Type of Contribution:
Research Paper
Review Paper
Case Study

ENERGY: JURNAL ILMIAH ILMU-ILMU TEKNIK
Vol. 15, No. 2 (2025) pp 121-134
DOI: 10.51747/energy.v15i2.15201

E-ISSN: 2962-2565

Optimization of the Recurrent Neural Network (RNN) Model for SQL Injection Intrusion Detection In Databases

Turmuzi[1]*, **Kusrini**[1]

[1] Master Informatics Engineering, Amikom Yogyakarta University, 55283, **Indonesia**

*turmuzi@students.amikom.ac.id

Abstract

This study focuses on optimizing the Recurrent Neural Network (RNN) model, specifically through the integration of Long Short-Term Memory (LSTM) layers, to improve the detection of SQL injection attacks in database systems. SQL injection is a significant cybersecurity threat that can compromise sensitive data and organizational integrity. Using a dataset from Kaggle, the research explores various hyperparameter configurations, such as activation functions (Softmax, Tanh, ReLU, PReLU, and LReLU), optimizers (Adam, SGD), learning rates (0.001, 0.0001), and regularization techniques (Dropout and L2). The study reports a notable performance improvement with the Tanh activation function, Adam optimizer, and a learning rate of 0.001, achieving an accuracy of 98.17% and a loss of 0.4084 after 100 epochs. Compared to other models, such as those using standard RNNs or hybrid BERT-LSTM models, the proposed RNN-LSTM model outperforms in detection accuracy while maintaining lower computational requirements, making it more suitable for real-time applications. This research contributes to the field of database security by demonstrating the efficacy of optimized RNN-LSTM models for real-time SQL injection detection and highlights the importance of hyperparameter tuning for improved performance. The findings pave the way for future advancements in intelligent threat detection systems.

Keywords: Optimization; LSTM; RNN; Intrusion Detection; SQL Injection

Article Info
Submitted: 2025-07-02
Revised: 2025-08-04
Accepted: 2025-08-18
Published: 2025-8-22

Publisher
Universitas Panca Marga

1. Introduction

In In today's rapidly evolving digital landscape, the safeguarding of sensitive data is paramount. SQL injection attacks, which exploit vulnerabilities in database-driven applications, remain one of the most dangerous cybersecurity threats [1].

These attacks can manipulate SQL queries to bypass authentication, access sensitive information, and damage organizational systems. As reported by the Open Web Application Security Project (OWASP), SQL injection is consistently ranked as one of the top vulnerabilities in web applications, emphasizing the critical need for robust intrusion detection mechanisms. Despite significant advancements in cybersecurity, SQL injection attacks continue to be a prevalent method for cybercriminals to exploit web applications, posing a constant challenge to database security.

Machine learning techniques have shown promise in improving intrusion detection, and among these, Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, have gained attention for their ability to process sequential data and identify temporal patterns [2]. Previous studies have demonstrated the success of RNN-based models in detecting SQL injection attacks [3]. For example, AlAzzawi (2023) reported achieving high accuracy with a standard RNN model in identifying SQL injection attempts [4]. While Liu and Dai (2024) implemented a hybrid BERT-LSTM model for intrusion detection with notable precision [5]. These efforts illustrate the growing potential of deep learning models in cybersecurity but also highlight ongoing challenges [6], particularly in terms of optimizing model performance and achieving real-time detection capabilities.

While prior research has contributed significantly to the field, gaps remain in optimizing RNN-LSTM models for SQL injection detection. Existing studies often focus on achieving high accuracy but overlook issues related to computational efficiency, model overfitting, and the ability to adapt to evolving attack strategies. Moreover, many models fail to address the imbalanced nature of SQL injection datasets [7], where benign queries outnumber malicious ones, leading to inaccurate evaluation results. This study aims to fill these gaps by focusing on the optimization of RNN-LSTM models to improve both detection accuracy and computational efficiency, with a particular emphasis on reducing false positives and enhancing real-time detection capabilities.

The primary objective of this research is to develop an optimized RNN-LSTM model that offers a balance between high accuracy and computational efficiency. This study investigates various hyperparameter configurations, including activation functions (Tanh, ReLU, LReLU), optimizers (Adam, SGD), learning rates, and regularization methods, to enhance the model's performance [8]. By addressing the challenges of class imbalance, overfitting, and model adaptability, this research seeks to contribute to the development of more reliable and scalable intrusion detection systems. Additionally, the study aims to provide insights into the novel application of RNN-LSTM models in the context of SQL injection detection and lay the groundwork for future advancements in intelligent threat detection models.

