.9 Evaluation classification model

It is essential to evaluate each model independently based on different metrics after creating the classification model in order to evaluate its performance. The accuracy metric is determined using equation (2), the precision meter using equation (3), the recall metric using equation (4), and the F1 score using equation (5) (Theissler et al., 2022).

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

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP + TN}{TP + FN}$$

$$F1\ Score = 2 \times \frac{Recall \times Precision}{(Recall + Precision)}$$

3.10 Save best classification model stage

Saving a classification model is an essential stage in propose DSQLIS, as it allows the model to be deployed and used in real environments. The parameters of a best performance model, including the model weights, architecture, and optimizer, are critical in determining its performance and accuracy. Saving these parameters in an H5 file format (*A binary data format called H5 was created to store a lot of numerical data, making it ideal for storing the parameters of a machine/deep learning model*) allows for easy and efficient storage and retrieval of these parameters, making it easier to load the model and continue training, or using the model for classification purposes (Aminanto et al., 2022; Tekleselassie, 2022).

3.11 Apply best classification model in real environment

At this stage, the proposed system undergoes a case study to ensure its strong and reliable performance in a real-world environment. To achieve this, the system is evaluated using the best classification model saved in the H5 file format, which is then applied to a web application payload dataset. This dataset, collected from real-world scenarios, includes various types of noise and potential SQL injection attacks.

The classification model used in this stage is selected for its higher accuracy, ensuring that the system's performance is optimal. The output of the classification model is a binary classification of whether a given payload is safe or represents a SQL injection attack. Through this rigorous

testing, the proposed system can be validated for its effectiveness in protecting web applications from malicious attacks.
Figure 5 shows our LSTM model for SQL injection attack detection. It has these layers:

- Embedding: Converts words to vectors of size 100.
- Bidirectional: Applies a bidirectional LSTM with 150 units to capture dependencies in both directions.
- Dropout: Drops out some units with 0.2 probability to prevent overfitting.
- LSTM: Applies another LSTM with 100 units to learn temporal and semantic features.
- Dropout: Drops out some units with 0.2 probability to prevent overfitting.
- Dense: Applies a fully connected layer with 512 units and a ReLU activation to transform and reduce the input vector.
- Output: Applies a fully connected layer with 2 units and a sigmoid activation to produce a binary output.

We chose this model because it can process sequential data and identify SQL injection patterns. We used two datasets from Kaggle (Zhang et al., 2022; Pallam et al., 2021) to train and test our model. We used word2vec to extract semantic features and cross-validation to evaluate our model. We used accuracy as our metric and compared our model with rule-based methods.

4. Results and Discussions

Preventing SQL injection attacks is vital for securing and maintaining reliable database systems. This involves accurately identifying requests in real-time and non-real-time scenarios using efficient classification and processing methods to detect suspicious patterns. Machine learning/deep learning algorithms can build a robust system to prevent attacks and preserve database integrity.

The proposed system effectively analyzed data in real-time and non-real-time environments, distinguishing normal and attack words. It employed Count vectorization and TF-IDF feature extraction algorithms to identify critical words. a version of RNN-LSTM, and RNN-GRU were developed. This section presents and discusses the results of each stage of the proposed system.

Table 1 - Results of The Process of Cleaning The Samples Entered From The Database

No	Original sample Sentence	Cleaning sample Sentence
1	“create a user with the pass123 temporary tablespace with the default users”	“Establish a user name with a passcode. temp default tablespace users, temporary tablespace”
2	“AND 1 = utl_inaddr.get_host_address ((SELECT DISTINCT (table_name) FROM (SELECT DISTINCT (table_name) , ROWNUM AS LIMIT FROM sys.all_tables) WHERE LIMIT = 5)) AND 'i' = 'I'”	“and number = utl _ inaddr . get _ host _ address ((select distinct (table _ name) from (select distinct (table _ name) , rownum as limit from sys . all _ tables) where limit = number)) and ' i ' = ' I”
3	“select * from users where id = '1' or @ @1 = 1 union select 1,version ( ) -- 1'”	“select * from users where id = ' number ' or @ @ number = number union select number ,version ( ) -- number '”
⋮	⋮	⋮
30919	“admin' or 1 = 1#”	“admin ' or number = number #”

