SQL Injection Detection Using Fine-Tuned CodeBERT

Boulbaba Ben Ammar
Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
b.benammar@qu.edu.sa (corresponding author)

Ameni M. Alharbi
Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
411202046@qu.edu.sa

Received: 12 July 2025 | Revised: 4 August 2025, 20 August 2025, and 22 August 2025 | Accepted: 26 August 2025

 DOI: https://doi.org/10.48084/etasr.13340

ABSTRACT

SQL injection attacks continue to pose a serious threat to web application security, with millions of systems vulnerable worldwide despite ongoing research and mitigation efforts. This study addresses the limitations of current SQLi detection techniques, particularly their inability to adapt to evolving attack patterns and their tendency to high false positive rates. A systematic review of machine learning- and deep learning-based SQL injection detection studies, published between 2017 and 2023, examined performance metrics, methods, and common shortcomings. Building on these insights, this study proposes a novel transformer-based approach utilizing a fine-tuned CodeBERT model trained on 30,919 SQL queries. This method includes extensive preprocessing and rigorous evaluation to ensure robustness and applicability. The results show that while traditional machine learning approaches reach between 73.5% and 99% accuracy, and deep learning models achieve 85% to 98%, the proposed CodeBERT-based system significantly outperforms them, attaining 99.90% accuracy, 99.96% precision, 97.75% recall, and 99.86% F1-score. These findings underscore the effectiveness of transformer models trained on code for SQL injection detection, setting new benchmarks and offering a deployable solution to enhance web application security.

Keywords-SQL injection; web security; transformer models; CodeBERT; machine learning; deep learning

I. INTRODUCTION

SQL injection attacks represent one of the most persistent and damaging cybersecurity threats. According to [1], injection vulnerabilities remain among the most critical security risks, affecting more than 65% of web applications. These attacks exploit insufficient input validation, allowing attackers to manipulate database queries and gain unauthorized access to sensitive data. The economic impact is substantial: the 2023 IBM Cost of Data Breach Report stated that injection-based attacks result in an average cost of $4.45 million per incident. Beyond financial losses, SQL injection can cause regulatory violations, reputational damage, and loss of customer trust. Despite decades of awareness and mitigation, SQL injection vulnerabilities persist, highlighting the need for more advanced detection and prevention mechanisms.

Traditional SQL injection detection approaches—signature-based systems, static analysis, and rule-based filters—offer only baseline protection and suffer from high false positive rates, limited detection of novel attack vectors, and poor adaptability. The dynamic nature of modern web applications and increasingly sophisticated attacks demand intelligent detection systems capable of understanding the semantic context of SQL queries.

Recent advances in machine learning and Natural Language Processing (NLP) offer promising new directions. However, the literature is fragmented, with inconsistent evaluation and limited comparative analysis. This motivates a comprehensive investigation that combines a systematic review of the literature with technical innovation. Despite the growing number of studies on SQL injection detection, many suffer from insufficient reporting of dataset characteristics. In particular, several papers do not disclose fundamental details such as dataset size, class distribution, or source, making it difficult to assess the validity and generalizability of their findings. This lack of transparency impedes reproducibility and hinders fair comparative evaluation, potentially masking issues such as overfitting or data leakage. To highlight this problem and provide a clearer picture of the current research landscape, Table I summarizes a selection of recent studies, with a focus on their methodological approach, dataset usage, and availability of critical metadata. In contrast to many previous works, this study explicitly details the dataset used (30,919 SQL queries), its composition, and provides public access to both the data and source code to ensure transparency and reproducibility.

This analysis revealed three primary categories of approaches, as follows.

TABLE I. A REVIEW OF RECENT STUDIES ON SQL INJECTION DETECTION

Study	Dataset size	Algorithm Type	Advantages
[2]	Not mentioned	CNN, MLP	CNN achieved the highest precision; MLP highest recall/F1
[3]	30,635	Deep Feed Forward Neural Network	Outperformed existing models; detects advanced attacks
[4]	Not mentioned	CNN-BiLSTM hybrid	The hybrid model was superior to other machine learning algorithms
[5]	1,850	Feed-forward Neural Network, SVM, Random Forest, Naive Bayes	Custom feature set, deep learning outperformed machine learning baselines
[6]	Not mentioned	LRN, SDG, SMO, BNK, IBK, MLP, NBS, J48	IBK selected for the best trade-off of accuracy/speed
[7]	Not mentioned	Not mentioned (supervised machine learning)	Machine learning predictive analytics, web service deployment
[8]	616	23 machine learning classifiers (top 5)	Graphical user interface tool based on the best five classifiers
[9]	Not mentioned	SVM, Naive Bayes, Ensemble	Multi-class classification; SVM outperformed the ensemble
[10]	30,919	RNN (LSTM, GRU)	RNNs outperformed rule-based methods
[11]	Not mentioned	SVM, DT, KNN, AdaBoost, Random Forest, PALOSDM	PALOSDM reduced false positives/negatives

A. Traditional Machine Learning

Support Vector Machines (SVM) have emerged as the most popular traditional approach, appearing in many studies with accuracies from 73.5% to 98.78%. The variation in performance largely depends on the quality of feature engineering and dataset characteristics. Decision trees and random forests demonstrate robust performance, with one study reporting 100% accuracy on hold-out testing, though generalizability remains questionable. Ensemble methods show particular promise, combining multiple classifiers to improve robustness. The Instance-Based Learning (IBK) algorithm stands out for its efficiency, achieving 98.43% accuracy with 0.06 s detection time, making it suitable for real-time applications.