2. Methods

2.1 Stages of Research

This study employs a quantitative experimental research design to evaluate the performance of an optimized Recurrent Neural Network (RNN) model with Long Short-Term Memory (LSTM) layers in detecting SQL injection attacks [9]. The experiments aim to identify the most effective hyperparameter configurations, while also addressing key challenges such as class imbalance and model overfitting.

2.2 System Design

The dataset used for this research was sourced from Kaggle and contains over 30,000 SQL queries [10]. labeled as either benign (0) or malicious (1). Specifically, the dataset includes 11,382 malicious queries and 19,537 benign queries, resulting in a 36.8% to 63.2% distribution of malicious and benign queries, respectively. This class imbalance presents a challenge for model training and evaluation, as models trained on imbalanced datasets may exhibit bias toward the majority class (benign queries). To address this issue, a series of preprocessing techniques were employed to balance the dataset and improve model performance. These included:

- **Resampling Methods**: The dataset was balanced using oversampling techniques such as the Synthetic Minority Over-sampling Technique (SMOTE), which generated synthetic examples of the minority class (malicious queries). This ensures that the model is exposed to a more balanced distribution of classes during training, which helps mitigate bias.
- **Evaluation Metrics**: In addition to accuracy, which may be misleading in the case of imbalanced data, additional evaluation metrics such as precision, recall, and F1-score were used to assess model performance. These metrics provide a more reliable measure of model effectiveness, particularly for detecting malicious queries.

2.3 Data Preprocessing:

Several preprocessing steps were applied to the dataset before training the model:

- **Tokenization**: SQL queries were tokenized into individual components to enable the model to process the data as sequences.
- **Numerical Encoding**: The tokenized queries were numerically encoded using one-hot encoding and word embeddings to represent the categorical tokens as vectors.
- **Sequence Padding**: To ensure that all input sequences had the same length, padding was applied to standardize the sequence lengths across the dataset. This is necessary for the model to process the data efficiently.

- **Normalization**: The numerical features were normalized to ensure consistent scaling and to prevent any single feature from disproportionately influencing the model's performance.

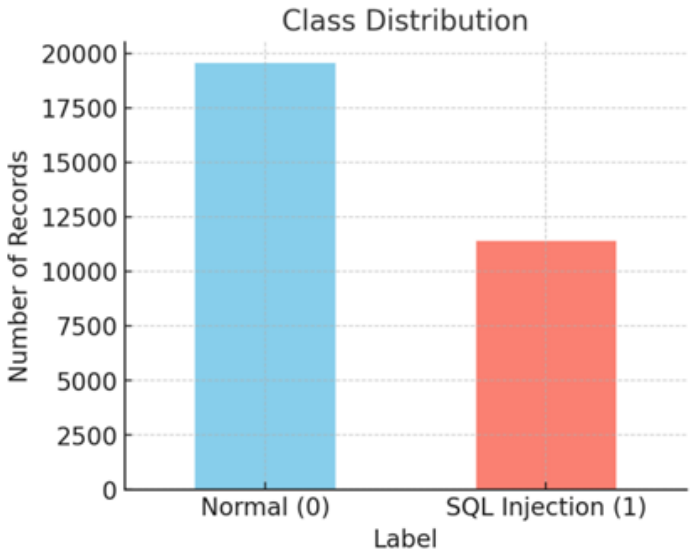

Figure 1. SQL Injection Dataset Labeling

Tabel 1. Data Sample of Dataset SQL Injection

Query	Label
" or pg_sleep(__TIME__) --	1
create user name identified by pass123 temporary tablespace temp	1
AND 1 = utl_inaddr.get_host_address(...)	1
select * from users where id = '1' or @@1=1 --	1
select * from users where id = 1 or 1#" (...)	1

Preprocessing steps included tokenization of SQL queries, numerical encoding using one-hot encoding or embeddings, and sequence padding to ensure uniform input lengths. The dataset was also normalized, and label encoding was applied for classification as shown in **Tabel 1**.

