mathematics

MDPI

Article

Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model

Maha Alghawazi *, Daniyal Alghazzawi and Suaad Alarifi

Information Systems Department, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 80200, Saudi Arabia; dghazzawi@kau.edu.sa (D.A.); salarifi@kau.edu.sa (S.A.)
* Correspondence: mmohammadalqhtani@stu.kau.edu.sa

Abstract: SQL injection attacks are one of the most common types of attacks on Web applications. These attacks exploit vulnerabilities in an application's database access mechanisms, allowing attackers to execute unauthorized SQL queries. In this study, we propose an architecture for detecting SQL injection attacks using a recurrent neural network autoencoder. The proposed architecture was trained on a publicly available dataset of SQL injection attacks. Then, it was compared with several other machine learning models, including ANN, CNN, decision tree, naive Bayes, SVM, random forest, and logistic regression models. The experimental results showed that the proposed approach achieved an accuracy of 94% and an F1-score of 92%, which demonstrate its effectiveness in detecting QL injection attacks with high accuracy in comparison to the other models covered in the study.

Keywords: SQL injection attacks; recurrent neural network (RNN) autoencoder; ANN; CNN; decision tree; naive Bayes; SVM; random forest; logistic regression

MSC: 68T99

Citation: Alqhwazi, M.; Alghazzawi, D.; Alarifi, S. Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model. *Mathematics* **2023**, *11*, 3286. https://doi.org/10.3390/math11153286

Academic Editor: Tao Zhou

Received: 7 July 2023
Revised: 23 July 2023
Accepted: 24 July 2023
Published: 26 July 2023

1. Introduction

Structured query language (SQL) is a programming language used to manage, organize, and manipulate relational databases. It also allows the user or an application program to interact with a database by inserting new data, deleting old data, and changing previously stored data. Structured query language injection attacks (SQLIAs) pose a severe security threat to Web applications [1]. These attacks involve the malicious execution of SQL queries on a server, enabling unauthorized access to and retrieval of restricted data stored within databases [2]. Figure 1 illustrates the basic process of an SQLIA.

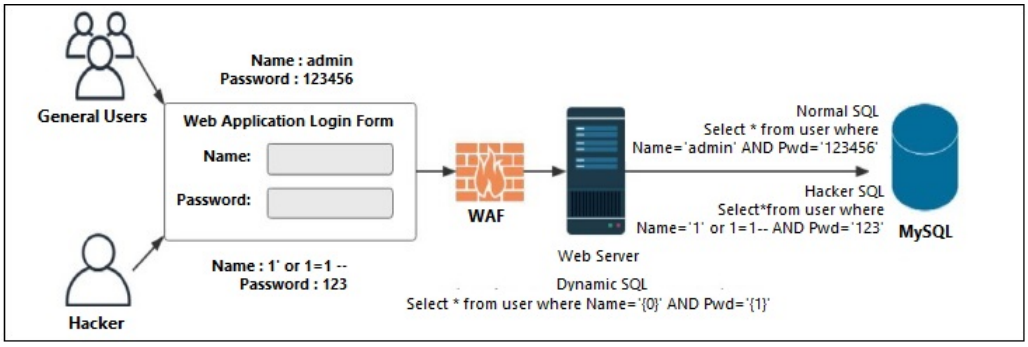

Figure 1. SQL injection attack process adopted from [3].

Attackers can exploit Web applications by injecting SQL statements or sending special symbols through user input to target the database tier and gain unauthorized access to valuable assets [3]. Due to the absence of proper validation in some Web applications, which

is usually the programmer's fault, attackers can bypass authentication mechanisms and gain access to databases, enabling them to retrieve or manipulate data without appropriate authorization [2].

In recent years, researchers have proposed many detection methods, including machine learning algorithms and deep neural network models. Deep neural networks, also known as deep learning, are a rapidly evolving research area within the field of machine learning. They were developed to bring machine learning closer to its original goal of achieving artificial intelligence. Deep learning involves training complex models that can learn the underlying patterns and representations of large datasets. This has proven to be a powerful technique for interpreting various forms of data, including text, images, and sounds. Deep learning has also been successfully applied to Web security detection, highlighting its potential impact on a broad range of applications [4]. However, one of the major drawbacks of using neural networks is their tendency to make overconfident predictions. This means that they have a high degree of certainty in their predictions, even when they are incorrect [5,6]. Even though the models perform well on test data from the same distribution as the training data, they do not know the limits of their knowledge and make erroneous guesses outside that domain. This pitfall arises because neural networks learn highly nonlinear functions that do not output calibrated probability estimates for unfamiliar data [7]. To address this issue, researchers have developed various techniques for estimating predictive uncertainty in neural networks. Lakshminarayanan et al. [7] introduced deep ensembles, where multiple models are independently trained on the same data and their predictions are averaged to capture model uncertainty. Mishra et al. [5] evaluated Bayesian neural networks (BNNs) as a technique that can provide accurate predictions along with reliably quantified uncertainties. Amodei et al. [6] suggested using model rollouts/lookahead during training to avoid reward hacking, improve safety, and reduce overconfidence. In summary, while neural networks have shown great promise in many applications, it is important to be aware of their tendency to make overconfident predictions and the potential pitfalls of overfitting. Estimating predictive uncertainty using techniques such as deep ensembles can help mitigate these issues and improve the reliability of neural network predictions.