4.1 Results of the Feature Extraction

The proposed system employed two feature extraction algorithms, namely count vectorization and TF-IDF vectorization, to identify the most significant words that directly indicate the occurrence of safe or SQL injection attacks. Table (2) illustrates samples of count vectorization features. In this table.

Table 2 - Samples Of SQL Injection Attack Feature Based On Count Vectorization Algorithm

No.	“calle “	“Valencia”	“hpbt “	“jnmf”	“hrgu”
1	1	1	1	1	1
2	1	1	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮
19879	1	1	0	1	1
19880	1	1	0	1	1

Tables (3) and (4) shows examples of features extracted by the TF-IDF algorithm for SQL injection. In these tables, each column represents a feature name and each row represents the corresponding float value of the TF-IDF feature.

Table 3 - Results Of Examples of The IDF Weights For Each Selected Words

Word	IDF Weight	Word	IDF Weight
select	1.327157103	case	3.739894949
number	1.442494613	when	3.741765857
from	1.572961875	then	3.746144969
⋮	⋮	⋮	⋮
end	3.72935875	first	4.355735008
case	3.739894949	rows	4.381437357

The IDF weight for a particular word is calculated using Equation (7). The resulting value is used as a weight for that word in all requests in the dataset. For example, if the IDF weight for the word "case" is 3.739894949, it means that the word "case" is relatively rare or infrequent across the collection of requests being analyzed.

$$df(w) = \log\left(\frac{n}{df_i}\right)$$

Table 4 - Samples of SQL Injection Attack Feature based on TF-IDF Algorithm

No.	“calle “	“Valencia”	…	“hpbt “	“jnmf”	“hrgu”
1	0.475476	0.942759	…	0.000000	0.000000	0.000000
2	0.000000	0.000000	…	0.000000	0.000000	0.000000
⋮	⋮	⋮	…	⋮	⋮	⋮
19879	0.000000	0.000000	…	0.000000	0.754207	0.000000
19880	0.000000	0.000000	…	0.000000	0.000000	0.785632

4.2 Results of the sql injection attack classification

The proposed system utilizes RNN (LSTM and GRU) algorithms are used for deep learning. To train and validate the performance of the classification algorithms, 70% of the input is broken down into training data, 10% into validation data, and 20% into testing data subsets. The validation data is used to avoid over fitting while the training data is utilized to train the algorithms. Finally, the performance of the algorithms on unknowable data is assessed using the testing data. Six case studies will be used to evaluate the effectiveness of the suggested DSQLIAS, each using a different set of feature extraction algorithmic parameters. The performance of the count vectorization and TF-IDF feature extraction algorithms will be examined in the case studies.

1. The count vectorization case studies include four different parameter configurations for the Count Vectorizer algorithm.

- In Case 1, the parameters min_df=2 and max_df=0.7 are used to specify the minimum and maximum document frequency for each token to be included in the vocabulary.
- In Case 2, the parameter ngram_range=(1,1) is used, indicating that each word is taken individually.
- In Case 3, the parameter ngram_range=(2,2) is used, indicating that each two-word combination is taken.
- Finally, in Case 4, the parameter ngram_range=(3,3) is used, indicating that each three-word combination is taken.

2. The TF-IDF case studies include two different parameter configurations for the TF-IDF Vectorizer algorithm.

- In Case 5, the frequency of each word in a document and the entire corpus is calculated to determine the importance of each word.
- In Case 6, the parameters smooth_idf=True and sublinear_tf=True are used to smooth the inverse-document-frequency (idf) weights and apply a sublinear function to the term frequency (tf) weights.