2.4 Stages of Research

The research process involves a series of steps necessary to carry out the experiment. The following is a research flowchart and an explanation of each step:

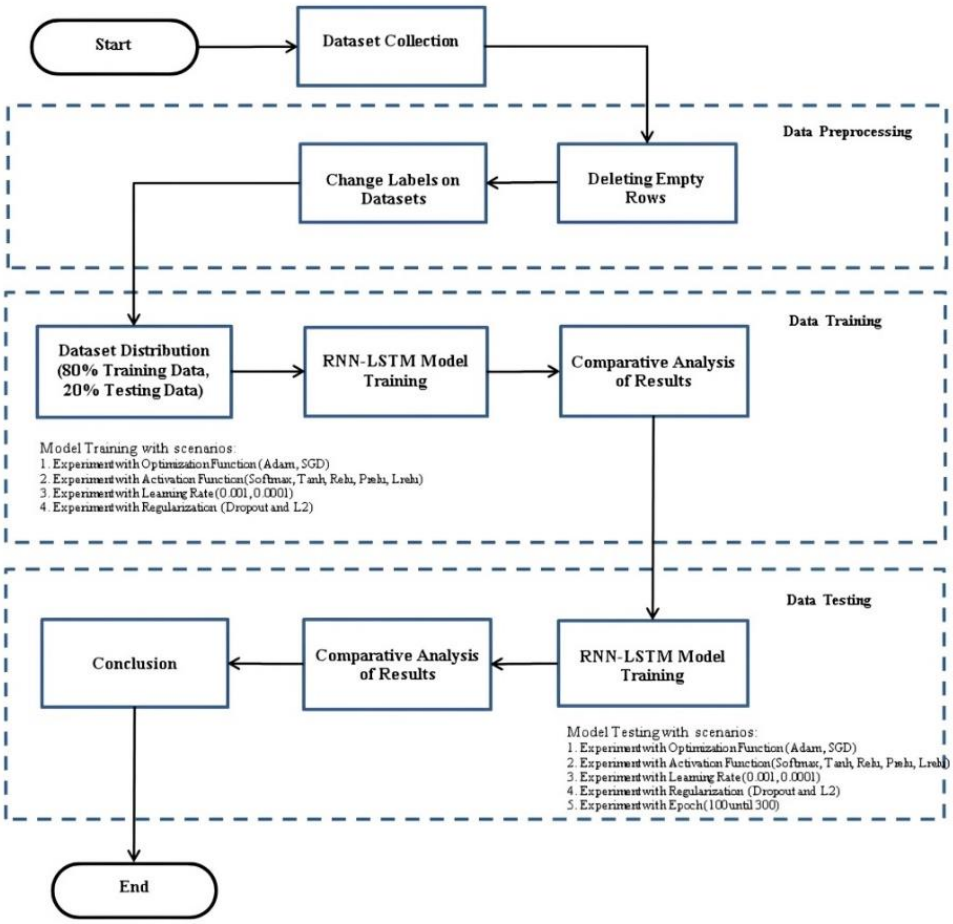

Figure 2. Research Flow Chart

The research flow consists of several key stages, as outlined below:

1. **Data Splitting**: The dataset was divided into training and testing subsets, with 80% of the data allocated for training and 20% for testing. This division allows for the evaluation of model performance on unseen data, providing an accurate assessment of generalization capability.
2. **Model Training**: The RNN model was trained using various configurations of hyperparameters, including activation functions, optimizers, learning rates, and the number of epochs. The following hyperparameters were systematically tested:
   - Activation Functions: Tanh, ReLU, LReLU, PReLU, and Softmax were tested. Tanh was selected due to its ability to avoid saturation and perform well with sequential data, which is crucial for SQL injection detection.
   - Optimizers: The Adam optimizer and Stochastic Gradient Descent (SGD) were evaluated. Adam was chosen for its adaptive learning rate capabilities, which can accelerate convergence, especially for complex datasets like those involved in intrusion detection tasks.

- Learning Rates: Two learning rates, 0.001 and 0.0001, were tested. A learning rate of 0.001 was preferred based on prior studies [4][5] indicating its effectiveness in balancing training speed and model stability.
- Regularization: A combination of Dropout (set at 0.5) and L2 regularization was used to prevent overfitting, especially with the imbalance in the dataset and the high risk of overfitting given the complexity of the model.
- Epochs: The number of epochs was varied between 50, 70, and 100 to assess the effect of longer training on model performance and to avoid overfitting by monitoring performance on the validation set.

3. **Model Evaluation**: After training, the model's performance was evaluated using the test set. As mentioned, multiple evaluation metrics (accuracy, precision, recall, F1-score) were used to measure the model's performance comprehensively. The accuracy score was complemented by the confusion matrix, which provides insights into the true positives, false positives, true negatives, and false negatives, offering a better understanding of the model's ability to detect malicious queries.