Detection of SQL injection attacks is crucial to ensure the security and integrity of Web applications and their associated data. To address this issue, a deep learning architecture based on the recurrent neural network (RNN) autoencoder model is proposed for detecting SQL injection attacks. The RNN autoencoder is a special case of the RNN-based encoder–decoder (RNN-ED) model. The autoencoder consists of an encoder RNN that encodes the input sequence into a hidden state and a decoder RNN that decodes the hidden state back into the original input sequence. The encoder and decoder RNNs are trained jointly using backpropagation to minimize the reconstruction error between the input and output sequences [8].

The aim of this study was to develop an architecture based on a recurrent neural network (RNN) autoencoder to detect SQL injection attacks. Moreover, the proposed approach that addresses this attack is discussed and compared with other approaches. The research questions were:

Q1: Is the proposed RNN autoencoder-based architecture effective for detecting SQL injection attacks?
Q2: How can the RNN autoencoder be optimized to improve its performance in detecting SQL injection attacks?
Q3: Can an RNN autoencoder outperform other SQL injection attack machine learning detection models?

The main contributions of this paper are as follows:

- Proposing an SQLIA detection architecture based on a recurrent neural network (RNN) autoencoder algorithm;
- Comparing the proposed architecture and different machine learning techniques used for detecting and preventing SQLIAs.

The paper is structured as follows: Section 2 reviews the related research in this area. The methodology is discussed in Section 3. Experiment results and the discussion are shown in Section 4. The last section provides the conclusion and discusses future work.

2. Literature Review

This section explores a variety of ML and DL techniques found in the literature for the detection of SQL injection attacks.

Ketema [1] used a deep learning convolutional neural network (CNN) to build a model to prevent an SQLI using a public benchmark dataset. The model was trained using deep learning with different hyperparameter values and five different scenarios. The model achieved an accuracy of 97%. Roy et al. [9] presented a method for detecting SQL injection attacks using machine learning classifiers. The authors used five ML classifiers (logistic regression, AdaBoost, naive Bayes, XGBoost, and random forest) to classify SQL queries as either legitimate or malicious. The proposed model was trained and evaluated using a publicly available dataset of SQL injection attacks on Kaggle. The results of the study showed that the best performance was achieved by the naive Bayes classifier, with an accuracy of 98.33%. Finally, the authors performed a comparison with previous work. Overall, the study demonstrated the potential of machine learning classifiers in improving the accuracy and efficiency of SQL injection attack detection.

S.S. Anandha Krishnan et al. [10] proposed a machine learning-based approach for detecting SQL injection attacks. The authors argued that traditional signature-based approaches are ineffective against advanced attacks, and machine learning can help address this issue. The authors first described the various types of SQL injection attacks and their impact on Web applications. They then outlined the proposed framework, which consisted of preprocessing the data, feature extraction, model training, and evaluation. The results showed that the CNN classifier model performed better than the other classifiers in terms of accuracy, precision, recall, and F1-score. Rahul et al. [11] proposed a novel method of protecting against SQL injection and cross-site scripting (XSS) attacks by augmenting the Web application firewall (WAF) with a honeypot. The WAF filters incoming traffic using established patterns, while the honeypot is designed to attract attackers and capture information about their attack methods, which is then used to improve the WAF's ability to detect and prevent future attacks. The proposed method was evaluated through experiments, and the results suggested that the combination of a honeypot and WAF can effectively protect Web applications from these types of attacks.

Zhang et al. [4] proposed a method for detecting SQL injection attacks using a deep neural network. The authors stated that traditional methods of SQL injection attack detection have limitations, prompting the development of their new approach. The authors gathered a dataset of clean and malicious queries and used it to train a deep neural network classifier with several layers. They then compared the result of the proposed method with the traditional machine learning algorithms, including KNN, DT, and LSTM algorithms. Liu et al. [12] proposed a new approach called DeepSQLi for the automated detection of SQL injection vulnerabilities in Web applications using deep semantic learning techniques. DeepSQLi uses a deep neural network to learn the semantic meanings of SQL queries and identify potential injection vulnerabilities. The model is trained using a dataset of benign and malicious SQL queries and leverages multiple layers of convolutional and recurrent neural networks. The experimental results showed that DeepSQLi outperformed SQLmap, and more SQLi attacks could be identified faster while using a lower number of test cases. Chen et al. [3] presented a novel approach for detecting and preventing SQL injection attacks on Web applications using deep learning algorithms. The authors trained and evaluated the performance of a convolutional neural network (CNN) and a multilayer perceptron (MLP) and compared them in terms of accuracy, precision, recall, and F1-score metrics. The experimental results showed that the CNN and MLP models both performed well for SQL injection attack detection.

In summary, deep learning-based approaches have shown great promise in detecting SQL injection attacks. These approaches can learn the underlying patterns in the input data and detect any anomalies, making them more effective in detecting disguised attacks. In this research, our goal was to explore the effectiveness of the proposed RNN autoencoder in detecting SQL injections.

3. Materials and Methods

The proposed architecture is depicted in Figure 2.

