Indonesian Journal of Data and Science

Volume 7 Issue 2
ISSN 2715-9930
https://doi.org/10.56705/ijodas.v7i2.460

Research Article

A Comparative Study of LSTM and CNN Models in SQL Injection Attack Detection

Abdul Rachman Manga' [1,*]; Wahyu Kadri Rahmat Suat[2]; Huzain Azis [3]
[1] Universitas Muslim Indonesia, Makassar, Indonesia, abdulrachman.manga@umi.ac.id
[2] Universitas Muslim Indonesia, Makassar, Indonesia, wahyukadrirahmatsuat.iclabs@umi.ac.id
[3] Universitas Muslim Indonesia, Makassar, Indonesia, huzain.azis@umi.ac.id

Correspondence should be addressed to Abdul Rachman Manga'; abdulrachman.manga@umi.ac.id

Received 10 May 2026; Accepted 30 July 2026; Published 31 July 2026

© Authors 2026. CC BY-NC 4.0 (non-commercial use with attribution, indicate changes).
License: https://creativecommons.org/licenses/by-nc/4.0/ — Published by Indonesian Journal of Data and Science.

Abstract:
Introduction: SQL Injection (SQLi) remains a critical cybersecurity threat because it exploits vulnerabilities in user input validation and can compromise the confidentiality, integrity, and availability of information systems. This study compares Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) architectures for detecting malicious SQL queries under identical experimental conditions. **Method**: A publicly available dataset containing 148,327 malicious and benign SQL query instances was preprocessed through missing-value removal, label encoding, tokenization, sequence transformation, padding, and embedding representation. LSTM and CNN models were evaluated using three train-test split scenarios of 70:30, 80:20, and 90:10. Performance was assessed using accuracy, precision, recall, F1-score, and confusion matrices, with the 80:20 split selected for detailed evaluation. **Results and Discussion**: LSTM consistently achieved higher accuracy across the evaluated splits, ranging from 97.84% to 98.00%. Under the 80:20 configuration, LSTM achieved 97.86% accuracy, 99.34% precision, 96.55% recall, and a 97.92% F1-score, compared with CNN at 97.00%, 97.68%, 96.56%, and 97.12%, respectively. LSTM also reduced false positives from 356 to 99, demonstrating better discrimination between legitimate and malicious queries. **Conclusion**: LSTM provides more reliable SQL Injection detection than CNN by better capturing sequential dependencies within SQL query structures, making it a promising approach for practical cybersecurity systems.

Keywords: SQL Injection; Deep Learning; Long Short-Term Memory; Convolutional Neural Network; Embedding.

Dataset link: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

1. Introduction

The rapid digital transformation of web-based applications has significantly improved the accessibility and efficiency of information systems across various sectors, including education, healthcare, finance, e-commerce, and government services. However, this rapid growth has also increased the number and sophistication of cyberattacks targeting web applications [1]. Among these threats, Structured Query Language Injection (SQL Injection or SQLi) remains one of the most critical security vulnerabilities because it exploits weaknesses in user input validation to manipulate database queries. Successful SQL Injection attacks may allow unauthorized users to access, modify, or delete sensitive information, resulting in severe consequences for data confidentiality, integrity, and availability.

Despite continuous advancements in web security technologies, SQL Injection attacks remain difficult to eliminate due to the constantly evolving attack techniques and the diversity of application architectures. Conventional detection approaches, including signature-based and rule-based techniques, generally perform well against previously known attack patterns but often fail to recognize newly emerging or obfuscated SQL Injection payloads. Consequently, there is an increasing demand for intelligent detection mechanisms capable of automatically learning complex patterns from SQL queries and adapting to evolving attack behaviors [2], [3].

Recent advances in deep learning have demonstrated promising capabilities for cybersecurity applications, particularly in intrusion detection and malicious query classification. Various architectures have been proposed to improve SQL Injection detection performance. Kakisim proposed a multi-view consensus deep learning framework that effectively integrates multiple feature representations to improve detection accuracy [4]. Liu introduced a hybrid BERT–LSTM architecture that combines contextual language representation with sequential learning to detect SQL Injection attacks more effectively [5]. Thalji *et al.* developed AE-Net, an autoencoder-based framework that extracts deep features for SQL Injection detection and improves classification performance [6]. Lu *et al.* proposed a semantic learning-based detection approach that captures contextual information from SQL queries to distinguish malicious and benign inputs more accurately [7]. In addition, Arasteh *et al.* employed Binary Gray Wolf Optimizer with machine learning algorithms to enhance feature selection and classification performance [8], while Bakır introduced UniEmbed, a feature fusion framework that combines multiple embedding representations for detecting both SQL Injection and Cross-Site Scripting attacks [9].

Although these studies have reported encouraging results, several limitations remain. Many existing approaches rely on hybrid architectures, multiple feature extraction techniques, or sophisticated optimization algorithms that increase computational complexity and reduce implementation efficiency in practical environments. Furthermore, most previous studies primarily focus on improving a single proposed architecture, while only limited attention has been given to systematically comparing widely adopted standalone deep learning models under identical experimental settings. Consequently, it remains difficult to objectively evaluate the respective strengths and limitations of sequence-based and convolution-based learning models for SQL Injection detection.