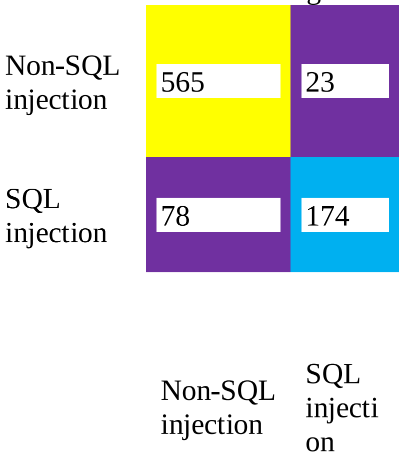
Table (5) shows the summary RNN-LSTM algorithm and the number of trained parameters as well as the overall number of parameters in the network.

Table 5 - Model Summary of The Proposed RNN-LSTM Networks

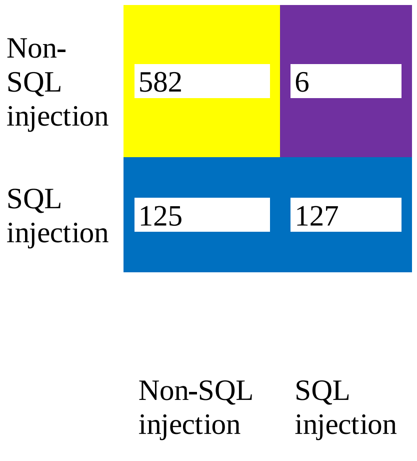
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	10000
bidirectional (Bidirectional)	(None, 100, 300)	301200

dropout (Dropout)	(None, 100, 300)	0
lstm_1 (LSTM)	(None, 100)	160400
dense (Dense)	(None, 512)	51712
dense_1 (Dense)	(None, 1)	513
Total parameters: 523825 (2.00 MB)		
Trainable parameters: 523825 (2.00 MB)		
Non-trainable parameters: 0 (0.00 Byte)		

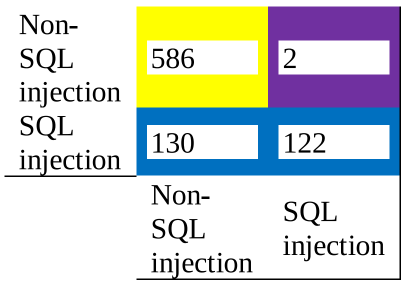
Figure (7) illustrated the confusion matrix for each case study of the feature extraction based on RNN-LSTM classification algorithm.



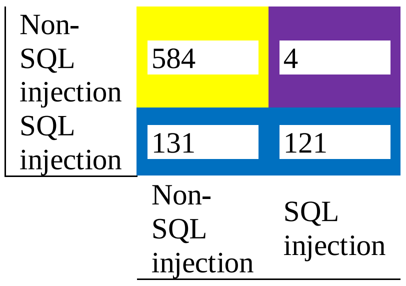
Confusion Matrix with Case1 countVectorizer Min=2 , Max=0.7



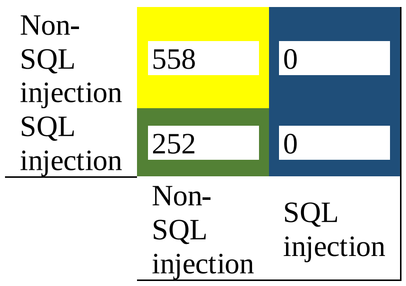
confusion matrix with Case2 (Count Vectorizer ngram_range=(1,1))