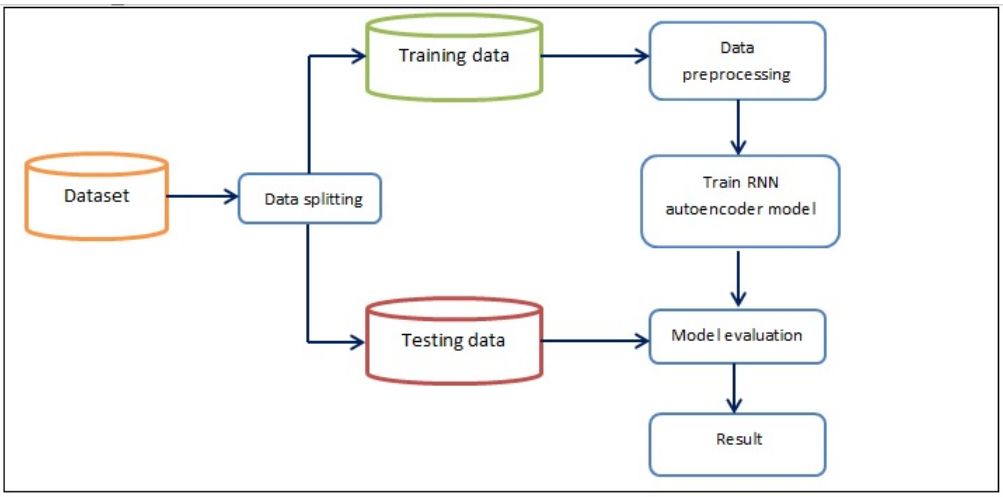

Figure 2. Steps of the proposed architecture.

The architecture consists of the following steps:

- Loading and preprocessing the dataset;
- Splitting the dataset into training and testing sets;
- Building the autoencoder model, which consists of an input layer, an encoder layer, and a decoder layer. The encoder layer reduces the dimensionality of the input data, while the decoder layer reconstructs the original input from the encoded representation;
- Training the autoencoder model on the preprocessed training data;
- Extracting the encoded data from the trained autoencoder model for use in the RNN model;
- Building the RNN model, which consists of an LSTM layer and a dense output layer;
- Training the RNN model on the encoded data;
- Evaluating the model using a set of evaluation techniques.

3.1. Data Preprocessing

The Kaggle dataset [13] was utilized in this research to train, evaluate, and compare the performance of the RNN autoencoder with several classifiers. The dataset was prepared by collecting different SQL injection queries from multiple websites. The dataset contained 30,919 SQL query statements of the form "SELECT FROM" and related variations. Each statement had a binary label, with 1 indicating malicious and 0 benign.

In order to enhance the accuracy of our trained models, we performed data cleaning on the selected dataset. This involved removing any null values and eliminating duplicate records. The removal of missing or null values is crucial, as it prevents the model from learning incorrect relationships or making predictions based on incomplete data. After completing the cleaning process, the dataset consisted of a total of 30,907 records, with 19,529 normal statements and 11,378 malicious statements. The statistics for the dataset are depicted in Figure 3. Each record contained two main features: "Query", which represented the statement itself, and "Label", which indicated whether the statement was normal (0) or malicious (1).

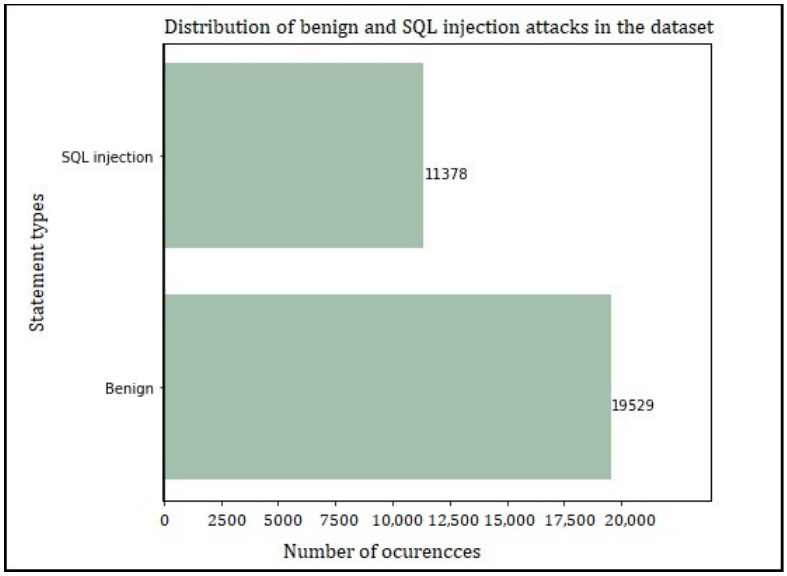

Figure 3. Distribution of benign and SQL injection attacks in the dataset.

Stratified sampling was applied, which ensured that the training and testing sets had similar proportions of each class. This is important for imbalanced datasets like the SQL injection dataset, where the number of malicious queries is much lower than the number of benign queries [14].

3.2. Data Splitting

The dataset was divided into two parts: 80% for training and 20% for testing. This division allowed us to train the proposed approach with the majority of the data and assess its performance with unseen samples.

3.3. Building and Training RNN Autoencoder Model

We developed an architecture for an RNN autoencoder that combines an autoencoder and a recurrent neural network (RNN) for SQL injection attack detection. Figure 4 illustrates the architecture of the proposed model.

