

SQL Injection Detection Using Character-Level Embedding with Convolutional Neural Network and Gated Recurrent Unit

Ahmad Andri Alfian¹, Subhan^{2*}

^{1,2}Computer Science Department, Faculty of Mathematics and Natural Sciences,
Universitas Negeri Semarang, Indonesia

Abstract. Among the many threats facing web applications, SQL injection is still one of the most widely encountered, as it allows attackers to alter SQL queries to obtain unauthorized access to a database or compromise the information stored within it. Existing approaches for SQL injection detection primarily rely on handcrafted features or word-based representations, limiting their ability to effectively represent the fine-grained syntactic information embedded in SQL queries.

Purpose: This study proposes and evaluates a SQL injection detection model that integrates Character-Level Embedding with a Convolutional Neural Network, and a Gated Recurrent Unit. Because the model retains fine-grained character-level information, it is expected to classify normal and malicious SQL queries more accurately.

Methods: The proposed model combines Character-Level Embedding, CNN, and GRU to perform binary classification of SQL queries. Experiments were carried out using the Kaggle SQL Injection Dataset, which consists of 30,907 SQL queries in total, including 19,529 legitimate queries and 11,378 SQL injection queries. After character-level tokenization and sequence padding, the data were split into training, validation, and testing sets with an 80:10:10 ratio. The model was built in TensorFlow and Keras, trained using the Adam optimizer and Binary Cross-entropy loss, then evaluated using Accuracy, Precision, Recall, F1-score, and Confusion Matrix.

Result: Evaluation results show that the proposed hybrid CNN-GRU model performs well in classifying legitimate and malicious SQL queries, reaching 99.64% Accuracy, 100% Precision, 99.03% Recall, and 99.51% F1-score.

Novelty: By pairing Character-Level Embedding with a hybrid CNN-GRU design, the proposed model is able to capture both localized character patterns and longer-range sequential relationships without relying on manual feature engineering. This approach therefore offers a dependable alternative for SQL injection detection and contributes to stronger web application security.

Keywords: Character-Level Embedding, Convolutional Neural Network, SQL Injection Detection, Gated Recurrent Unit

Received August 2026 / **Revised** August 2026 / **Accepted** September 2026

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).



INTRODUCTION

Web applications have become a critical component in various sectors, including government, e-commerce, healthcare, finance, and education [1]. Along with the rapid growth of web-based information systems, the number of cyberattacks targeting these applications has also increased significantly [2]. One of the most common attacks is Structured Query Language Injection (SQLi) [3], in which attackers take advantage of weak input validation to insert malicious SQL commands into an application's queries. When an SQL Injection attack succeeds, it can undermine the confidentiality, integrity, and availability of a database, giving attackers the ability to retrieve, alter, or delete sensitive data, and in more severe cases, gain full control over the database [4].

SQL injection has long been regarded as one of the most critical threats to web application security, being consistently listed among the Open Worldwide Application Security Project (OWASP) top ten security risks for modern web applications [5]. Although traditional countermeasures, including rule-based detection, signature-based methods, input validation, and Web Application Firewalls (WAFs), remain widely implemented, attackers have continued to develop obfuscation strategies capable of circumventing these mechanisms. Rule-based detection, in particular, can be evaded through the modification of query structures, the insertion of comments, payload encoding, or the use of alternative SQL syntax, thereby reducing the effectiveness of conventional detection approaches in dynamic attack environments [6].

*Corresponding author.

Email addresses: subhan@mail.unnes.ac.id (Subhan)

DOI: 10.15294/rji.v4i2.62096

To address these limitations, machine learning and deep learning approaches have increasingly been applied to SQL Injection detection [7]. Unlike traditional approaches that depend on manually defined rules or handcrafted features, deep learning models can automatically learn representative features directly from SQL queries. As a result, such methods have shown superior performance in identifying both known and previously unseen attack patterns [4].