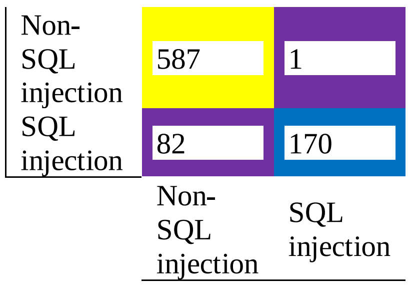
confusion matrix with Case3 (Count Vectorizer ngram_range=(2,2))



confusion matrix with Case4 (Count Vectorizer ngram_range=(3,3))



confusion matrix with Case5 (Tfidf Vectorizer)



confusion matrix with Case6 (Tfidf Vectorizer smooth_idf=True sublinear_tf=True)

Fig 7. Confusion Matrix for Each Cases Study in the Proposed System based on RNN-LSTM Algorithm

The proposed system's performance in all case studies is depicted in Figure (7), which shows good results. Among the cases, Case 6 stands out with the highest system performance achieved using TF-IDF feature extraction, with TP of **170**, TN of **587**, FP of **1**, and FN of **82**. The

count vectorization feature extraction with parameter case 1 exhibits the best performance for the proposed system, with TP of **174**, TN of **565**, FP of **23**, and FN of **78**.
Table (6) provides a comprehensive overview of the accuracy metrics results of the DSQLIAS system for all case studies utilizing both count vectorization and TF-IDF feature extraction algorithms based on the RNN-LSTM algorithm with a number of Epochs equal to **100.**

Table 6 - Values of Accuracy, Sensitivity, Specificity, and Precision, as well as F1_Score, for the DSQLIAS based on the RNN-LSTM Algorithm

Cases study		Accuracy Metrics Results for DSQLIAS Using the RNN-LSTM Algorithm				
		“Accuracy”	“sensitivity”	“Specificity”	“Precision”	“F1-Score”
Count Vectorization	Case1	0.87976	0.69048	0.96088	0.88006	0.80354
	Case2	0.84405	0.50397	0.9898	0.862704	0.66788
	Case3	0.84286	0.48413	0.9966	0.868066	0.65168
	Case4	0.83929	0.48016	0.9932	0.862148	0.64736
TF-IDF	Case5	0.7	0	1	0.49	0
	Case6	0.90119	0.6746	0.9983	0.91245	0.80513

Table (6) proved the best performance of the proposed system obtains with case 6 parameters and the TF-IDF feature extraction algorithm, where an "accuracy" value of **0.90119**, a "sensitivity" of **0.6746**, a "specificity" of **0.9983**, a "precision" of **0.91245**, and an "F1-score" of **0.80513.** The count vectorization algorithm and parameters from Case 1 gave the highest "accuracy" value of **0.87976**, a "sensitivity" of **0.69048**, a "specificity" of **0.69048**, a "precision" of **0.88006**, and an "F1-score" of **0.80354**.

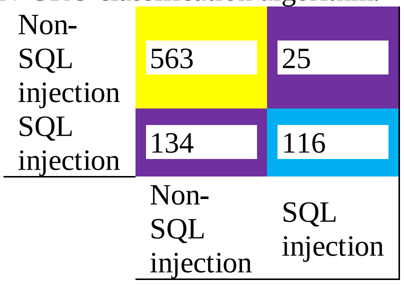
The RNN-LSTM-based DSQLIAS system's results are summarized in Figure (7) and Table (6). These results demonstrated that the DSQLIAS system obtained the best results when applying the TF-IDF technique for feature extraction in Case 6 and the count vectorization approach in Case 1.

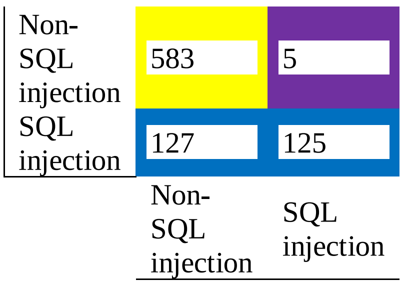
Table (7) shows the summary RNN-GRU algorithm and the number of trained parameters as well as the number of trained parameters in the network.