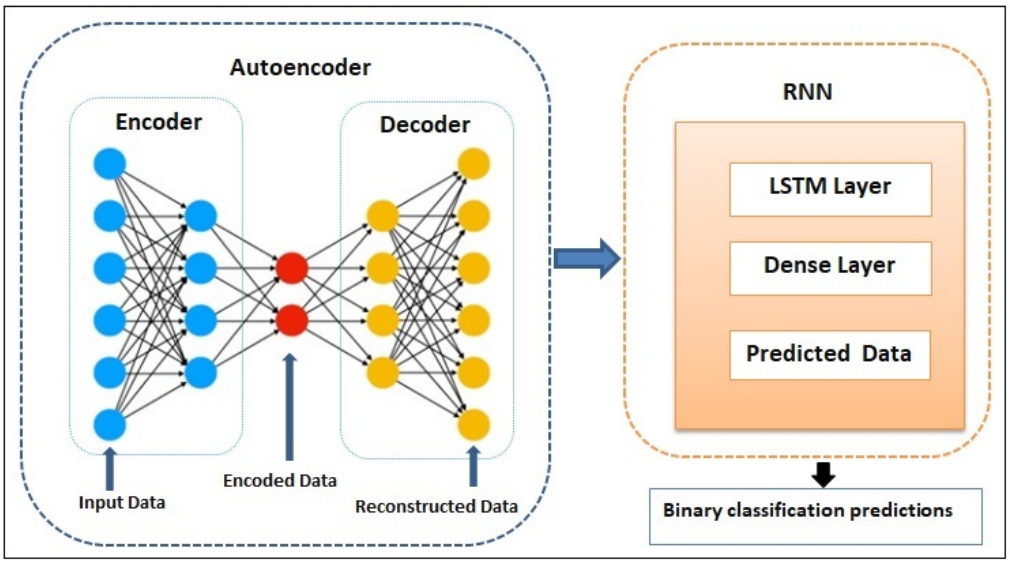

Figure 4. The RNN autoencoder architecture for SQL injection attack detection.

As shown in Figure 4, the proposed architecture consists of two main parts: the autoencoder and the RNN. The autoencoder contains an input layer, an encoder, and a

decoder. The encoder takes the input data and compresses it into a lower-dimensional latent space, which is then fed to the decoder. The decoder then reconstructs the input data from the encoded representation. The size of the latent space can affect the performance of the autoencoder and RNN model, as a smaller latent space may lead to loss of information, while a larger latent space may lead to overfitting. The dimensionality of the latent space in an autoencoder is a crucial hyperparameter that should be carefully tuned [15]. In this research, we experimented with different values for the latent space using a grid search technique to find a value that resulted in a good balance between representation power and computational efficiency. The result of the hyperparameter tuning process showed that 64 was the optimal value for the latent space hyperparameter, which meant that the encoder layer compressed the input data into a 64-dimensional latent space. The RNN was designed to take the compressed representation of the input data learned by the autoencoder and use it to make binary classification predictions [16]. The RNN consisted of an LSTM layer and a dense layer, which takes the encoded data from the autoencoder as input and processes it through an LSTM layer, from where it is then fed to a dense layer to make a prediction with the output.

3.4. Model Evaluation

After training the RNN autoencoder model on the training set, we applied it to the testing set and calculated various performance metrics, such as the ROC curve, accuracy, precision, recall, and F1-score, to measure the effectiveness of the RNN autoencoder in detecting SQLIAs. The mathematical representation of these metrics was as follows.

The accuracy metric measures the percentage of correctly classified samples [17], and it is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{1}$$

Precision, another important metric, represents the probability that a sample will be correctly classified [17]. It is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall, also known as sensitivity or the true-positive rate, indicates the proportion of positive samples that are correctly classified [17]. The recall score is calculated as follows:

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

The F1-score is a combined metric that considers both precision and recall, providing a balanced measure of model performance [18]. It is calculated as follows:

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

TN is the true-negative rate. It indicates the number of correctly predicted normal requests. TP is the true-positive rate. It indicates the number of correctly predicted malicious requests. FN is the false-negative rate. It indicates the number of incorrectly predicted normal requests. FPis the false-positive rate. It indicates the number of incorrectly predicted malicious requests.

4. Results and Discussion

This section provides a description of the experimental results. The Python environment was used to implement the system. Table 1 summarizes the performance of the RNN autoencoder in terms of the evaluation metrics.

Table 1. Performance metrics for the proposed model.

Performance Metrics	Result
Accuracy	94%
Precision	95%
Recall	90%
F1-Score	92%

The results from Table 1 show that the RNN autoencoder performed better in terms of prediction accuracy. The RNN autoencoder achieved an accuracy of 94% and an F1-score of 92%. Further, we used the receiver operating characteristic (ROC) curve to check the performance of the proposed approach. The ROC curve is a graph that shows the relationship between the true-positive rate (TPR) and false-positive rate (FPR) for different classification thresholds [19].

The AUC curve for the RNN autoencoder model is shown in Figure 5. We obtained the value of 0.94, which indicated that our model could successfully separate 94% of positive and negative rates.