Recent studies have demonstrated that deep learning and machine learning techniques are effective for detecting SQL injection attacks. Conventional machine learning approaches typically depend on manually engineered features combined with classifiers such as Random Forest, Naïve Bayes, Logistic Regression, and AdaBoost. Although these methods have achieved promising classification results, their performance remains closely tied to the quality of the feature extraction strategy employed [8], [9]. More recently, deep learning-based methods, including Deep Neural Networks (DNN), Recurrent Neural Network (RNN) Autoencoders, and hybrid architectures such as SIDNet, have been proposed to automatically learn feature representations from SQL queries and achieve higher detection accuracy [4], [6], [10]. Collectively, these findings suggest that deep learning offers a more effective alternative to conventional approaches by learning distinct features directly from the data itself. Table 1 presents a summary of deep learning-based methods for SQL Injection detection alongside Character-Level Embedding approaches used for text classification.

Table 1. Summary of Previous Study

Ref.	Method	Result
[6]	SIDNet-1 and SIDNet-2	Achieved the highest accuracy of 98.02 and 97.83%.
[11]	RNN autoencoder	Achieved a classification accuracy of 94.00%.
[9]	Bert Encoding and AdaBoost	Achieved an accuracy of 98.90%.
[10]	RNN-LSTM and RNN-GRU	Achieved accuracies of 90.11% and 84.89%.
[8]	Logistic Regression, AdaBoost, Random Forest, Naive Bayes, and XGBoost	Naive Bayes achieved the highest accuracy of 98.33%.
[4]	SQLNN deep neural network	Achieved an accuracy of 96.00%.
[12]	Character-Level Convolutional Autoencoder	Achieved an accuracy of 91.10%
[13]	URL2Graph++ with CharCNN, BERT, and Graph Neural Network	Achieved an accuracy of 89.20%

However, most previous studies represent SQL queries at the word or token level. Such representations may overlook important character-level information that frequently distinguishes malicious queries from legitimate ones. SQL Injection payloads commonly exploit special characters, quotation marks, logical operators, comments, delimiters, and other symbols whose semantic meaning depends heavily on their character composition [14]. Therefore, character-level representation is expected to preserve richer syntactic information and enhance the model's capability to identify slight attack variations.

Character-Level Embedding has demonstrated strong results across various cybersecurity applications by learning feature representations directly from character sequences without relying on manual feature engineering. This approach is particularly suitable for processing structured textual data that contain numerous symbols and syntax patterns, such as URLs, source code, and SQL queries [12], [13]. By modeling information at the character level, the embedding process can better capture lexical variations commonly exploited by attackers to evade detection.

Based on the identified research gaps, this study proposes a SQL Injection detection model that integrates Character-Level Embedding with a hybrid CNN-GRU architecture. Unlike previous SQL Injection detection studies that primarily employ word-level or token-level representations, the proposed approach represents SQL queries at the character level, enabling the preservation of fine-grained syntactic information such as quotation marks, SQL operators, comments, delimiters, and other special characters that frequently characterize SQL Injection payloads. Character-Level Embedding converts SQL queries into dense vector representations while preserving character-level semantics. Subsequently, CNN is employed to extract local structural patterns from character sequences, whereas GRU captures long-term sequential dependencies among characters. By combining Character-Level Embedding, CNN, and GRU within a unified architecture, the proposed architecture is expected to simultaneously learn both local and contextual characteristics of SQL queries, providing a more comprehensive feature representation for SQL Injection detection than existing token-level approaches.

This research focuses on developing and evaluating a CNN-GRU model that utilizes Character-Level Embedding to detect SQL injection attacks. The model is evaluated on a publicly available SQL injection dataset using Accuracy, Precision, Recall, F1-score, and Confusion Matrix as performance measures. The proposed method aims to provide an effective deep learning-based solution that can complement existing SQL injection detection techniques.

METHODS

Research Approach and Design

The research was conducted using a quantitative methodology, with a deep learning-based model developed for the classification of SQL queries as either legitimate or malicious. Character-Level Embedding, Convolutional Neural Network (CNN), and Gated Recurrent Unit (GRU) were integrated into the proposed architecture. The experimental procedure covered data preparation and preprocessing, character-level tokenization, sequence padding, dataset splitting, model implementation and training, followed by performance evaluation.