2.5 Hyperparameter Selection Justification:

The choice of hyperparameters was informed by the characteristics of the dataset and prior research in the field of intrusion detection. The selection of the LSTM architecture was based on its proven effectiveness in handling sequential data, making it ideal for capturing the temporal dependencies inherent in SQL queries. Additionally, Tanh was chosen as the activation function due to its non-saturating nature, which allows for more stable gradients and better performance with the dataset. The Adam optimizer was selected for its ability to adaptively adjust learning rates, ensuring faster convergence and reducing the risk of overshooting the optimal solution.

2.6 Class Imbalance Handling in Model Training:

To address the issue of class imbalance, in addition to oversampling, a weighted loss function was used during model training. This approach assigns higher penalties for misclassifying malicious queries, compensating for the imbalance by making the model more sensitive to the minority class. This step ensures that the model doesn't favor the majority class (benign queries) and improves its ability to correctly classify malicious SQL injection attempts.

3.Results and Discussion

The experimental results confirm the efficacy of the optimized Recurrent Neural Network (RNN) model with Long Short-Term Memory (LSTM) layers in

detecting SQL injection attacks. As presented in Table 2, the performance of the model was evaluated across various hyperparameter configurations, including activation functions, optimizers, learning rates, and the number of epochs. The key findings highlight the superior performance of the Tanh activation function in combination with the Adam optimizer, resulting in the highest accuracy (98.17%) and the lowest loss (0.4084) after 100 epochs. This configuration outperformed other activation functions and optimizers, providing a compelling justification for its use in the final model.

3.1 Key Findings

a. Activation Function and Optimizer:

The comparison of different activation functions and optimizers demonstrates that Tanh coupled with Adam optimizer consistently delivers the best results in terms of both accuracy and loss reduction. Non-saturating activation functions, like Tanh and ReLU, were particularly effective, as they avoided the gradient saturation problem that can impede training, especially with deep neural networks. The Adam optimizer was superior to SGD, likely due to its adaptive learning rate, which allowed for faster convergence and better overall performance. This combination of Tanh and Adam provides a stable, efficient, and accurate model for SQL injection detection.

b. Impact of Learning Rate:

The learning rate played a critical role in determining the model's convergence speed and overall performance. A learning rate of 0.001 yielded the best results, while smaller learning rates such as 0.00001 resulted in slower convergence, which ultimately impacted the model's ability to achieve high accuracy. Learning rates that are too low can cause the model to get stuck in suboptimal local minima, thereby reducing its ability to improve during training.

c. Effect of Epochs:

The number of epochs was also found to influence model performance. While increasing the epochs from 50 to 100 resulted in improved accuracy, the performance gain became marginal after reaching 100 epochs. This indicates that the model began to converge after a certain number of epochs, and further training did not significantly improve results. In contrast, 50 epochs may not provide sufficient training for the model to reach its optimal accuracy, especially in the case of complex datasets with high variance.

Tabel 2. Accuracy Comparison based on Activation Function and Optimizer

Index	Activation	Optimizer	Learning Rate	Epochs	Accuracy	Loss
0	**LReLU**	**Adam**	**0.001**	**50**	**0.9655**	**0.3594**
1	Tanh	SGD	0.001	50	0.9578	3.0912
2	LReLU	SGD	0.001	50	0.9529	3.1287
3	LReLU	Adam	0.00001	50	0.9241	0.883
4	Tanh	Adam	0.00001	50	0.9195	0.9188
5	PReLU	Adam	0.00001	50	0.9190	0.9921
6	**Tanh**	**Adam**	**0.001**	**70**	**0.9565**	**0.5838**
7	PReLU	Adam	0.001	70	0.9541	0.4586
8	ReLU	SGD	0.001	70	0.9537	3.4182
9	Tanh	Adam	0.00001	70	0.9402	0.6298
10	PReLU	Adam	0.00001	70	0.9353	0.6744
11	ReLU	Adam	0.00001	70	0.9351	0.6220
12	**Tanh**	**Adam**	**0.001**	**100**	**0.9817**	**0.4084**
13	Tanh	SGD	0.001	100	0.9636	1.7652
14	LReLU	SGD	0.001	100	0.9550	3.9411
15	Tanh	Adam	0.0001	100	0.9476	0.5514
16	PReLU	Adam	0.0001	100	0.9460	0.4692
17	LReLU	Adam	0.0001	100	0.9457	0.4732