Motivated by these limitations, this study presents a comparative evaluation of two widely used deep learning architectures, namely Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), for SQL Injection attack detection. Both models are trained using the same publicly available SQL Injection dataset, identical preprocessing procedures, embedding representation, and three train-test split scenarios (70:30, 80:20, and 90:10). Their performance is evaluated using accuracy and confusion matrix analysis to ensure a fair and comprehensive comparison.

The main contributions of this study are summarized as follows. First, this work provides a systematic comparison between standalone LSTM and CNN architectures using identical preprocessing and embedding strategies, enabling a fair evaluation of both models. Second, the study investigates the effect of different train-test split scenarios on model performance and generalization capability. Third, the experimental results provide practical insights into selecting appropriate deep learning architectures for SQL Injection detection, particularly for developing efficient and reliable cybersecurity systems.

2. Method

Research Framework

This study employed a comparative experimental approach to evaluate the performance of Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models for SQL Injection (SQLi) attack detection. The objective of this study was to compare the capability of sequence-based and convolution-based deep learning architectures under identical experimental conditions. To ensure a fair comparison, both models were implemented using the same dataset, preprocessing procedures, and evaluation metrics.

The research workflow began by collecting a publicly available SQL Injection dataset, followed by data preprocessing, including missing value removal, label encoding, text tokenization, sequence transformation, and sequence padding. The processed data were then divided into training and testing subsets using three train-test split scenarios (70:30, 80:20, and 90:10). For each scenario, the same preprocessing pipeline and hyperparameter settings were maintained, while only the proportion of training and testing data was varied.

The preprocessed data were subsequently used to train the LSTM and CNN models independently. Each model learned feature representations through an embedding layer before performing binary classification to distinguish malicious SQL Injection queries from legitimate queries. Finally, the trained models were evaluated using several

performance metrics, including Accuracy, Precision, Recall, F1-score, and Confusion Matrix [10]. The overall research framework is illustrated in **Figure 1**.

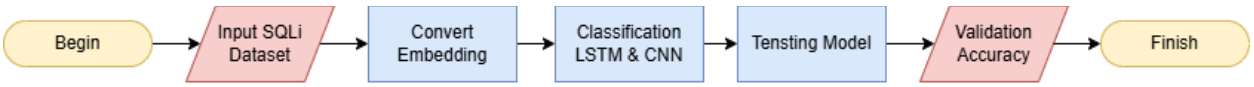


Figure 1. Research Flow

A. Dataset

The experiments were conducted using a publicly available SQL Injection dataset obtained from the Kaggle repository. This dataset was selected because it contains a large collection of SQL queries representing both malicious and benign requests and has been widely adopted as a benchmark for SQL Injection detection research [11]. The dataset consists of 148,327 query instances organized into two attributes: Query, which contains the SQL statement, and Label, which represents the corresponding class. A label value of 1 indicates a malicious SQL Injection query, whereas 0 denotes a legitimate query.

Prior to model development, the dataset was examined to identify incomplete records. Missing values were removed to improve data quality and ensure consistency during preprocessing. After cleaning, the remaining dataset was used throughout the experimental process for both model training and evaluation. **Table 1** presents a sample of the dataset used in this study.

Table 1. Dataset Sample

Query SQL	Definition	Data Type
select * from users where id = '1' or @ @1 = 1 union select 1,version ( ) -- 1'	SQL query input	Object
? or 1 = 1 --	SQL query input	Object
select * from users where id = '1' * (\ ) or 1 = 1 -- 1'	SQL query input	Object
select * from users where id = 1 or "%{" or 1 = 1 -- 1	SQL query input	Object
admin" or "1" = "1"--	SQL query input	Object

Data Preprocessing

Data preprocessing was performed to transform raw SQL queries into numerical representations suitable for deep learning models. This stage consisted of five sequential processes: missing value removal, label encoding, text tokenization, sequence transformation, and sequence padding.

First, records containing missing values were removed from the dataset to prevent incomplete information from affecting model performance. Next, the binary class labels were transformed into numerical values using the LabelEncoder provided by the Scikit-learn library [12]. This process encoded malicious queries as class 1 and legitimate queries as class 0, allowing the models to perform binary classification efficiently.

The SQL queries were then converted into numerical tokens using the Keras Tokenizer [13]. During this process, an Out-of-Vocabulary (OOV) token was introduced to represent previously unseen words encountered during inference, thereby improving the robustness of the trained models. The tokenizer generated integer indices corresponding to each token appearing in the SQL queries.

After tokenization, every query was transformed into a sequence of integers and standardized using the pad_sequences function with a maximum sequence length of 100 tokens [14]. Padding ensured that all input sequences had the same dimensionality before being processed by the deep learning models. This transformation enabled both LSTM and CNN architectures to receive consistent input representations while preserving the semantic structure of SQL queries [15].

Deep Learning Models

This study compared two deep learning architectures, namely Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), to classify SQL queries into malicious and benign categories. Both architectures employed an embedding layer to automatically learn dense vector representations of SQL query tokens before feature extraction and classification [16].

1) Long Short-Term Memory (LSTM)

The LSTM model was designed to capture sequential dependencies among SQL query tokens [17], [28]. The architecture consisted of an Embedding layer with an embedding dimension of 128, followed by a SpatialDropout1D layer with a dropout rate of 0.2 to reduce overfitting. Feature extraction was then performed using an LSTM layer containing 64 hidden units with dropout and recurrent dropout values of 0.2. Finally, a Dense output layer with a sigmoid activation function produced the binary classification result. The architecture of the proposed LSTM model is illustrated in **Figure 2**.