Data acquisition was performed using a publicly available dataset containing SQL injection queries. The collected data were prepared for deep learning through a preprocessing stage, where each query was decomposed into individual characters using Character-Level Tokenization. Padding was subsequently applied to accommodate the different sequence lengths and generate fixed-size inputs. The prepared dataset was then partitioned into training, validation, and testing subsets using an 80:10:10 ratio.

The training data were utilized for parameter optimization of the hybrid CNN-GRU architecture, while validation data were used to observe the learning behavior and minimize the likelihood of overfitting [15]. Once training was completed, model performance was assessed on the testing data using Accuracy, Precision, Recall, F1-score, and Confusion Matrix. The research workflow is depicted in Figure 1.

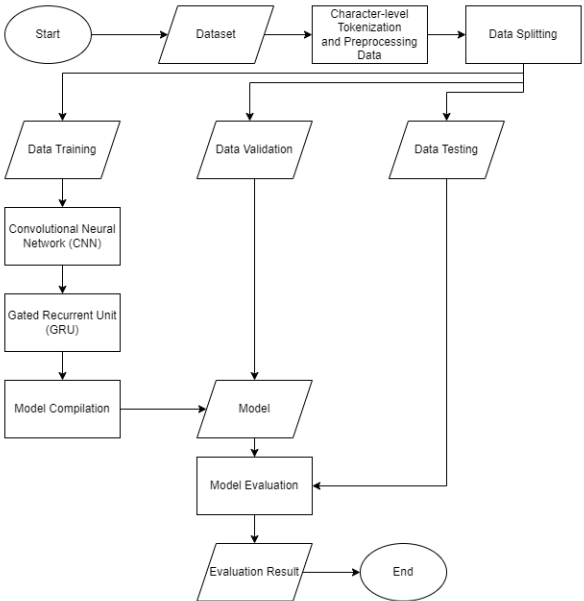


Figure 1. Research Flowchart

Dataset

The dataset used in this study was obtained from the publicly available SQL Injection Dataset provided on Kaggle [16]. The dataset contains 30,919 SQL queries consisting of 19,589 legitimate queries and 11,330 SQL Injection queries. Each SQL statement is labeled as legitimate or malicious.

The dataset encompasses several categories of SQL injection attacks, including authentication bypass, UNION-based, Boolean-based, and comment-based techniques. Such diversity provides the model with a range of attack characteristics from which representative patterns can be learned, supporting the detection of various SQL injection query structures. Table 2 details the dataset composition by presenting the class

labels, sample SQL queries, record counts for each class, and the total number of queries analyzed in this research.

Table 2. Dataset Class

Class	SQL query sample	Amount
Normal Query	SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK') ;	19,537
SQL Injection Query	select * from users where id = 1 or ".@" or 1 = 1 -- 1	11,382
Total		30,919

Character-Level Tokenization and Data Preprocessing

Unlike conventional Natural Language Processing approaches that tokenize text into words, this study employs Character-Level Tokenization. Each SQL query is decomposed into individual characters, including alphabetic characters, digits, operators, whitespace, punctuation marks, quotation marks, parentheses, and SQL symbols. Subsequently, each unique character is assigned a numerical index using the Keras Tokenizer configured in character-level mode.

Character-level representation enables the model to preserve detailed lexical and syntactic information [17] that is frequently exploited in SQL Injection attacks. Therefore, subtle modifications involving symbols and special characters can still be recognized during the learning process. Table 3 shows an example of the character-level tokenization process used in this study.

Table 3. Example of Character-Level Tokenization

Original SQL Query	Character-Level Split	Character-Level Tokenization
select * from users where id = 1 or " (]" or 1 = 1 -- 1	[], [s], [e], [l], [e], [c], [t], [], [], [f], [r], [o], [m], [], [w], [s], [e], [r], [s], [], [w], [h], [e], [r], [e], [], [i], [d], [], [], [=], [], [], [1], [], [o], [r], [], [], [], [], [l], [], [], [], [], [o], [r], [], [1], [], [], [=], [], [], [1], [], [], [], [], [1]	[1], [4], [2], [6], [2], [9], [5], [1], [39], [1], [21], [3], [8], [16], [1], [19], [4], [2], [3], [4], [1], [31], [15], [2], [3], [2], [1], [14], [17], [1], [1], [32], [1], [1], [12], [1], [8], [3], [1], [40], [1], [1], [13], [1], [1], [58], [40], [1], [8], [3], [1], [12], [1], [1], [32], [1], [1], [12], [1], [38], [38], [1], [12]