This study conducted a series of experiments to evaluate the performance of the Recurrent Neural Network (RNN) model with LSTM layers in detecting SQL injection attacks. The parameters tested included activation functions, optimizer types, learning rate values, and the number of epochs. Regularization was performed using a combination of the Dropout and L2 methods. The following is a summary of the results of several parameter combinations that have been tested: For systems where computational efficiency is critical as summarized in **Tabel 2**, using LReLU with Adam optimizer and 50 epochs is sufficient to achieve high accuracy. However, for environments prioritizing maximum detection capability, Tanh with Adam optimizer and 100 epochs provides the highest accuracy, albeit with increased training time and other functions were tested but not all of them were used in the final model.

3.2 Discussion of Overfitting and Model Limitations

Despite the promising results, the study also identified the risk of overfitting in certain configurations. Overfitting occurs when a model learns to perform well on the training data but fails to generalize effectively to unseen data. This issue was particularly evident in configurations with a high number of epochs and certain optimizer settings. Although Dropout and L2 regularization were applied to mitigate overfitting, there remains a potential concern, particularly with small or imbalanced datasets. Regularization techniques like SMOTE for balancing the

dataset and early stopping could further help to address overfitting and improve the model's generalizability.

Furthermore, zero-day attacks and highly obfuscated SQL injections remain challenges that are not fully addressed by this model. While the optimized RNN-LSTM model demonstrated high accuracy in detecting known SQL injection patterns, new attack variants that have not been seen during training could still evade detection. In future work, the integration of autoencoders for anomaly detection or the use of unsupervised learning methods could help the model adapt to previously unseen attacks. Additionally, adversarial training could improve the model's robustness against more sophisticated or obfuscated SQL injection techniques.

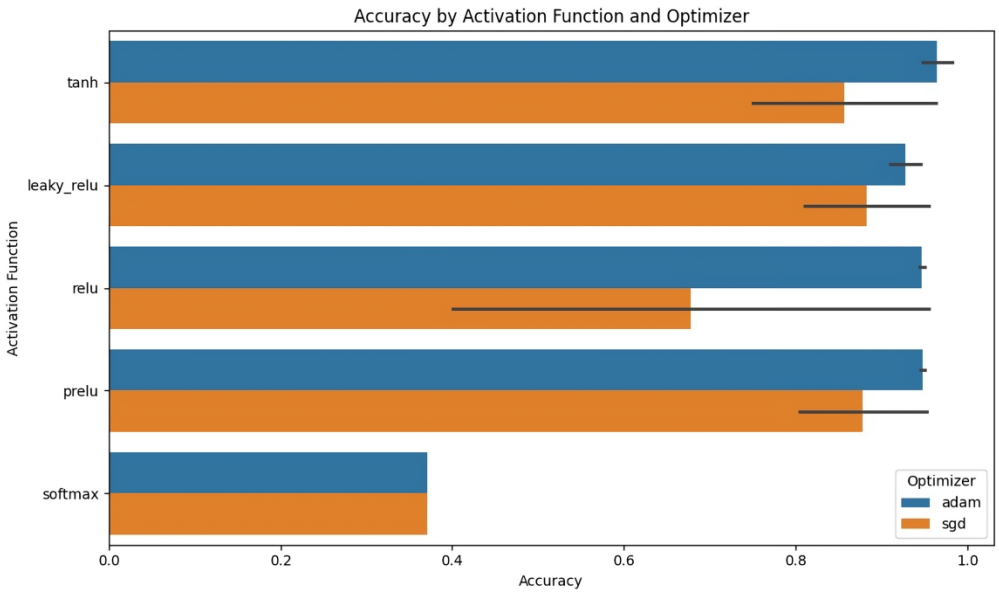

Figure 3. Accuracy Comparison based on Activation Function and Optimizer

Figure 3 shows that Tanh with Adam achieves the highest accuracy, indicating the advantage of adaptive optimizers and non-saturating activation functions. From the experimental results, it can be seen that the Tanh, ReLU, and PreLU activation functions produce higher accuracy compared to other activation functions. This shows that non-linear activation that avoids saturation, such as sigmoid/tanh, provides better performance. Adam shows more stable and consistent performance with higher accuracy than SGD in most scenarios. This is due to Adam's ability to adaptively adjust the learning rate during the training process.