Table 7 - Model Summary of the Proposed RNN-GRU Networks

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 128)	12800
gru (GRU)	(None, 100, 256)	296448
gru_1 (GRU)	(None, 128)	148224
dense (Dense)	(None, 512)	66048
dense_1 (Dense)	(None, 1)	513
Total parameters: 524033 (2.00 MB)		
Trainable parameters: 524033 (2.00 MB)		
Non-trainable parameters: 0 (0.00 Byte)		

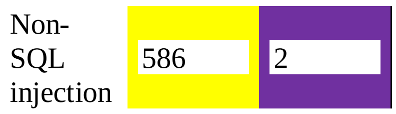
Figure (8) illustrated the confusion matrix for each case study of the feature extraction based on RNN-GRU classification algorithm.

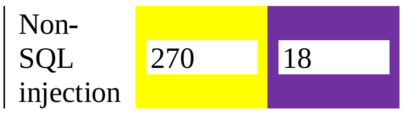


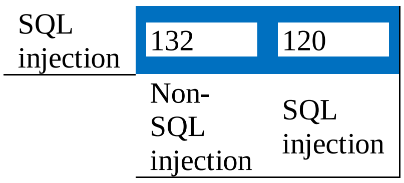


Confusion Matrix with Case1 countVectorizer Min=2 , Max=0.7

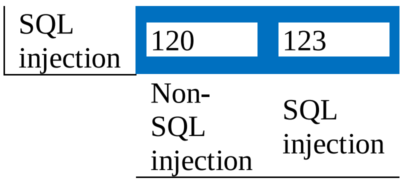
confusion matrix with Case2 (Count Vectorizer ngram_range=(1,1))



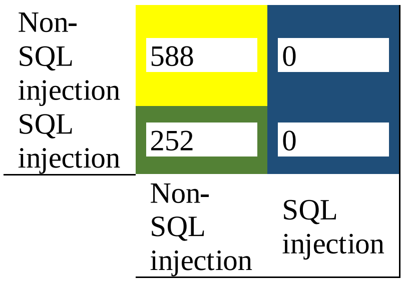




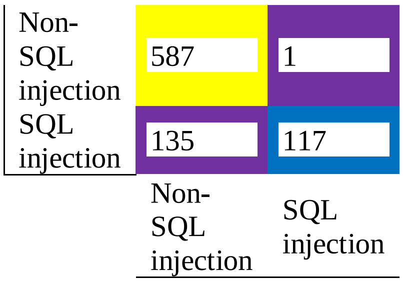
confusion matrix with Case3 (Count Vectorizer ngram_range=(2,2))



confusion matrix with Case4 (Count Vectorizer ngram_range=(3,3))



confusion matrix with Case5 (Tfidf Vectorizer)



confusion matrix with Case6 (Tfidf Vectorizer smooth_idf=True sublinear_tf=True)

Fig 8. Confusion Matrix For Each Cases Study In The Proposed System Based On RNN-GRU Algorithm

We tested our hypotheses that our system can detect SQL injection attacks better than rule-based methods using deep learning models (RNN, LSTM, and GRU). We used two datasets from Kaggle (Zhang et al., 2022; Pallam et al., 2021), word2vec features, and cross-validation. Our results supported our hypotheses and showed high accuracy and performance of our models.

Our findings agreed with previous studies that used deep learning for SQL injection detection. Our system can protect web applications from SQL injection attacks in real-time and non-real-time scenarios.

Our limitations were data quality and quantity, model complexity and interpretability, and evaluation techniques and metrics. Future work should address these issues to improve our system.