Data preprocessing was carried out to convert raw SQL queries into numerical representations suitable for processing by the deep learning model. Since SQL queries vary considerably in length, the resulting character sequences cannot be directly processed in batches. To standardize the input data, sequence padding was applied so that all character sequences had the same length. Sequences that did not reach the predefined maximum length were completed using zero-padding, and those that surpassed it were shortened through truncation. This step ensures that input dimensions stay uniform throughout the training and inference stages.

Data Splitting

The preprocessed dataset was subsequently randomized and allocated to training, validation, and testing sets at a ratio of 80:10:10. Parameter learning was conducted using the training set, with the validation set serving as a basis for monitoring model learning and optimizing hyperparameters. Final performance assessment was performed exclusively on the testing set, which contained SQL queries unseen during the training process.

To maintain a balanced class distribution, the dataset splitting process preserved the proportion of legitimate SQL queries and SQL Injection queries across all subsets. Table 4 presented the distribution of samples used in this study.

Table 4. Data Splitting Result

Dataset	Normal SQL Query	SQL Injection Query	Total
Training (80%)	15,638	9,097	24,735
Validation (10%)	1,948	1,144	3,092
Testing (10%)	1,951	1,141	3,092
Total	19,537	11,382	30,919

Convolutional Neural Network

Local patterns in the character sequences are extracted using a one-dimensional Convolutional Neural Network (CNN). Each input character is first represented as a dense vector using an Embedding layer, which converts character indices into continuous vector representations. This representation enables the

model to learn relationships among characters while retaining syntactic patterns present in SQL queries. The Embedding layer employs 50-dimensional vectors, with the input sequences fixed at a maximum length of 300 characters.

The resulting character embeddings are subsequently passed through a one-dimensional convolutional (Conv1D) layer that applies the Rectified Linear Unit (ReLU) as an activation function, a kernel size of 3, and 64 filters. Through this convolution operation, multiple filters work to detect local patterns, including combinations of SQL keywords, operators, quotation marks, comments, and special characters that commonly appear within SQL Injection payloads. According to [18], modern convolutional neural networks perform feature extraction using the cross-correlation operation, in which the convolution kernel is applied directly to the input sequence without kernel reversal. The one-dimensional convolution operation adopted in this study is presented in Equation 1. Subsequently, the convolution output is transformed using the ReLU activation function shown in Equation 2 [19].

$$y_i = \sum_{j=0}^{k-1} x_{i+j} w_j + b \quad (1)$$

where:

x_{i+j} is the input embedding vector,

w_j is the convolution kernel,

k is the kernel size,

b is the bias.

$$f(x) = \max(0, x) \quad (2)$$

The feature maps generated by the convolutional layer are subsequently downsampled using MaxPooling1D with a pool size of 2. Through this operation, the dimensionality of the feature representation is reduced while important features are retained. Computational efficiency is thereby improved and redundant information is reduced before the extracted features move on to the next stage of the network. The max-pooling operation is defined in Equation 3 [18].

$$p_i = \max(x_i, x_{i+1}, \dots, x_{i+m-1}) \quad (3)$$

where m denotes the pooling window size.

Gated Recurrent Unit

Once generated by the CNN, the feature maps are further processed through a Gated Recurrent Unit (GRU). GRU represents a recurrent neural network architecture developed to capture sequential dependencies within input sequences, while also addressing the vanishing gradient problem that commonly affects conventional recurrent neural networks [20].

Unlike CNN, which focuses on extracting local features, the GRU layer learns contextual relationships among characters across an entire SQL query. Through this capability, the proposed model becomes able to recognize complex sequential patterns that differentiate legitimate SQL statements from malicious SQL Injection payloads. Compared with Long Short-Term Memory (LSTM), GRU utilizes fewer trainable parameters, resulting in lower computational complexity while maintaining competitive performance in sequence modeling.