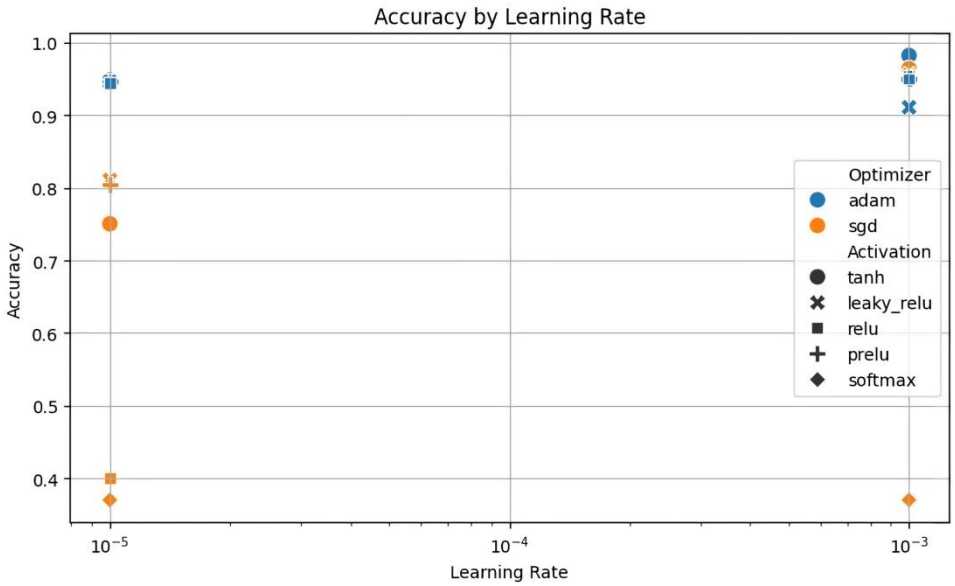

Figure 4. Accuracy Comparison with Learning Rate

3.2.1 Real-World Applicability

The model's performance in a controlled experimental environment is promising, but its real-world applicability in dynamic, large-scale systems needs further exploration. Real-world environments introduce variability, such as new attack vectors, evolving database systems, and shifting patterns in user queries. To improve its deployment in practical settings, future adaptations of this model should focus on scalability, real-time processing capabilities, and the ability to integrate with existing web application firewalls (WAFs) or other security tools. The ability to detect zero-day attacks and adapt to changing attack patterns is crucial for long-term effectiveness in production systems.

As shown in **Figure 4** learning rate of 0.001 has been proven to produce higher accuracy compared to very small learning rates such as 0.00001. Learning rates that are too small tend to cause models to converge slowly and stagnate.

Increasing the number of epochs from 50 to 100 generally improves accuracy, but only up to a certain point. After reaching a certain number of epochs, the improvement in performance is not very significant, as seen in **Figure 5**. In conclusion, the optimized RNN-LSTM model offers a highly accurate and efficient approach to detecting SQL injection attacks, providing a solid foundation for real-time intrusion detection systems. By addressing the limitations and extending the model to handle more complex scenarios, this research paves the way for more effective cybersecurity solutions.