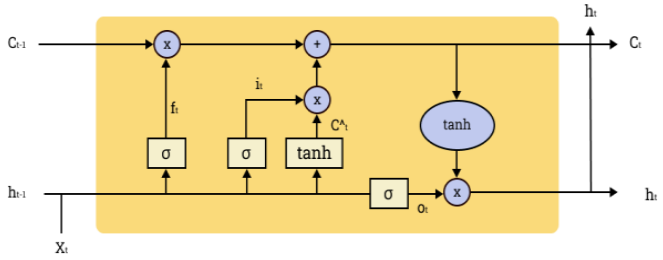


Figure 2. LSTM Architecture

2) Convolutional Neural Network (CNN)

The CNN architecture was designed to identify local textual patterns within SQL queries through one-dimensional convolution. The model began with an Embedding layer having an embedding dimension of 128, followed by a Conv1D layer consisting of 128 filters with a kernel size of 5 and ReLU activation. A Global Max Pooling layer was employed to extract the most informative features from the convolution outputs. Two Dropout layers with a rate of 0.5 were applied to reduce overfitting, while a Dense hidden layer containing 64 neurons performed high-level feature learning. Finally, a Dense layer with sigmoid activation generated the binary classification output [27]. The CNN architecture is presented in **Figure 3**.

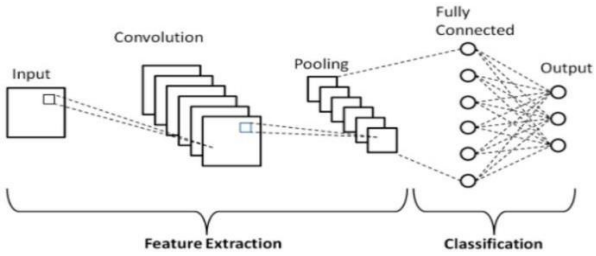


Figure 3. CNN Architecture

Experimental Setup

All experiments were conducted using the TensorFlow deep learning framework with identical hyperparameter settings to ensure a fair comparison between the two models [18]. The dataset was evaluated using three train-test split scenarios, namely 70:30, 80:20, and 90:10. Except for the proportion of training and testing data, all preprocessing procedures, model architectures, and training configurations remained unchanged throughout the experiments.

Both models were trained for 100 epochs using a batch size of 1024. The Adam optimizer was employed to minimize the Binary Cross-Entropy loss function [19]. During the training process, the ModelCheckpoint callback was utilized to automatically save the model that achieved the best validation performance, thereby preventing the

final evaluation from using a suboptimal model. The complete hyperparameter configuration used in this study is summarized in **Table 2**.

Table 2. Hyperparameter Configuration of LSTM and CNN Models

Algorithm	Layer/Kernel Function	Parameter	Value
LSTM	Embedding	Input_dim	5000
		Output_dim	128
		Input_shape	100
	SpatialDropout1D	Rate	0.2
	LSTM	Units	64
		Dropout	0.2
		Recurrent_dropout	0.2
		Return_sequences	False
	Dense (Output)	Units	1
		Activation	Sigmoid
	Compile	Optimizer	Adam
		Loss	Binary Crossentropy
		Metrics	Accuracy
	Training Config	Epochs	100
		Batch_size	1024
		Validation_split	0.2
		Callbacks	ModelCheckpoint (save best model by val_loss)
CNN	Embedding	Input_dim	len(word_index) + 1
		Output_dim	128
	Conv1D	Filters	128
		Kernel_size	5
		Activation	ReLU
	Dense (Hidden)	Units	64
		Activation	ReLU
	Dense (Output)	Units	1
		Activation	Sigmoid
	Compile	Optimizer	Adam
		Loss	Binary Crossentropy
		Metrics	Accuracy
	Training Config	Epochs	100
		Batch_size	1024

Performance Evaluation

The performance of the proposed models was evaluated using several classification metrics derived from the confusion matrix, including Accuracy, Precision, Recall, and F1-score. Accuracy measures the proportion of correctly classified samples among all observations, whereas Precision evaluates the reliability of positive predictions [20], [21]. Recall measures the capability of the model to correctly identify malicious SQL Injection queries, while the F1-score provides a balanced assessment of Precision and Recall.

The evaluation metrics used in this study are mathematically defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

where TP, TN, FP, and FN denote True Positive, True Negative, False Positive, and False Negative, respectively.

$$Precesion = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 - Score = 2\ x\ \frac{Precision\ x\ Recall}{Precision\ x\ Recall} \quad (4)$$

In addition to numerical evaluation, confusion matrices were generated to visualize the distribution of correct and incorrect classifications for each model. Training and validation accuracy curves, together with loss curves, were also analyzed to observe the convergence behavior and generalization capability of both deep learning architectures during the learning process. These evaluation metrics provide a comprehensive assessment of model performance and facilitate an objective comparison between the LSTM and CNN models.

3. Result and Discussion

Experimental Results

This section presents the experimental results obtained from the proposed Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models for SQL Injection attack detection. To evaluate the robustness and generalization capability of both architectures, three train-test split scenarios were implemented, namely 70:30, 80:20, and 90:10. Each experiment employed identical preprocessing procedures, including missing value removal, label encoding, text tokenization, sequence transformation, sequence padding, and the same training parameters. Therefore, the performance differences observed in this study are attributed solely to the characteristics of the deep learning architectures rather than variations in the experimental configuration.