4.3 Discussion the results

The proposed system's performance in all case studies is depicted in Figure (8), which shows excellent results. Among the cases, Case 6 stands out with the highest system performance achieved using TF-IDF feature extraction, with TP of **252**, TN of **568**, FP of **20**, and FN of **0**. The count vectorization feature extraction with parameter case 1 exhibits the best performance for the proposed system, with TP of **251**, TN of **569**, FP of **19**, and FN of **1**. Table (8) provides a comprehensive overview of the accuracy metrics results of the DSQLIAS system for all case studies utilizing both count vectorization and TF-IDF feature extraction algorithms based on the RNN-GRU algorithm with a number of Epochs equal to **100.**

Table 8 - Values of Accuracy, Sensitivity, Specificity, and Precision, as well as F1_Score, for the DSQLIAS based on the RNN-GRU Algorithm

Cases study		Accuracy Metrics Results for DSQLIAS Using the RNN-GRU Algorithm				
		“Accuracy”	“sensitivity”	“Specificity”	“Precision”	“F1-Score”
Count Vectorization	Case1	0.84871	0.49825	0.99748	0.866391	0.66893
	Case2	0.84286	0.49603	0.9915	0.86325	0.66125
	Case3	0.84048	0.47619	0.9966	0.856341	0.64445
	Case4	0.825	0.4881	0.96939	0.832518	0.64927
TF-IDF	Case5	0.7	0	1	0.49	0
	Case6	0.8489	0.46429	0.9983	0.86657	0.6338

Table (8) proved the best performance of the proposed system obtains with case 1 parameters and the count vectorization feature extraction algorithm, where an “accuracy” value of **0.84871**, a “sensitivity” of **0.49825**, a “specificity” of **0.99748**, a “precision” of **0.86639**, and an "F1-score" of **0.66893**. The TF-IDF algorithm and parameters from Case 6 gave the highest "accuracy" value of **0.8489**, a "sensitivity" of **0.46429**, a "specificity" of **0.9983**, a "precision" of **0.86657**, and an "F1-score" of **0.6338**.

Table (9) presents a comparison between the performance of the propose system based on count vectorization (Case 1) and TF-IDF (case 6) feature extraction algorithms and accuracy metrics.

Table 9 - Comparison Performance of DSQLIAS

Classification approach	algorithms	Metrics	Feature Extraction	
			Count Vectorization	TF-IDF Vectorization
	RNN-LSTM	Acc.	0.87976	0.90119
		Sen.	0.69048	0.6746
		Spec.	0.96088	0.9983
		Precis.	0.88006	0.91245
		F1-score	0.80354	0.80513
	RNN-GRU	Acc.	0.84871	0.8489
		Sen.	0.49825	0.46429
		Spec.	0.99748	0.9983
		Precis.	0.86639	0.86657
		F1-score	0.66893	0.6338

5. Conclusion

This paper has presented a novel system for detecting and classifying SQL injection attacks using RNN deep learning models. The system, named DSQLIAS, uses natural language processing techniques to extract key features from SQL queries that indicate the presence or absence of malicious patterns. The system employs two RNN algorithms, LSTM and GRU, to analyze the features and produce binary outputs. The system was evaluated on two datasets from Kaggle (Zhang et al., 2022; Pallam et al., 2021) and compared with existing rule-based methods. The results showed that DSQLIAS achieved high accuracy and outperformed the rule-based methods in both real-time and non-real-time scenarios. The results also showed that TF-IDF was a better feature extraction technique than count vectorization for SQL injection attack detection. The RNN-LSTM model achieved an accuracy of **0.90119**, while the RNN-GRU model achieved an accuracy of **0.8489**. These findings demonstrate the effectiveness and superiority of using deep learning models for SQL injection attack detection, and contribute to the field of web application security.

6. Future works

There are several opportunities for future research in RNN-based SQL injection detection (Alghawazi et al., 2022). For example, researchers may explore methods for improving the accuracy of existing models, integrating multiple detection techniques, and addressing the limitations of deep learning algorithms, such as overfitting and the need for large training datasets. Additionally, researchers may explore the use of RNNs for detecting other types of web application vulnerabilities. The designer can construct a more robust and accurate detection system by combining various machine and deep learning algorithms and comparing it to the individual methods to find the best SQL injection attack detection method.