Sequential dependencies are captured using a GRU layer configured with 64 hidden units, followed by Dropout regularization at a rate of 0.5. The learned representations are passed to a 64-neuron fully connected layer using ReLU activation, and classification is subsequently produced by a single Sigmoid-activated output neuron. The GRU controls information flow across time steps through its update and reset gating mechanisms. The mathematical formulation of the update gate, reset gate, candidate hidden state, and hidden state update is presented in Equations 4, 5, 6, and 7 [21], [22]. The final output probability for binary classification is obtained using the Sigmoid activation function as defined in Equation 8 [18].

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (4)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (5)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \quad (6)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (7)$$

$$\hat{y} = \frac{1}{1 + e^{-x}} \quad (8)$$

Model Compilation and Training

Once the network architecture had been established, Adam was selected as the optimization algorithm with a learning rate of 0.001 due to its adaptive learning behavior and efficient convergence. Binary Cross-entropy served as the objective function for evaluating the discrepancy between model predictions and ground-truth labels in the binary SQL injection classification task. Training performance was subsequently monitored using classification accuracy.

A batch of 32 was used throughout the 10 training epochs. The training data served as the basis for parameter optimization, while the validation data were used after each epoch to evaluate learning progression and model generalization. This evaluation procedure allowed the training behavior to be monitored continuously and potential overfitting to be detected. The network parameters were optimized via backpropagation with the Adam optimizer, with no early stopping applied. All experiments were performed on the Kaggle Notebook platform using the default CPU runtime environment. The proposed model was implemented in Python with TensorFlow and Keras for model construction and training, whereas Scikit-learn was utilized to evaluate model performance.

Model Evaluation

Model performance was assessed using the testing dataset that was excluded from the training process. To summarize the classification results, the predicted and actual labels were compared through a confusion matrix [23]. The matrix categorizes the predictions into True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), providing the fundamental values required for calculating the evaluation metrics. The confusion matrix is presented in Table 5.

Table 5. Confusion Matrix

		Predict	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Model performance was assessed using Accuracy, Precision, Recall, and F1-score, with the values obtained from the confusion matrix. Accuracy reflects the overall proportion of correct classifications, while Precision indicates how accurately the model identifies queries classified as SQL injection attacks. Recall represents the proportion of actual malicious queries successfully detected by the model. Meanwhile, F1-score provides a balanced assessment by calculating the harmonic mean of Precision and Recall. The corresponding formulas are presented in Equations 9, 10, 11, and 12.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$Precision = \frac{TP}{TP + FP} \quad (10)$$

$$Recall = \frac{TP}{TP + FN} \quad (11)$$

$$F1 - score = \frac{2 \times Precision \times recall}{Precision + recall} \quad (12)$$

RESULT AND DISCUSSION

Model Development

Implementation of the proposed Character-Level Embedding CNN-GRU architecture was carried out with TensorFlow and Keras. The input characters are first represented as dense vectors through the Embedding layer. Local character features are then extracted by a Conv1D layer employing 64 filters with a kernel size of 3. After the feature representations are downsampled using MaxPooling1D, they are supplied to a GRU layer comprising 64 hidden units to capture sequential dependencies across the SQL query sequences.

To improve generalization, two Dropout layers with a rate of 0.5 are incorporated after the GRU and fully connected layers. A Dense layer with Sigmoid activation is then used as the output layer to classify each input query as either legitimate or SQL injection. Table 6 demonstrates the proposed model's complete architecture.

Table 6. Proposed CNN-GRU Architecture

Layer (Type)	Configuration	Output Shape	Parameters
Input	Sequence Length = 300	300	0
Embedding	Output Dimension = 50	300 x 50	4,100
Conv1D	Filters = 64, Kernel Size = 3, Activation = ReLU	298 x 64	9,664
MaxPooling1D	Pool Size = 2	149 x 64	0
GRU	Units = 64	64	24,960
Dropout (1)	Rate = 0.5	64	0
Dense (1)	Units = 64, Activation = ReLU	64	4,160
Dropout (2)	Rate = 0.5	64	0
Dense (2)	Units = 1, Activation Sigmoid	1	65
Total Parameters			42,949
Trainable Parameters			42,949
Non-trainable Parameters			0

The resulting architecture contains 42,949 trainable parameters and no non-trainable parameters. This relatively compact parameter suggests that the proposed model remains computationally efficient while still being capable of learning representative character-level features for SQL Injection detection. The established architecture is subsequently used for the training and evaluation processes discussed in the next sections.