The classification accuracy obtained by both models under the three experimental scenarios is presented in **Table 3**.

Table 3. Classification Accuracy under Different Train-Test Split Ratios.

Model	70:30	80:20	90:10
LSTM	97.84%	97.86%	98.00%
CNN	97.15%	97.00%	96.84%

As shown in **Table 3**, the LSTM model consistently outperformed the CNN model across all train-test split scenarios. The highest classification accuracy of 98.00% was obtained using the 90:10 train-test split, while the CNN model achieved its highest accuracy of 97.15% under the 70:30 split. Although the 90:10 split produced the best overall accuracy for the LSTM model, the 80:20 train-test split was selected for detailed performance evaluation because it provides a more balanced distribution between training and testing data. This split is widely adopted in machine learning studies since it offers sufficient training samples while maintaining a relatively large testing dataset for reliable model evaluation.

The experimental results also indicate that the LSTM architecture demonstrates more stable performance across different data distributions than the CNN model. The variation in LSTM accuracy remains relatively small, ranging from 97.84% to 98.00%, whereas the CNN model exhibits greater performance fluctuations across the three experimental scenarios. These findings suggest that the sequential learning mechanism employed by LSTM is more effective in capturing contextual relationships within SQL queries, enabling the model to generalize better under different training and testing conditions.

Overall, the results demonstrate that increasing the proportion of training data slightly improves the classification accuracy of the LSTM model. However, the improvement is relatively modest, indicating that the proposed model

maintains robust performance regardless of the selected train-test split ratio. Consequently, the LSTM architecture is considered more suitable for SQL Injection attack detection than the CNN architecture under the experimental settings adopted in this study.

Performance Evaluation

To provide a comprehensive assessment of the proposed models, additional evaluation metrics were calculated using the confusion matrix generated from the 80:20 train-test split, which was selected as the primary experimental configuration. Besides overall classification accuracy, Precision, Recall, and F1-score were employed to evaluate the capability of each model in distinguishing malicious SQL Injection queries from legitimate SQL queries. These metrics provide a more detailed understanding of the classification performance, particularly in cybersecurity applications where minimizing classification errors is critical.

Table 4. Performance Comparison of LSTM and CNN Models

Metrix	LSTM	CNN
Accuracy (%)	97.86	97.00
Precision (%)	99.34	97.68
Recall (%)	96.55	96.56
F1-score (%)	97.92	97.12

As presented in **Table 4**, both models achieved high classification performance in detecting SQL Injection attacks. Nevertheless, the LSTM model consistently produced superior results in terms of Accuracy, Precision, and F1-score. The LSTM model achieved an overall accuracy of 97.86%, which is 0.86 percentage points higher than the CNN model. Although the improvement appears relatively small, it demonstrates the effectiveness of sequential learning in extracting contextual information from SQL queries.

The most notable improvement is observed in the Precision metric, where the LSTM model achieved 99.34%, compared to 97.68% for the CNN model. This indicates that the LSTM architecture is more effective in minimizing false positive predictions, thereby reducing the likelihood of classifying legitimate SQL queries as malicious attacks. Such performance is particularly important in practical intrusion detection systems because excessive false alarms may increase the workload of security administrators and reduce confidence in automated security mechanisms.

Meanwhile, both models achieved comparable Recall values, indicating that they exhibit similar capabilities in identifying malicious SQL Injection attacks. However, the higher F1-score obtained by the LSTM model demonstrates a better balance between Precision and Recall, suggesting that the proposed architecture provides a more reliable overall classification performance.

The evaluation results confirm that incorporating sequential contextual information through the LSTM architecture significantly enhances SQL Injection detection performance while maintaining a balanced trade-off between attack detection capability and classification reliability.

Confusion Matrix Analysis

To further investigate the classification performance of the proposed models, confusion matrix analysis was performed using the 80:20 train-test split, which produced a balanced evaluation between training and testing data. The confusion matrix provides detailed information regarding the number of correctly and incorrectly classified instances, including True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). These values offer a comprehensive understanding of the classification behavior of each model beyond the overall accuracy.

The confusion matrices obtained from the LSTM and CNN models are presented in Figure 4 and Figure 5, respectively.

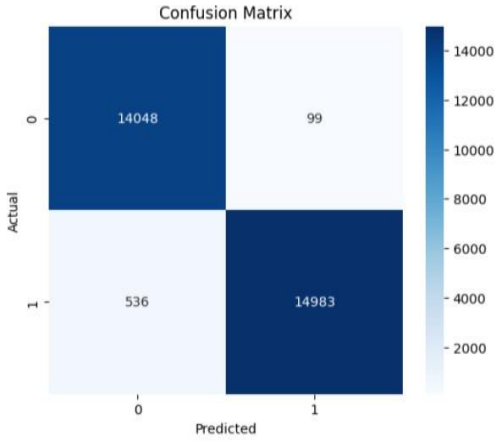


Figure 4. Confusion Matrix of the LSTM Model (80:20 Train-Test Split).