References

Alarfaj, F. K., & Khan, N. A. (2023). Enhancing the Performance of SQL Injection Attack Detection through Probabilistic Neural Networks. *Applied Sciences*, *13*(7), 4365. https://doi.org/10.3390/app13074365

Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of SQL injection attack using machine learning techniques: a systematic literature review. *Journal of Cybersecurity and Privacy*, *2*(4), 764-777. https://doi.org/10.3390/jcp2040039

Alwan, Z. S., & Younis, M. F. (2017). Detection and prevention of SQL injection attack: a survey. *International Journal of Computer Science and Mobile Computing*, *6*(8), 5-17.

Aminanto, M. E., Purbomukti, I. R., Chandra, H., & Kim, K. (2022). Two-Dimensional Projection-Based Wireless Intrusion Classification Using Lightweight EfficientNet. *Computers, Materials & Continua*, *72*(3), 5301. https://doi.org/10.32604/cmc.2022.026749

Arock, M. (2021). Efficient detection of SQL injection attack (SQLIA) Using pattern-based neural network model. In *2021 International conference on computing, communication, and intelligent systems (ICCCIS)* (pp. 343-347). IEEE. https://doi.org/10.1109/ICCCIS51004.2021.9397066

ArunKumar, K. E., Kalaga, D. V., Kumar, C. M. S., Kawaji, M., & Brenza, T. M. (2021). Forecasting of COVID-19 using deep layer recurrent neural networks (RNNs) with gated recurrent units (GRUs) and long short-term memory (LSTM) cells. *Chaos, Solitons & Fractals*, *146*, 110861. https://doi.org/10.1016/j.chaos.2021.110861

Bhateja, N., Sikka, S., & Malhotra, A. (2021). A review of SQL injection attack and various detection approaches. *Smart and Sustainable Intelligent Systems*, 481-489, https://doi.org/10.1002/9781119752134.ch34

Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). SQL injection attack detection and prevention techniques using deep learning. In *Journal of Physics: Conference Series* (Vol. 1757, No. 1, p. 012055). IOP Publishing. https://doi.org/10.1088/1742-6596/1757/1/012055

Chen, Z., & Guo, M. (2018). Research on SQL injection detection technology based on SVM. In *MATEC web of conferences* (Vol. 173, p. 01004). EDP Sciences. https://doi.org/10.1051/matecconf/201817301004

Demilie, W. B., & Deriba, F. G. (2022). Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques. *Journal of Big Data*, *9*(1), 124. https://doi.org/10.1186/s40537-022-00678-0

Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022). A deep learning approach for detection of SQL injection attacks using convolutional neural networks. In *Proceedings of Data Analytics and Management: ICDAM 2021, Volume 2* (pp. 293-304). Springer Singapore. https://doi.org/10.1007/978-981-16-6285-0_24

Farhan, A. H., & Hasan, R. F. (2023). Detection SQL Injection Attacks Against Web Application by Using K-Nearest Neighbors with Principal Component Analysis. In *Proceedings of Data Analytics and Management: ICDAM 2022* (pp. 631-642). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-19-7615-5_52

Ghozali, I., Asy'ari, M. F., Triarjo, S., Ramadhani, H. M., Studiawan, H., & Shiddiqi, A. M. (2022). A Novel SQL Injection Detection Using Bi-LSTM and TF-IDF. In *2022 7th International Conference on Information and Network Technologies (ICINT)* (pp. 16-22). IEEE. https://doi.org/10.1109/ICINT55083.2022.00010

Hassan, M. M., Ahmad, R. B., & Ghosh, T. (2021). SQL injection vulnerability detection using deep learning: a feature-based approach. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, *9*(3), 702-718. http://dx.doi.org/10.52549/.v9i3.3131

Jemal, I., Cheikhrouhou, O., Hamam, H., & Mahfoudhi, A. (2020). Sql injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, *15*(6), 569-580.