Training Results

After the preprocessing stages had been completed, the training process was conducted using the designated training data. The model learned representations of character-level patterns and associated them with the corresponding SQL query classes. The learning process was monitored based on accuracy and loss obtained from both the training and validation sets. Figure 2 displays the accuracy curves, while Figure 3 presents the training and validation loss curves.



Figure 2. Training and Validation Accuracy

According to Figure 2, the accuracy of both the training and validation sets improved substantially during the initial epoch and then converged toward a stable performance level. This trend indicates effective learning of the classification patterns by the proposed model. Figure 3 further shows that the training and

validation losses decreased as the number of epochs increased, with a small fluctuation in validation loss observed near the end of training. The limited discrepancy between the training and validation curves provides an indication of satisfactory generalization to unseen data. In addition, the stable validation accuracy and relatively narrow accuracy gap suggest that the model did not experience substantial overfitting. The minor increase in validation loss during the final epoch did not result in reduced validation accuracy, indicating relatively stable model performance. These observations demonstrate that the CNN-GRU architecture effectively learned representative characteristics from SQL queries without exhibiting considerable instability during training.

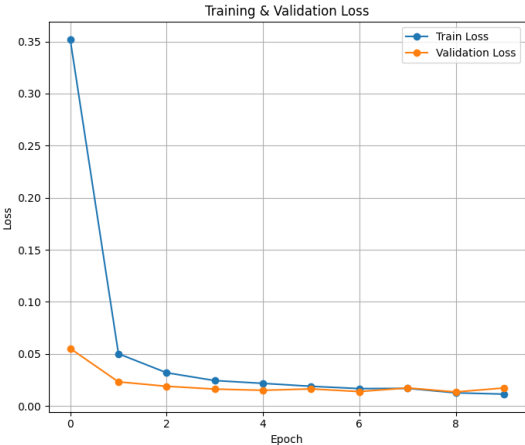


Figure 3. Training and Validation Loss

Performance Evaluation

The trained model was subsequently evaluated using the testing dataset. Classification performance was analyzed through a confusion matrix, with Accuracy, Precision, Recall, and F1-score calculated based on its results. The confusion matrix obtained during testing is shown in Figure 4.

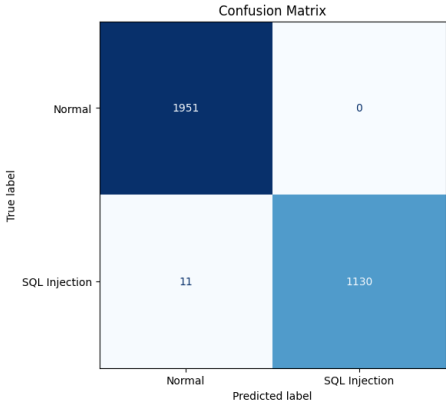


Figure 4. Confusion Matrix

The results presented in the confusion matrix show a high number of correct classifications for both normal and SQL injection queries, with relatively few incorrect predictions. This demonstrates the effectiveness of the proposed model in differentiating malicious SQL queries from legitimate ones. The complete evaluation results are summarized in Table 7.

Model	Accuracy	Precision	Recall	F1-score
CNN	99.56%	99.64%	99.17%	99.40%
Character-Level Embedding + CNN-GRU	99.64%	100%	99.03%	99.51%

The proposed architecture achieved favorable results across Accuracy, Precision, Recall, and F1-score. Its high precision suggests effective control of false positive classifications, while the high recall indicates that most malicious SQL injection queries were correctly detected. The resulting F1-score, which balances both

measures, further demonstrates the model's reliability in distinguishing SQL injection attacks from legitimate queries.