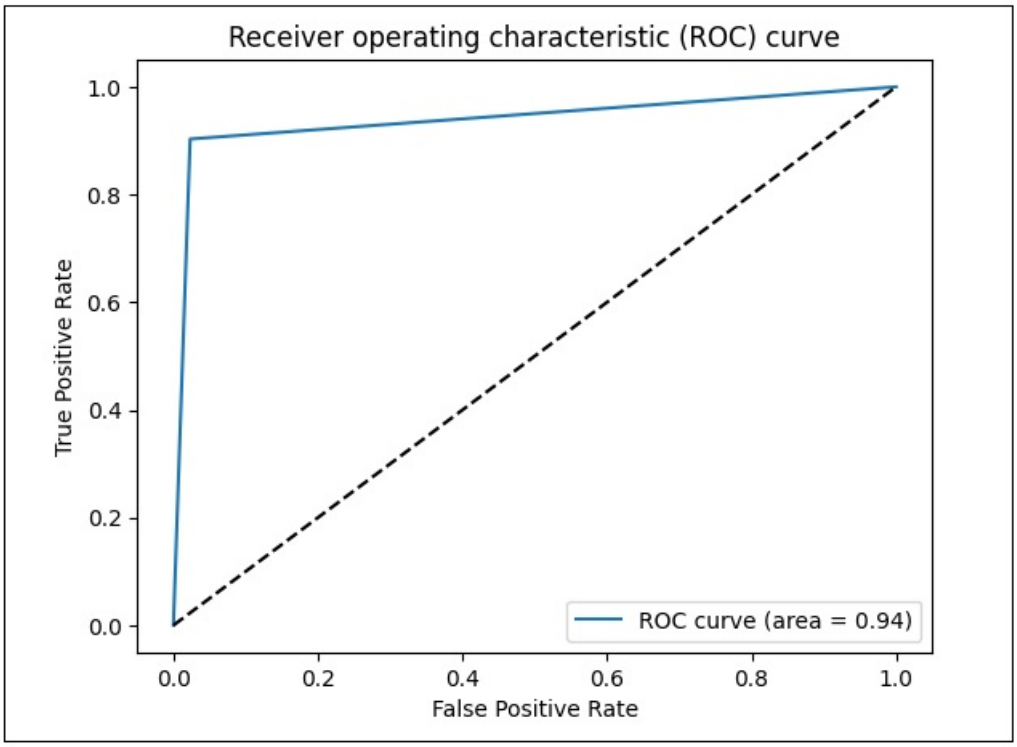

Figure 5. Receiver operating curve (ROC) for our proposed approach.

Regarding RQ1, based on the results provided, it appears that the proposed RNN autoencoder model performed well in correctly identifying instances of SQL injection attacks in the dataset and can be effective for the detection of SQL injection attacks.

Regarding RQ2, one of the most used methods to optimize RNN autoencoders to improve their performance in detecting SQL injection attacks is to adjust the hyperparameters of the model, such as epochs [19]. To find the optimal number of epochs to train the model, we experimented with various numbers of epochs and checked how they affected the accuracy. In the first iteration, we used 10 epochs.

With 10 epochs, we obtained an accuracy of 88%. From Figure 6, we can infer that the validation error decreased. Next, we set the number of epochs to 50.

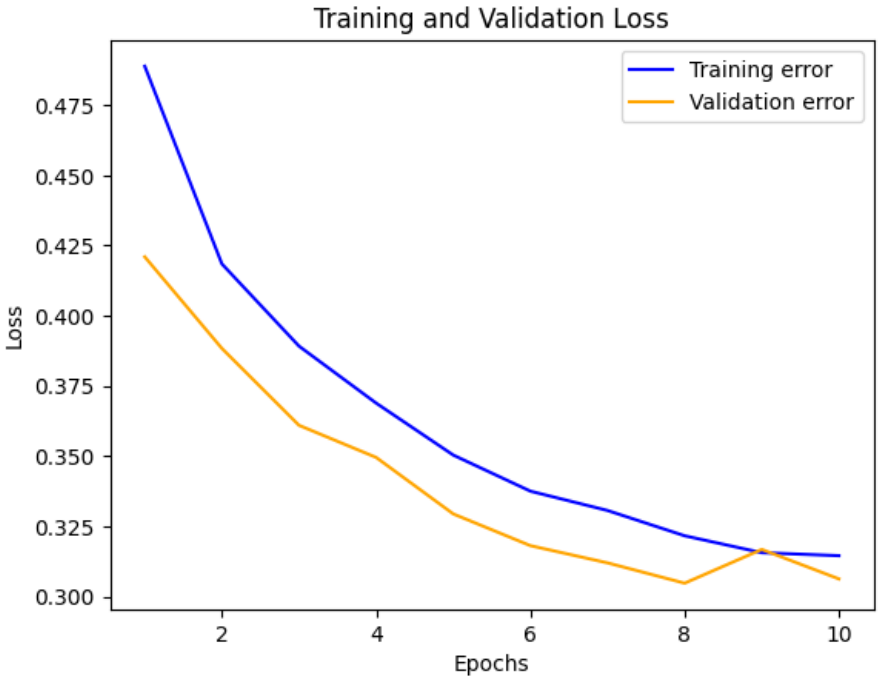

Figure 6. Loss in SQL injection dataset using 10 epochs.

As shown in Figure 7, the accuracy of the model increased to 94% with 50 epochs. Next, we tried to increase the number of epochs to 100.