Jothi, K. R., Pandey, N., Beriwal, P., & Amarajan, A. (2021, March). An efficient SQL injection detection system using deep learning. In *2021 International conference on computational intelligence and knowledge economy (ICCIKE)* (pp. 442-445). IEEE. https://doi.org/10.1109/ICCIKE51210.2021.9410674

Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2006). Secubat: a web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web* (pp. 247-256). https://doi.org/10.1145/1135777.1135817

Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., ... & Omar, N. (2021). SQL injection attacks prevention system technology. *Asian Journal of Research in Computer Science*, *10*(3), 13-32. https://doi.org/10.9734/AJRCOS/2021/v10i330242

Krishnan, S. A., Sabu, A. N., Sajan, P. P., & Sreedeep, A. L. (2021). SQL injection detection using machine learning. *REVISTA GEINTEC-GESTAO INOVACAO E TECNOLOGIAS, 11*(3), 300-310.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444. https://doi.org/10.1038/nature14539

Li, Q., Wang, F., Wang, J., & Li, W. (2019). LSTM-based SQL injection detection method for intelligent transportation system. *IEEE Transactions on Vehicular Technology*, *68*(5), 4182-4191. https://doi.org/ 0.1109/TVT.2019.2893675

Markoulidakis, I., Kopsiaftis, G., Rallis, I., & Georgoulas, I. (2021). Multi-class confusion matrix reduction method and its application on net promoter score classification problem. In *The 14th pervasive technologies related to assistive environments conference* (pp. 412-419). https://doi.org/10.1145/3453892.3461323

Nagasundari, S., & Honnavali, P. B. (2019). SQL injection attack detection using ResNet. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE. https://doi.org/10.1109/ICCCNT45670.2019.8944874

Oudah, M. A., Marhusin, M. F., & Narzullaev, A. (2022). SQL injection detection using machine learning with different TF-IDF feature extraction approaches. In *International Conference on Information Systems and Intelligent Applications* (pp. 707-720). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-16865-9_57

Pallam, R., Konda, S. P., Manthripragada, L., & Noone, R. A. (2021). Detection of Web Attacks using Ensemble Learning. *learning*, *3*(4), 5.

Roy, P., Kumar, R., & Rani, P. (2022). SQL injection attack detection by machine learning classifier. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* (pp. 394-400). IEEE. https://doi.org/10.1109/ICAAIC53929.2022.9792964

Sudharshan, K., Naveen, C., Vishnuram, P., Krishna Rao Kasagani, D. V. S., & Nastasi, B. (2022). Systematic review on impact of different irradiance forecasting techniques for solar energy prediction. *Energies*, *15*(17), 6267. https://doi.org/10.3390/en15176267

Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020). Detection of SQL injection based on artificial neural network. *Knowledge-Based Systems*, *190*, 105528. https://doi.org/10.1016/j.knosys.2020.105528

Theissler, A., Thomas, M., Burch, M., & Gerschner, F. (2022). ConfusionVis: Comparative evaluation and selection of multi-class classifiers based on confusion matrices. *Knowledge-Based Systems, 247,* 108651. https://doi.org/10.1016/j.knosys.2022.108651

Yu, L., Luo, S., & Pan, L. (2019, July). Detecting SQL injection attacks based on text analysis. In *3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019)* (pp. 95-101). Atlantis Press. https://doi.org/10.2991/iccia-19.2019.14

Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022). Deep Neural Network-Based SQL Injection Detection Method. *Security and Communication Networks*, 2022, 4836289. https://doi.org/10.1155/2022/4836289

Zhao, J., Wang, N., Ma, Q., & Cheng, Z. (2019). Classifying malicious URLs using gated recurrent neural networks. In *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2018)* (pp. 385-394). Springer International Publishing. https://doi.org/10.1007/978-3-319-93554-6_36.