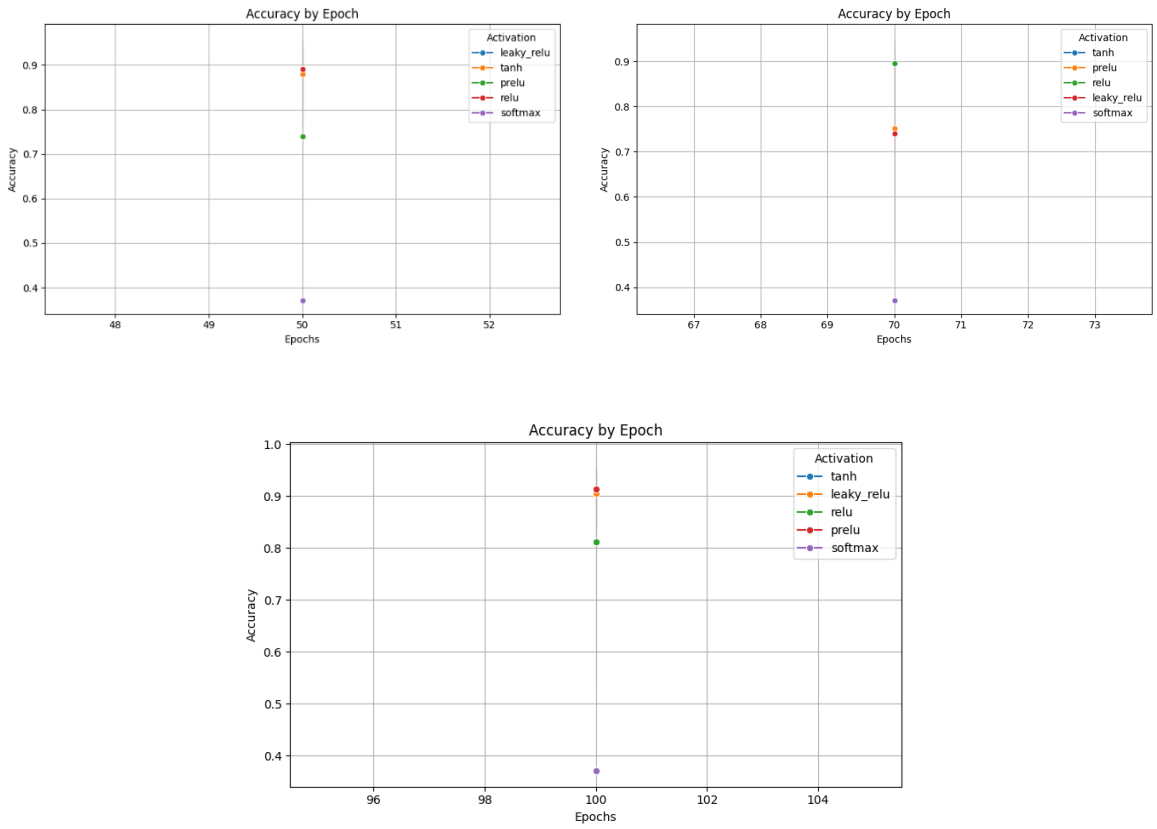

Figure 5. Accuracy Comparison with Epochs

3.2.2 Future Research Directions

The optimization of recurrent neural networks (RNNs) for SQL injection intrusion detection presents significant opportunities for enhancing the robustness of database security systems. The current landscape of intrusion detection systems (IDSs) emphasizes the necessity for models that can effectively categorize SQL queries as either benign or malicious, particularly in light of evolving attack strategies such as zero-day and obfuscated SQL injection attacks.

First, RNNs have been instrumental in processing sequential data due to their ability to retain contextual information, which is vital when analyzing SQL syntax and semantics. Some research proposes a novel method using RNNs that captures the syntax and semantics of SQL queries, demonstrating the effectiveness of RNNs in distinguishing between benign and malicious queries based on training data composed of both types of queries [11]. However, as SQL injection techniques grow increasingly sophisticated, integrating more advanced deep learning architectures, such as Transformer-based models, may vastly improve detection capabilities. [12] illustrate how a hybrid model combining Bidirectional Encoder Representations from Transformers (BERT) and Long Short-Term Memory (LSTM) networks enhances detection by effectively analyzing complex SQL query patterns [12].

Moreover, reinforcement learning (RL) approaches offer promising avenues for real-time model adjustments and enhancements. López-Martín et al. (2020)

discuss how RL can dynamically adapt models for intrusion detection, enabling RNN models for SQL injection detection to learn from new attack data continuously without extensive retraining [13]. Such adaptations are integral in environments characterized by high traffic, where conventional static models may falter under the weight of evolving attack patterns [14].

Furthermore, the scalability of detection models in high-traffic scenarios is crucial. High-performance detection systems capable of processing large datasets efficiently will ensure that the integrity of database systems is maintained even as transaction volumes escalate. Future research should prioritize the development of hybrid models that incorporate both RNNs for sequence modeling and RL for proactive adaptations, possibly leveraging architectures that extend beyond traditional RNNs, including autoencoders. Other research proposes using RNN autoencoders for detecting SQL injection attacks, indicating the potential for hybrid approaches to enhance detection capabilities [15].

In summary, future directions in optimizing RNN models for SQL injection detection should encompass the exploration of advanced architectures, the integration of RL techniques for dynamic resilience against new threats, and a focus on scalability for high-traffic environments. These measures will enhance the effectiveness of intrusion detection systems in safeguarding database integrity against emerging cyber threats.

4. Conclusion

This study demonstrates the effectiveness of an optimized Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) layers for detecting SQL injection attacks, achieving 98.17% accuracy and a loss of 0.4084 after 100 epochs. By fine-tuning hyperparameters such as the Tanh activation function, Adam optimizer, and a learning rate of 0.001, this model outperforms previous RNN models, like AlAzzawi's 2023 approach (94.5% accuracy). The results highlight the importance of adaptive optimizers, non-saturating activation functions, and addressing class imbalance. These findings are valuable for real-time intrusion detection systems, especially in database security, and suggest the model's suitability for web application firewalls and security monitoring. Future research could explore advanced architectures like Transformers, improve adversarial attack resilience, and focus on scalability and real-time processing for high-traffic environments, further advancing SQL injection detection and database application security.