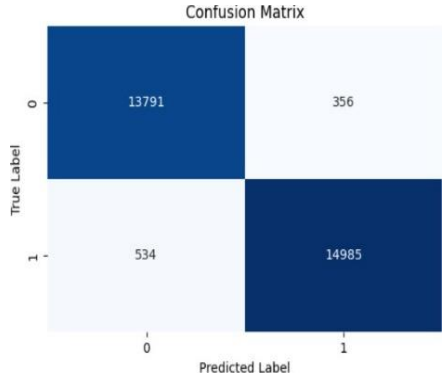


Figure 5. Confusion Matrix of the CNN Model (80:20 Train-Test Split).

Based on **Figure 4**, the LSTM model correctly classified 14,048 legitimate SQL queries and 14,983 malicious SQL Injection queries. Meanwhile, the model generated 99 false positive predictions and 536 false negative predictions. These results indicate that the LSTM architecture effectively distinguishes between normal and malicious SQL queries while maintaining a low false alarm rate.

In contrast, the CNN model correctly classified 13,791 legitimate queries and 14,985 malicious queries, while producing 356 false positive predictions and 534 false negative predictions, as illustrated in Figure 5. Although the CNN model achieved comparable true positive predictions, it generated a considerably higher number of false positive classifications than the LSTM model.

A comparison between both confusion matrices reveals that the most significant difference lies in the false positive predictions. The LSTM model reduced the number of false positive cases from 356 to 99, corresponding to an approximate 72.2% reduction compared with the CNN model. This improvement demonstrates the ability of the LSTM architecture to preserve contextual relationships within SQL queries, enabling the model to better discriminate between legitimate SQL statements and SQL Injection attacks.

From the perspective of practical cybersecurity applications, reducing false positive predictions is highly desirable because unnecessary security alerts may increase the workload of security administrators and decrease confidence in automated intrusion detection systems. Therefore, despite both models exhibiting comparable recall values, the substantially lower false positive rate achieved by the LSTM model represents an important practical advantage for real-world SQL Injection detection systems.

Furthermore, the relatively small number of false negative predictions observed in both models indicates that most malicious SQL Injection attacks were successfully identified. This finding confirms that both deep learning architectures are capable of learning discriminative representations of SQL queries. However, the superior balance between true positive detection and false alarm reduction achieved by the LSTM model makes it more suitable for deployment in practical web application security environments.

Learning Curve Analysis

The To evaluate the learning behavior of both deep learning models during training, the accuracy and loss values were monitored throughout 100 training epochs. The corresponding learning curves are presented in **Figure 6** to **Figure 9**. These curves provide insight into the convergence process, model stability, and generalization capability during the optimization phase.

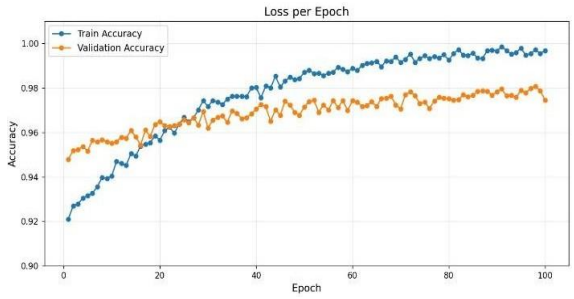


Figure 6. Training and Validation Accuracy of the LSTM Model**.**

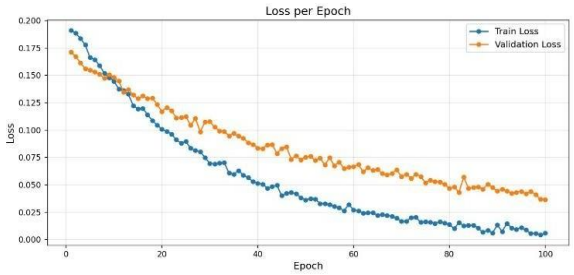


Figure 7. Training and Validation Loss of the LSTM Model**.**

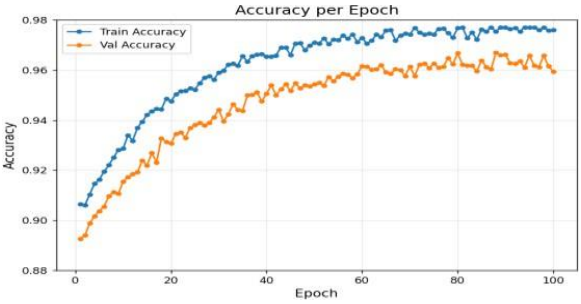


Figure 8. Training and Validation Accuracy of the CNN Model**.**

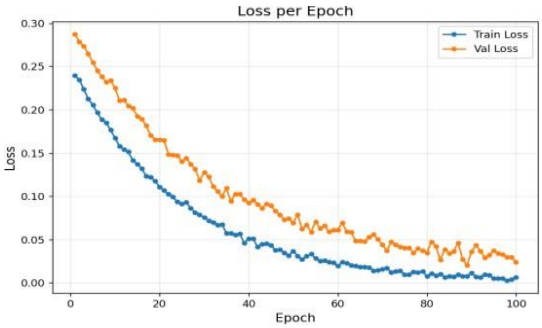


Figure 9. Training and Validation Loss of the CNN Model**.**

Figure 6 and **7** illustrate the learning process of the LSTM model during training. The training and validation accuracy increased steadily throughout the training process before converging near the final epochs. Simultaneously, both training and validation loss gradually decreased and stabilized without exhibiting significant divergence. These observations indicate that the LSTM model successfully learned meaningful sequential representations from SQL queries while maintaining good generalization performance.