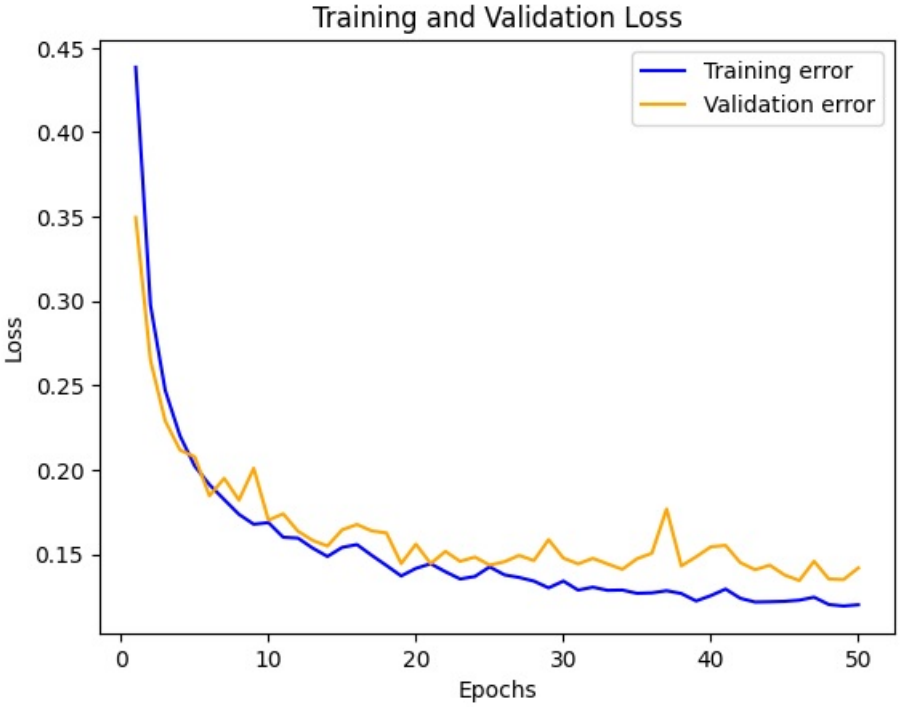

Figure 7. Loss in SQL injection dataset using 50 epochs.

As shown in Figure 8, with 100 epochs, the accuracy increased to 95% but the validation error also increased. This may cause overfitting. Using a small number of epochs, the model cannot capture the underlying patterns in the data, and this may cause underfitting. Furthermore, training the model using many epochs may lead to overfitting, where the model even learns noise or unwanted parts of the data [20]. Therefore, from the this experiment, we deduced that we could stop the training process early at around 50 epochs

to obtain better performance from the model without underfitting or overfitting. Then, a grid search technique was used to find the optimal combination of hyperparameters, such as the activation function. Table 2 summarizes the choices for the different hyperparameters after using the grid search.

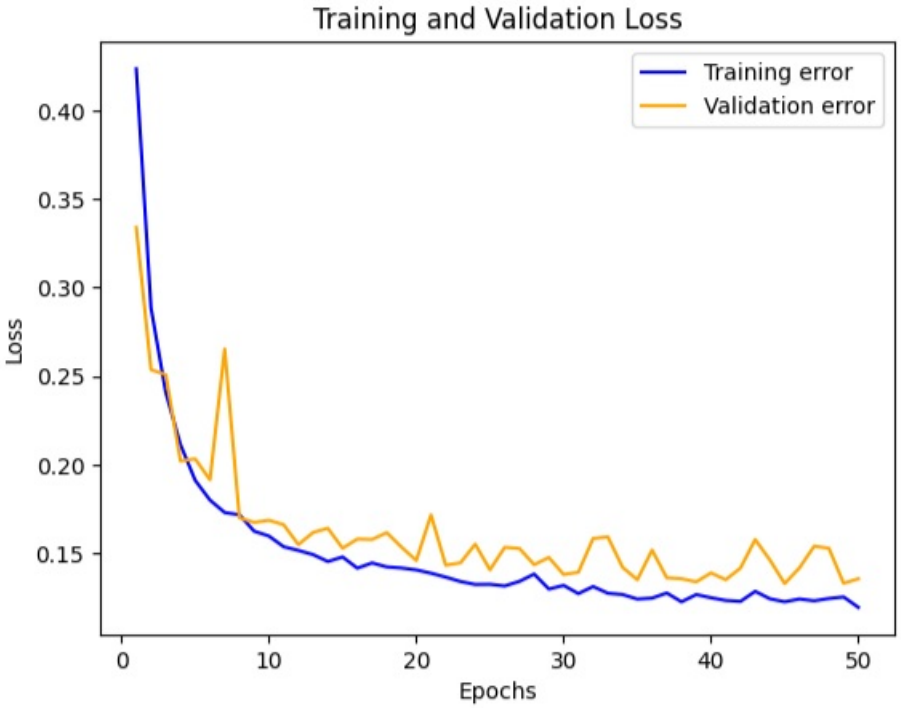

Figure 8. Loss in SQL injection dataset using 100 epochs.

Table 2. Values for several hyperparameters.

Hyperparameters	Value
Number of hidden layers	3
Hidden layer size (neurons)	64 units
Optimizer	Adam
Loss function	Binary cross-entropy
Activation function	ReLU and sigmoid
Number of epochs	50
Batch size	128

The proposed model achieved the best performance when trained for 50 epochs using the Adam optimizer, a batch size of 128, the ReLU activation function for the encoder layer, and the sigmoid activation function for the decoder layer in the autoencoder and output layer in the RNN.