The comparison with the baseline CNN shows that the proposed Character-Level Embedding CNN-GRU architecture achieved higher Accuracy, Precision, and F1-score, despite the relatively modest improvement. The integration of the GRU component allows the model to complement local character-level feature extraction with sequential dependency modeling, contributing to improved classification results. While Recall decreased slightly from 99.17% to 99.03%, the reduction was minimal and had no substantial effect on the model's overall performance. These findings indicate that the proposed architecture offers a favorable balance between classification accuracy and generalization capability relative to the CNN model.

Performance Comparison with Previous Studies

Further assessment was conducted by benchmarking the proposed approach against previous research employing machine learning and deep learning techniques for SQL injection detection. The comparative results between the proposed model and the existing approaches are summarized in Table 8.

Table 8. Comparison with Previous Studies			
Ref.	Method	Dataset	Result (Accuracy)
[6]	SIDNet-1 and SIDNet-2	SQL Injection	98.02 and 97.83%
[11]	RNN autoencoder	SQL Injection	94.00%
[9]	Bert Encoding and AdaBoost	SQL Injection	98.90%
[10]	RNN-LSTM and RNN-GRU	SQL Injection	90.11% and 84.89%
[8]	Logistic Regression, AdaBoost, Random Forest, Naive Bayes, and XGBoost	SQL Injection	The highest accuracy 98.33%
[4]	SQLNN deep neural network	SQL Injection	96.00%
	Proposed Method	Character-Level Embedding + CNN-GRU	SQL Injection 99.64%

The comparison indicates that the proposed Character-Level Embedding CNN-GRU model achieves competitive performance compared with previous SQL Injection detection methods. Traditional machine learning algorithms generally depend on manually engineered features, whereas the proposed approach automatically learns representative features directly from SQL queries through deep learning. Similarly, when compared with recurrent-only architectures, the integration of CNN and GRU enables the model to capture local character patterns together with long-range sequential dependencies simultaneously.

Character-Level Embedding further improves the model by preserving fine-grained syntactic information such as quotation marks, parentheses, SQL operators, comments, and special symbols commonly found in SQL Injection payloads. As a result, the proposed architecture can identify malicious query structures more effectively than approaches relying solely on word-level representations. Overall, these findings indicate that combining Character-Level Embedding, CNN, and GRU provides a robust approach for SQL Injection detection while maintaining competitive performance compared with recent state-of-the-art methods.

CONCLUSION

The research presented a hybrid CNN-GRU architecture incorporating Character-Level Embedding for SQL injection detection. Evaluation results demonstrated the model's ability to classify SQL queries effectively by leveraging character-level information and sequential relationships within the input data. The proposed approach therefore shows potential as an alternative deep learning technique for SQL injection detection and for supporting enhanced web application security. Further investigation using more extensive and diverse datasets is recommended to assess and improve the model's robustness and generalization capability.

REFERENCES

- [1] European Union Agency for Cybersecurity., *ENISA threat landscape 2023: July 2022 to June 2023*. LU: Publications Office, 2023. doi: 10.2824/782573.
- [2] A. I. Mallick and R. Nath, "Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments," 2024.
- [3] T. Li *et al.*, "A Survey on Web Application Testing: A Decade of Evolution," Apr. 25, 2025, *arXiv*: arXiv:2412.10476. doi: 10.48550/arXiv.2412.10476.
- [4] W. Zhang *et al.*, "Deep Neural Network-Based SQL Injection Detection Method," *Secur. Commun. Netw.*, vol. 2022, pp. 1–9, Mar. 2022, doi: 10.1155/2022/4836289.
- [5] "OWASP Top 10:2025." Accessed: Feb. 23, 2026. [Online]. Available: <https://owasp.org/Top10/2025/>