The absence of substantial fluctuations between the training and validation curves suggests that the selected hyperparameters, including the embedding dimension, batch size, dropout rate, and learning rate, were appropriate for the SQL Injection classification task. Although minor differences between the training and validation accuracy appeared in the later epochs, these variations remained relatively small and did not indicate severe overfitting.

Similarly, **Figure 8** and **9** present the learning curves of the CNN model. The model exhibited continuous improvements in classification accuracy during the training process, while the loss values gradually decreased as the number of epochs increased. However, compared with the LSTM model, the CNN accuracy curve displayed slightly larger fluctuations during intermediate training epochs, suggesting that the convolutional architecture required additional iterations to stabilize feature extraction from SQL query sequences.

Despite these fluctuations, the CNN model converged successfully and achieved competitive classification performance. Nevertheless, its final validation accuracy remained lower than that of the LSTM model, which is consistent with the evaluation metrics presented in Table IV. This observation indicates that although convolutional feature extraction is capable of identifying important local patterns within SQL queries, it is less effective than recurrent sequence modeling in capturing long-range contextual dependencies that frequently characterize SQL Injection attacks.

Overall, the learning curves demonstrate that both deep learning architectures converged successfully within the 100 training epochs. However, the smoother convergence and higher validation accuracy achieved by the LSTM model indicate superior learning stability and better generalization capability. These findings further support the selection of the LSTM architecture as the more effective approach for SQL Injection attack detection under the experimental conditions adopted in this study.

Comparison with Previous Studies

To further validate the effectiveness of the proposed approach, the experimental results obtained in this study were compared with several recent studies on SQL Injection attack detection. Recent research has demonstrated that deep learning techniques significantly improve SQL Injection detection by learning complex representations from SQL query sequences. Various approaches have been explored, including semantic learning, hybrid transformer-recurrent networks, autoencoder-based feature extraction, multi-view consensus learning, optimization-assisted machine learning, and multi-feature embedding fusion. These methods have shown promising performance by exploiting different feature extraction and learning strategies for distinguishing malicious SQL queries from legitimate ones [18], [19].

Unlike previous studies that primarily focused on developing more sophisticated hybrid architectures, this study evaluates two widely adopted deep learning models, namely Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), under identical preprocessing procedures and experimental settings. By maintaining the same dataset, preprocessing pipeline, training parameters, and evaluation metrics, this study provides a fair comparison between sequential learning and convolution-based feature extraction without introducing additional variables that may influence classification performance [22], [23].

The experimental results demonstrate that the LSTM model consistently achieved superior performance across all train-test split scenarios. The highest classification accuracy of 98.00% was obtained using the 90:10 train-test split. However, the 80:20 split was selected for detailed performance evaluation because it provides a more balanced distribution between training and testing data while maintaining excellent classification performance. Furthermore, the LSTM model significantly reduced false positive predictions compared with the CNN model, demonstrating its ability to distinguish legitimate SQL queries from SQL Injection attacks more effectively.

Another important contribution of this study lies in its experimental design. Rather than introducing additional feature engineering techniques or complex hybrid architectures, the proposed approach employs standard text preprocessing combined with conventional deep learning models. This experimental setting highlights the intrinsic capability of the LSTM architecture to capture contextual dependencies within SQL query sequences while maintaining a relatively simple model structure.

Overall, the findings indicate that a conventional LSTM architecture, when combined with appropriate preprocessing techniques, is capable of achieving competitive SQL Injection detection performance without relying on transformer-based language models or computationally expensive hybrid frameworks [24]. Consequently, the proposed approach offers a practical balance between classification performance, computational efficiency, and implementation simplicity, making it suitable for real-world web application security systems [25], [26].

Discussion

The experimental results demonstrate that both LSTM and CNN are capable of detecting SQL Injection attacks with high classification performance. Nevertheless, the LSTM model consistently outperformed the CNN model

across all experimental scenarios, indicating that sequential modeling provides a more effective representation of SQL query structures than convolution-based feature extraction.

One possible explanation for this performance difference is the inherent characteristic of SQL Injection attacks. Malicious SQL queries are composed of ordered SQL keywords, operators, and clauses whose meanings depend heavily on their sequential arrangement. The LSTM architecture is specifically designed to preserve long-term contextual dependencies within sequential data, enabling the model to capture semantic relationships between SQL tokens more effectively than CNN, which primarily focuses on extracting local features through convolution operations.

This characteristic is reflected not only in the overall classification accuracy but also in the confusion matrix analysis. The LSTM model generated substantially fewer false positive predictions than the CNN model, indicating that legitimate SQL queries were less likely to be incorrectly classified as malicious attacks. From an operational perspective, reducing false positive predictions is highly desirable because excessive security alerts may increase the workload of security analysts and reduce trust in automated intrusion detection systems.

Another important observation is that the classification performance of the LSTM model remained relatively stable across the three train-test split scenarios. Although the highest accuracy was achieved using the 90:10 split, the improvement over the 80:20 configuration was relatively small. Therefore, the 80:20 train-test split was selected for detailed performance evaluation because it provides a more balanced distribution between training and testing data while maintaining reliable classification performance. This experimental configuration is also widely adopted in machine learning research, allowing more representative model evaluation compared with smaller testing datasets.