We compared the performance of the proposed approach with the performance of several classifiers, including the ANN, CNN, decision tree, naive Bayes, SVM, random forest, and logistic regression classifiers. The results are presented in Figure 9.

The results in Figure 9 show that the RNN autoencoder and the ANN were effective in detecting SQL injection attacks, achieving a high accuracy of 94% and F1-score of 92%. The RF, LR, and DT models also performed well, achieving accuracy scores of 92%, 93%, and 90%, respectively, and F1-scores of 89%, 90%, and 87%. The CNN model had the highest accuracy of 96% and an F1-score of 49%, indicating its potential for detecting SQL injection attacks. However, the naive Bayes and SVM models had lower accuracy and F1-scores, achieving accuracy scores of 82% and 75%, respectively, and F1-scores of 80% and 49%.

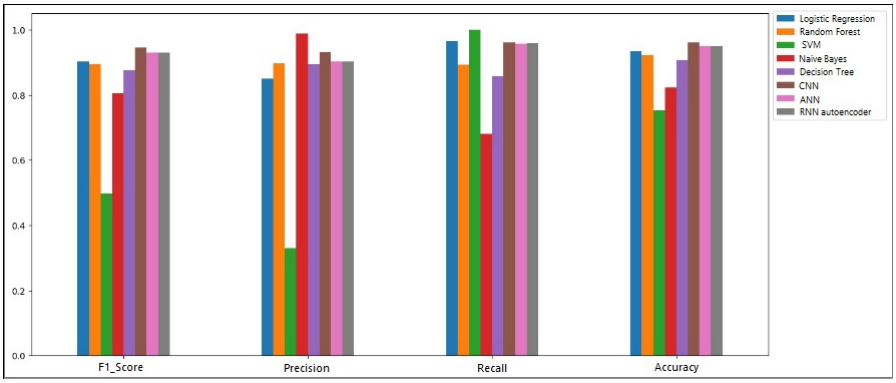

Figure 9. The comparison of evaluation metrics for different ML algorithms.

Regarding RQ3, the results indicated that the RNN autoencoder approach outperformed some of the other algorithms, including the logistic regression, decision tree, random forest, SVM, and naive Bayes algorithms, in terms of accuracy, precision, recall, and F1-score. The RNN autoencoder approach also performed comparably to some of the other algorithms, including the CNN and ANN models, in many NLP tasks, but each architecture has its strengths and weaknesses. According to a study by Yin et al. [21], CNNs perform better at tasks that require local feature extraction, such as sentiment analysis, while RNNs perform better at tasks that require an understanding of longer-term dependencies, such as question answering. They found that both CNNs and RNNs are sensitive when hyperparameter values are varied depending on the task. Banerjee et al. [22] developed CNN and RNN models with similar architectures for classifying radiology reports and found that RNNs were the more powerful model to encode sequential information. However, the study noted that CNNs required less hyperparameter tuning to prevent overfitting and were more stable, while RNNs needed more careful regularization.

In this research, since the SQL queries could contain longer-term dependencies, it made sense that the RNN autoencoder model achieved comparable accuracy to the CNN model. The added memory and sequencing modeling of the RNN likely helped it perform well with the longer query texts, but it may require additional tuning to match the performance of CNNs in some cases. This may explain why the CNN model slightly outperformed the RNN model.

In summary, our results are consistent with previous findings that indicate that RNNs are well suited for longer textual sequences but may require additional tuning to maximize performance compared to CNN models. The strong accuracy of 94% demonstrates the promise of the RNN autoencoder architecture for detecting SQL injection attacks. The key advantage of the RNN autoencoder is that it can learn a compressed representation of the input data, allowing it to capture the underlying patterns and relationships in the data more effectively than traditional methods.

5. Conclusions and Outlooks

A deep learning architecture model based on an RNN autoencoder was proposed for detecting SQL injection attacks. The autoencoder was trained to learn a compressed representation of the input data, while the RNN used this compressed representation to make binary classification predictions. In this study, the RNN autoencoder was trained with different optimization techniques on a public SQL injection dataset. The performance of the model was evaluated using standard evaluation metrics, such as accuracy, precision, recall, and F1-score. Additionally, an ROC curve was calculated to evaluate the model's performance. The experimental results showed that the proposed approach achieved an accuracy of 94% and an F1-score of 92%, indicating that the RNN autoencoder is a promising method for detecting SQL injection attacks. As part of future research, we plan to explore the use of a more complex architecture for the RNN autoencoder to detect SQL injection attacks. Additionally, we acknowledge that the dataset used in this study was

relatively small, and we recommend expanding the dataset and implementing the models in real-world scenarios in future investigations.