Authors' Declaration

Authors' contributions and responsibilities - The authors made substantial contributions to the conception and design of the study. The authors took

responsibility for data analysis, interpretation, and discussion of results. The authors read and approved the final manuscript.

Funding - No funding information from the authors.

Availability of data and materials - All data is available from the authors.

Competing interests - The authors declare no competing interest.

Additional information - No additional information from the authors.

References

[1] "OWASP Top Ten 2021: A03-Injection," OWASP Foundation. [Online]. Available: https://owasp.org/Top10/A03_2021-Injection/

[2] A. G. Kakisim, "A deep learning approach based on multi-view consensus for SQL injection detection," *Int. J. Inf. Secur.*, vol. 23, no. 2, pp. 1541–1556, Apr. 2024, doi: 10.1007/s10207-023-00791-y.

[3] J. Zheng, J. Li, C. Li, and R. Li, "A SQL Blind Injection Method Based on Gated Recurrent Neural Network," in *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*, Guilin, China: IEEE, July 2022, pp. 519–525. doi: 10.1109/DSC55868.2022.00078.

[4] A. ALAzzawi, "SQL Injection Detection Using RNN Deep Learning Model," *JAETS*, vol. 5, no. 1, pp. 531–541, Dec. 2023, doi: 10.37385/jaets.v5i1.2864.

[5] Y. Liu and Y. Dai, "Deep Learning in Cybersecurity: A Hybrid BERT–LSTM Network for SQL Injection Attack Detection," *IET Information Security*, vol. 2024, pp. 1–16, Apr. 2024, doi: 10.1049/2024/5565950.

[6] J. Guo, Q. Zhang, Y. Zhao, H. Shi, Y. Jiang, and J. Sun, "RNN-Test: Towards Adversarial Testing for Recurrent Neural Network Systems," *IIEEE Trans. Software Eng.*, vol. 48, no. 10, pp. 4167–4180, Oct. 2022, doi: 10.1109/TSE.2021.3114353.

[7] D. Stiawan *et al.*, "An Improved LSTM-PCA Ensemble Classifier for SQL Injection and XSS Attack Detection," *Computer Systems Science and Engineering*, vol. 46, no. 2, pp. 1759–1774, 2023, doi: 10.32604/csse.2023.034047.

[8] N. Thalji, A. Raza, M. S. Islam, N. A. 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, doi: 10.1109/ACCESS.2023.3337645.

[9] D. Hindarto, "Comparison of RNN Architectures and Non-RNN Architectures in Sentiment Analysis," *SinkrOn*, vol. 8, no. 4, pp. 2537–2546, Oct. 2023, doi: 10.33395/sinkron.v8i4.13048.

[10] S. Sajid, "SQL Injection Dataset." [Online]. Available: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset

[11] A. Alazzawi, "Sql injection detection using rnn deep learning model", Journal of Applied Engineering and Technological Science (Jaets), vol. 5, no. 1, p. 531-541, 2023. https://doi.org/10.37385/jaets.v5i1.2864

[12] Y. Liu and Y. Dai, "Deep learning in cybersecurity: a hybrid bert–lstm network for sql injection attack detection", Iet Information Security, vol. 2024, no. 1, 2024. https://doi.org/10.1049/2024/5565950

[13] M. López-Martín, B. Carro, & A. Sánchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems", Expert Systems With Applications, vol. 141, p. 112963, 2020. https://doi.org/10.1016/j.eswa.2019.112963

[14] S. Kim, S. Yoon, J. Cho, D. Kim, T. Moore, F. Free-Nelsonet al., "Divergence: deep reinforcement learning-based adaptive traffic inspection and moving target defense countermeasure framework", Ieee Transactions on Network and Service Management, vol. 19, no. 4, p. 4834-4846, 2022. https://doi.org/10.1109/tnsm.2021.3139928

[15] M. Alghawazi, D. Alghazzawi, & S. Alarifi, "Deep learning architecture for detecting sql injection attacks based on rnn autoencoder model", Mathematics, vol. 11, no. 15, p. 3286, 2023. https://doi.org/10.3390/math11153286