Compared with previous studies, the proposed approach demonstrates that high SQL Injection detection performance can still be achieved without relying on highly complex hybrid architectures or transformer-based language models. Instead, a conventional LSTM architecture combined with standard word embedding and appropriate preprocessing techniques is capable of producing competitive classification performance while maintaining lower implementation complexity. This finding is particularly valuable for practical cybersecurity applications where computational efficiency and ease of deployment remain important considerations.

Despite the encouraging results, this study has several limitations. The experiments were conducted using a single publicly available SQL Injection dataset, which may not fully represent the diversity of SQL Injection patterns encountered in real-world web applications. Furthermore, only standalone LSTM and CNN architectures were investigated. Additional experiments involving hybrid deep learning models, transformer-based architectures such as BERT, or attention mechanisms may further improve detection performance.

Future research should also evaluate the proposed models using larger and more diverse datasets collected from real-world web applications. In addition, further investigation into computational efficiency, inference time, memory consumption, and real-time deployment performance would provide valuable insights into the practical applicability of deep learning-based SQL Injection detection systems.

4. Conclusion

This study presented a comparative evaluation of Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models for SQL Injection attack detection using identical datasets, preprocessing procedures, and experimental settings. Three train-test split scenarios (70:30, 80:20, and 90:10) were employed to investigate the influence of training data proportions on model performance while ensuring a fair comparison between both deep learning architectures.

The experimental results demonstrate that both LSTM and CNN achieved high classification performance in detecting malicious SQL queries. However, the LSTM model consistently outperformed the CNN model across all train-test split scenarios. The best performance was obtained using the 80:20 split configuration, where LSTM achieved an accuracy of 97.86%, while CNN reached 97.00%. In addition, the confusion matrix analysis and other evaluation metrics indicate that LSTM provided more reliable classification performance with fewer misclassified SQL Injection queries than CNN.

Overall, the findings indicate that sequence-based learning offered by the LSTM architecture is more effective in capturing the contextual relationships among SQL query tokens than the convolution-based feature extraction employed by CNN. Therefore, LSTM can be considered a more suitable deep learning model for SQL Injection detection in binary classification tasks.

Future work may investigate more advanced deep learning architectures, such as Transformer-based models or hybrid deep learning approaches, as well as evaluate the proposed models on larger and more diverse SQL Injection datasets to further improve detection performance and generalization capability.

ACKNOWLEDGMENT

The Integrated Computer Labs (ICLabs) at Universitas Muslim Indonesia's Faculty of Computer Science provided support for this research and was instrumental in supplying the necessary equipment. The lab provides resources for data testing and analysis in addition to safe and reliable data storage solutions. This support was essential for guaranteeing efficient data processing and a seamless study procedure. Thanks to ICLabs' sufficient resources, the research was carried out more effectively, producing trustworthy results.

References

[1] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh, and R. Al-Qassas, "A Systematic Review of Detection and Prevention Techniques of SQL Injection Attacks," *Information Security Journal: A Global Perspective*, vol. 32, no. 4, pp. 252–265, 2023.

[2] Y. Chen, G. Liang, and Q. Wang, "Research on SQL Injection Detection Technology Based on Content Matching and Deep Learning," *Comput. Mater. Contin.*, vol. 84, no. 1, pp. 1145–1167, 2025, doi: https://doi.org/10.32604/cmc.2025.063319.

[3] R. T. Lo, W. J. Hwang, and T. M. Tai, "SQL Injection Detection Based on Lightweight Multi-Head Self-Attention," Appl. Sci., vol. 15, no. 2,pp. 1–17, 2025, doi: https://doi.org/10.3390/app15020571.

[4] A. G. Kakisim, "A Deep Learning Approach Based on Multi-View Consensus for SQL Injection Detection," *International Journal of Information Security*, vol. 23, no. 2, pp. 1541–1556, 2024, doi: https://doi.org/10.1007/s10207-023-00791-y.

[5] Y. Liu, "Deep Learning in Cybersecurity: A Hybrid BERT–LSTM Network for SQL Injection Attack Detection," *IET Information Security*, 2024, doi: https://doi.org/10.1049/2024/5565950.

[6] N. Thalji, A. Raza, M. S. Islam, N. Abdel Samee, and M. M. Jamjoom, "AE-Net: Novel Autoencoder-Based Deep Features for SQL Injection Attack Detection," *IEEE Access*, vol. 11, pp. 135507–135516, 2023.

[7] D. Lu, J. Fei, and L. Liu, "A Semantic Learning-Based SQL Injection Attack Detection Technology," *Electronics*, vol. 12, no. 6, p. 1344, 2023.

[8] B. Arasteh *et al*., "Detecting SQL Injection Attacks by Binary Gray Wolf Optimizer and Machine Learning Algorithms," *Neural Computing and Applications*, vol. 36, pp. 6771–6792, 2024.

[9] R. Bakır, "UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques," *Arabian Journal for Science and Engineering*, 2025.

[10] J.Triloka, H. Hartono, and S. Sutedi, "Detection of SQL Injection Attack Using Machine Learning Based On Natural Language Processing," *Int. J. Artif. Intell. Res.*, vol. 6, no. 2, 2022, doi: https://doi.org/10.29099/ijair.v6i2.355.

[11] S R. Menaka, G. Dharani, P. Kalaivani, S. R. Basha, S. K. S. Hareeth, and V. Kalaiyarasan, "An Efficient SQL Injection Detection with a Hybrid CNN & Random Forest Approach," *J. Inf. Syst. Eng. Manag.*, vol. 10, pp. 664–673, 2025, doi: https://doi.org/10.52783/jisem.v10i18s.2979.