Author Contributions: Conceptualization, M.A. and D.A.; methodology, M.A.; software, M.A.; validation, M.A., D.A. and S.A.; investigation, M.A.; resources, M.A.; data curation, M.A.; writing—original draft preparation, M.A.; writing—review and editing, S.A.; visualization, M.A.; supervision, D.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. IFPDP-284-22. The authors, therefore, acknowledge with thanks the DSR's technical and financial support.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SQL	structured query language
SQLIA	SQL injection attack
RNN-ED	RNN-based encoder–decoder
IDSs	intrusion detection systems
ML	machine learning
DL	deep learning
NB	naive Bayes classifier
DT	decision tree
LR	logistic regression
RF	random forest
SVM	support vector machine
CNN	convolutional neural network
ANN	artificial neural network
MLP	multilayer perceptron
RNN	recurrent neural network
LSTM	long short-term memory

References

1. Ketema, A. Developing Sql Injection Prevention Model Using Deep Learning Technique. Ph.D. Thesis, St. Mary's University, London, UK, 2022.
2. H.R., Y.W.; Kottegoda, H.; Andaraweera, D.; Palihena, P. A Comprehensive Review of Methods for SQL Injection Attack Detection and Prevention. 2022. Available online: https://www.researchgate.net/publication/364935556_A_comprehensive_review_of_methods_for_SQL_injection_attack_detection_and_prevention (accessed on 27 April 2023).
3. Chen, D.; Yan, Q.; Wu, C.; Zhao, J. SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *J. Phys. Conf. Ser.* **2021**, *1757*, 012055. [CrossRef]
4. Zhang, W.; Li, Y.; Li, X.; Shao, M.; Mi, Y.; Zhang, H.; Zhi, G. Deep Neural Network-Based SQL Injection Detection Method. *Secur. Commun. Netw.* **2022**, *2022*, 4836289. [CrossRef]
5. Mishra, A.A.; Edelen, A.; Hanuka, A.; Mayes, C. Uncertainty quantification for deep learning in particle accelerator applications. *Phys. Rev. Accel. Beams* **2021**, *24*, 114601. [CrossRef]
6. Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; Mané, D. Concrete problems in AI safety. *arXiv* **2016**, arXiv:1606.06565.
7. Lakshminarayanan, B.; Pritzel, A.; Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. *Adv.Neural Inf. Process. Syst.* **2017**, *30*, 6405–6416.
8. Yu, W.; Kim, I.Y.; Mechefske, C. Analysis of different RNN autoencoder variants for time series classification and machine prognostics. *Mech. Syst. Signal Process.* **2021**, *149*, 107322. [CrossRef]
9. Roy, P.; Kumar, R.; Rani, P. SQL Injection Attack Detection by Machine Learning Classifier. In Proceedings of the 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 9–11 May 2022; pp. 394–400.

10. Krishnan, S.A.; Sabu, A.N.; Sajan, P.P.; Sreedeep, A. SQL Injection Detection Using Machine Learning. *Rev. Geintec-Gest. Inov. Tecnol.* **2021**, *11*, 300–310.
11. Rahul, S.; Vajrala, C.; Thangaraju, B. A Novel Method of Honeypot Inclusive WAF to Protect from SQL Injection and XSS. In Proceedings of the 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON), Bengaluru, India, 19–21 November 2021; Volume 1, pp. 135–140.
12. Liu, M.; Li, K.; Chen, T. DeepSQLi: Deep semantic learning for testing SQL injection. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual, 18–22 July 2020; pp. 286–297.
13. Sajid576. SQL Injection Dataset. 2021. Available online: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset (accessed on 27 April 2023).
14. Chindove, H.; Brown, D. Adaptive Machine Learning Based Network Intrusion Detection. In Proceedings of the International Conference on Artificial Intelligence and its Applications, Virtual, 9–10 December 2021; pp. 1–6.
15. Gillette, A.; Chang, T. *ALGORITHMS: Assessing Latent Space Dimension by Delaunay Loss*; Technical Report; Lawrence Livermore National Lab. (LLNL): Livermore, CA, USA, 2020.
16. Do, J.S.; Kareem, A.B.; Hur, J.W. LSTM-Autoencoder for Vibration Anomaly Detection in Vertical Carousel Storage and Retrieval System (VCSRS). *Sensors* **2023**, *23*, 1009. [CrossRef] [PubMed]
17. Mwaruwa, M.C. Long Short Term Memory Based Detection Of Web Based Sql Injection Attacks. Ph.D. Thesis, University of Newcastle, Callaghan, Australia, 2019.
18. Ahmad, M.S.; Shah, S.M. Supervised machine learning approaches for attack detection in the IoT network. In *Internet of Things and Its Applications*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 247–260.
19. Said Elsayed, M.; Le-Khac, N.A.; Dev, S.; Jurcut, A.D. Network anomaly detection using LSTM based autoencoder. In Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Alicante, Spain, 16–20 November 2020; pp. 37–45.
20. Afaq, S.; Rao, S. Significance of epochs on training a neural network. *Int. J. Sci. Technol. Res* **2020**, *9*, 485–488.
21. Yin, W.; Kann, K.; Yu, M.; Schütze, H. Comparative study of CNN and RNN for natural language processing. *arXiv* **2017**, arXiv:1702.01923 .
22. Banerjee, I.; Ling, Y.; Chen, M.C.; Hasan, S.A.; Langlotz, C.P.; Moradzadeh, N.; Chapman, B.; Amrhein, T.; Mong, D.; Rubin, D.L.; et al. Comparative effectiveness of convolutional neural network (CNN) and recurrent neural network (RNN) architectures for radiology text report classification. *Artif. Intell. Med.* **2019**, *97*, 79–88. [CrossRef] [PubMed]