- [6] D. Muduli *et al.*, “SIDNet: A SQL Injection Detection Network for Enhancing Cybersecurity,” *IEEE Access*, vol. 12, pp. 176511–176526, 2024, doi: 10.1109/ACCESS.2024.3502293.
- [7] H. Sun, Y. Du, and Q. Li, “Deep Learning-Based Detection Technology for SQL Injection Research and Implementation,” *Appl. Sci.*, vol. 13, no. 16, p. 9466, Aug. 2023, doi: 10.3390/app13169466.
- [8] P. Roy, R. Kumar, and P. Rani, “SQL Injection Attack Detection by Machine Learning Classifier,” in *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India: IEEE, May 2022, pp. 394–400. doi: 10.1109/ICAAIC53929.2022.9792964.
- [9] M. Zivkovic *et al.*, “Optimizing SQL injection detection using BERT encoding and AdaBoost Classification,” in *Proceedings of the 2nd International Conference on Innovation in Information Technology and Business (ICIITB 2024)*, vol. 113, N. Bacanin and H. Shaker, Eds., in *Advances in Computer Science Research*, vol. 113., Dordrecht: Atlantis Press International BV, 2024, pp. 137–154. doi: 10.2991/978-94-6463-482-2_10.
- [10] A. ALazzawi, “SQL Injection Detection Using RNN Deep Learning Model,” *J. Appl. Eng. Technol. Sci. JAETS*, vol. 5, no. 1, pp. 531–541, Dec. 2023, doi: 10.37385/jaets.v5i1.2864.
- [11] M. Alghawazi, D. Alghazzawi, and S. Alarifi, “Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model,” *Mathematics*, vol. 11, no. 15, p. 3286, Jul. 2023, doi: 10.3390/math11153286.
- [12] S.-J. Bu and S.-B. Cho, “Deep Character-Level Anomaly Detection Based on a Convolutional Autoencoder for Zero-Day Phishing URL Detection,” *Electronics*, vol. 10, no. 12, p. 1492, Jun. 2021, doi: 10.3390/electronics10121492.
- [13] Y. Tian, Y. Jia, J. Sun, Y. Wang, Z. Liu, and X. Ling, “URL2Graph++: Unified semantic-structural-character learning for malicious URL detection,” *Inf. Fusion*, 2026.
- [14] J. Choi, Y.-A. Jung, and H. Ko, “Comparative Analysis of SQL Injection Defense Mechanisms Based on Three Approaches: PDO, PVT, and ART,” *Appl. Sci.*, vol. 15, no. 23, p. 12351, Nov. 2025, doi: 10.3390/app152312351.
- [15] T. Burzykowski, M. Geubbelmans, A.-J. Rousseau, and D. Valkenborg, “Validation of machine learning algorithms,” *Am. J. Orthod. Dentofacial Orthop.*, vol. 164, no. 2, pp. 295–297, Aug. 2023, doi: 10.1016/j.ajodo.2023.05.007.
- [16] “SQL Injection Dataset.” Accessed: Jul. 20, 2026. [Online]. Available: <https://www.kaggle.com/datasets/sajid576/sql-injection-dataset>
- [17] S. J. Mielke *et al.*, “Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP,” Dec. 20, 2021, *arXiv: arXiv:2112.10508*. doi: 10.48550/arXiv.2112.10508.
- [18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. in *Adaptive Computation and Machine Learning series*. MIT Press, 2016. [Online]. Available: <https://books.google.co.id/books?id=omivDQAAQBAJ>
- [19] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, in ICML’10. Madison, WI, USA: Omnipress, 2010, pp. 807–814.
- [20] I. D. Mienye, T. G. Swart, and G. Obaido, “Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications,” *Information*, vol. 15, no. 9, p. 517, Aug. 2024, doi: 10.3390/info15090517.
- [21] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” Sep. 03, 2014, *arXiv: arXiv:1406.1078*. doi: 10.48550/arXiv.1406.1078.
- [22] Y. Hu, H. Jin, X. Chu, and Y. Yi, “Transfer Learning-Based Piezoelectric Actuators Feedforward Control with GRU-CNN,” *Appl. Sci.*, vol. 16, no. 3, p. 1305, Jan. 2026, doi: 10.3390/app16031305.
- [23] M. Grandini, E. Bagli, and G. Visani, “Metrics for Multi-Class Classification: an Overview,” Aug. 13, 2020, *arXiv: arXiv:2008.05756*. doi: 10.48550/arXiv.2008.05756.