[12] A. Alazzawi, "Sql Injection Detection Using Rnn Deep Learning Model," *Sql Inject. Detect. Using Rnn Deep Learn. Model*, vol. 5, no. 1, pp. 531–541, 2023, doi: https://doi.org/10.37385/jaets.v5i1.2864.

[13] C. S. Datasets, "Enhancing Cyber Security : A Study of Data Preprocessing Techniques for Enhancing Cyber Security : A Study of Data Preprocessing Techniques for Cyber Security Datasets," no. September, 2024, doi: https://doi.org/10.32628/IJSRST2411427.

[14] T. Sabri, S. Bahassine, O. El Beggar, and M. Kissi, “An improved Arabic text classification method using word embedding,” Int. J. Electr. Comput. Eng., vol. 14, no. 1, pp. 721–731, 2024, doi: https://doi.org/10.11591/ijece.v14i1.pp721.

[15] H. Darwis, Z. Ali, Purnawansyah, H. Lahuddin, and H. Azis, “Deep Dive Into Pubmed Rct: Leveraging Tribrid Embedding Recurrent Neural Network Model,” *ICIC Express Lett.*, vol. 19, no. 1, pp. 111–118, 2025, doi: https://doi.org/10.24507/icicel.19.01.111.

[16] M. Alazab, S. Srinivasan, S. Venkatraman, V. Quoc Pham, V. Ravi, and Q.-V. Pham, “Deep learning for cyber security applications: A comprehensive survey,” *Techrxiv.Org*, no. October, pp. 0–34, 2023, doi: https://doi.org/10.36227/techrxiv.16748161.

[17] H. C. Altunay and Z. Albayrak, “A hybrid CNN + LSTMbased intrusion detection system for industrial IoT networks,” *Eng. Sci. Technol. an Int. J.*, vol. 38, p. 101322, 2023, doi: https://doi.org/10.1016/j.jestch.2022.101322.

[18] M. Sajid *et al.*, “Enhancing intrusion detection: a hybrid machine and deep learning approach,” *J. Cloud Comput.*, vol. 13, no. 1, 2024, doi: https://doi.org/10.1186/s13677-024-00685-x.

[19] H. Sun, Y. Du, and Q. Li, “Deep Learning-Based Detection Technology for SQL Injection Research and Implementation,” *Appl. Sci.*, vol. 13, no. 16, 2023, doi: https://doi.org/10.3390/app13169466.

[20] R. Amanda and J. Aulia, “Hybrid CNN-LSTM and Cox Model for Bipolar Risk Analysis Using Social Media Data,” *Indonesian Journal of Data and Science*, vol. 6, no. 2, pp. 222–231, 2025, doi: https://doi.org/10.56705/ijodas.v6i2.265.

[21] M. I. Abidin, I. Nurtanio, and A. Achmad, “Deepfake Detection in Videos Using Long Short-Term Memory and CNN ResNext,” *ILKOM Jurnal Ilmiah*, vol. 14, no. 3, pp. 178–185, Dec. 2022, doi: https://doi.org/10.33096/ilkom.v14i3.1254.178-185.

[22] R. Satra, I. A. Dahlan, H. Darwis, Purnawansyah, S. Mujaddid, and F. Fattah, “A Comparison of Accuracy: KNN, TabNet, and Wide & Deep Learning for DDoS Attack Detection in Software Defined Network,” *Proc. 2025 19th Int. Conf. Ubiquitous Inf. Manag. Commun. IMCOM 2025*, 2025, doi: https://doi.org/10.1109/IMCOM64595.2025.10857511.

[23] B. Gegeleso and O. Ebiesuwa, "Comparative Analysis of Random Forest and LSTM Models for Customer Churn Prediction Based on Customer Satisfaction and Retention," Indones. J. Data Sci., vol. 6, no. 2, pp. 313–323, 2025, doi: https://doi.org/10.56705/ijodas.v6i2.244.

[24] P. Romadloni, B. Adhi Kusuma, and W. Maulana Baihaqi, “Komparasi Metode Pembelajaran Mesin Untuk Implementasi Pengambilan Keputusan Dalam Menentukan Promosi Jabatan Karyawan,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 6, no. 2, pp. 622–628, 2022, doi: https://doi.org/10.36040/jati.v6i2.5238.

[25] Z. Jin, B. Yu, and T. Zhou, “The Mechanism of Employee Characteristics on Promotion: Building the Predictive Machine Learning Model,” Applied and Computational Engineering, vol. 134, pp. 123–137, 2025, doi: https://doi.org/10.54254/2755-2721/2025.22254.

[26] A. Rattrout, M. Jaradat, and R. Jayousi, "Machine Learning Advancements in SQL Injection Detection: NLP and Feature Engineering Strategies," 2023. doi: https://doi.org/10.21203/rs.3.rs-3446830/v1.

[27] C. Li, H. Li, Z. Liu, B. Li, and Y. Huang, “SeedSortNet: a rapid and highly effificient lightweight CNN based on visual attention for seed sorting,” *PeerJ Comput. Sci.*, vol. 7, pp. 1–21, 2021, doi: https://doi.org/10.7717/peerj-cs.639.

[28] L. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.