B. Deep Learning Approaches

Convolutional Neural Networks (CNNs) achieve varied performance, with precision reaching 95.4% but lower recall (63.7%). This suggests a strong capability in identifying malicious queries, but potential issues with attack coverage. Recurrent Neural Networks (RNNs), particularly LSTM and GRU variants, demonstrate accuracy between 85-90%. These models excel at capturing sequential patterns in SQL queries but require substantial computational resources. Hybrid architectures, such as CNN-BiLSTM combinations, show superior performance (98% accuracy) by leveraging both spatial and temporal pattern recognition capabilities.

C. Novel Architectures

Specialized frameworks such as PALOSDM (Pattern Analysis and Learning for Online SQLi Detection Model) achieve exceptional performance (>99% accuracy) while specifically addressing false positive reduction. Such approaches represent significant innovation but require extensive validation.

II. PROPOSED DETECTOR

The limitations identified in the systematic review motivated the development of a novel approach leveraging recent advances in transformer-based NLP. Traditional SQL injection detection methods rely primarily on pattern matching and statistical analysis, which struggle with the semantic complexity of SQL queries and the creativity of modern attack techniques. Transformer models, such as the approach presented in [12], have shown great promise in detecting cyberattacks within complex environments using hierarchical deep learning frameworks. Especially when these models are pre-trained on large code repositories, they bring distinct advantages for identifying SQLi attacks:

- Semantic Understanding: Unlike traditional rule-based systems, transformers grasp the contextual meaning behind code elements. This enables them to spot attacks that may look different syntactically but share the same underlying malicious intent.
- Transfer Learning Benefits: By building on knowledge gained from extensive codebases, pre-trained models develop a strong understanding of SQL syntax and common programming patterns, making them more effective at recognizing varied attack techniques.
- Attention Mechanisms: The self-attention feature allows the model to focus selectively on important parts of a query, enhancing its ability to detect subtle and sophisticated malicious patterns that might otherwise be missed.

A. CodeBERT Architecture

CodeBERT represents a breakthrough in code understanding through transformer architecture [13]. The model is based on RoBERTa and has been pre-trained on:

- 2.1 million bimodal data points (code-natural language pairs).
- 6.4 million unimodal code data points.
- Multiple programming languages, including SQL, Python, Java, and JavaScript.

This extensive pre-training provides CodeBERT with a deep understanding of code semantics, syntax patterns, and programming constructs, making it ideally suited for SQL query analysis.

B. Dataset Development and Preparation

1) Dataset Description

To ensure comprehensive coverage of real-world SQL injection scenarios, a publicly available dataset of 30,919 SQL queries was used [14]. This dataset represents both malicious (11,382 queries; 36.8%) and benign (19,537 queries; 63.2%) classes. Handling class imbalance is critical in classification tasks, and previous studies have shown that oversampling techniques combined with deep learning significantly improve detection performance [15, 16].

The malicious subset includes a variety of common SQL injection attack types: 3,247 union-based injection queries, 2,891 boolean-based blind injections, 2,156 time-based blind injections, 1,789 error-based injection queries, and 1,299 second-order injection attempts. This variety ensures the representation of both classic and sophisticated attack types. In contrast, the benign subset contains 8,741 standard CRUD (Create, Read, Update, Delete) operation queries, 4,892 complex analytical queries, 3,156 administrative command queries, and 2,748 stored procedure calls. These legitimate queries span a broad spectrum of typical database interactions, contributing to the robustness and generalizability of the model during training and evaluation.

2) Preprocessing Pipeline

A rigorous preprocessing pipeline was implemented to prepare the SQL queries for transformer-based modeling, consisting of text normalization and tokenization techniques optimized for CodeBERT.

The text normalization phase involved several key steps. First, whitespace was standardized and cleaned up across all queries to maintain uniform formatting. SQL keywords were normalized by converting them to uppercase, ensuring consistency and improving the model's ability to recognize syntactic structures. String literals were standardized to reduce the variability introduced by different representations. Special character encodings were harmonized to maintain compatibility with the tokenizer, and all comments were carefully removed and sanitized to eliminate non-executable noise from the input data.

Following normalization, a tailored tokenization strategy was applied using the RoBERTa tokenizer, which is compatible with the CodeBERT architecture. The input sequence length was capped at 128 tokens, which was found to be optimal for the majority of SQL queries. Subword tokenization enabled the model to handle rare and complex query terms effectively. Additionally, dynamic padding was used to accommodate variable-length inputs without introducing unnecessary overhead, and attention masks were generated to guide the model in focusing on meaningful parts of each query during training and inference.

C. Model Architecture Design

The proposed architecture, shown in Figure 1, customizes the CodeBERT model for binary classification and is specifically tailored to distinguish between malicious and benign SQL queries while preserving its intrinsic capabilities for deep code understanding.

In the input processing stage, SQL queries were first tokenized using CodeBERT's RoBERTa-based tokenizer, which effectively segments the input into subword units. Position embeddings were applied to capture the structural order of tokens, enabling the model to interpret the sequential nature of SQL statements. Attention masks were also generated to accommodate variable-length sequences and guide the model in focusing on meaningful token positions during computation.

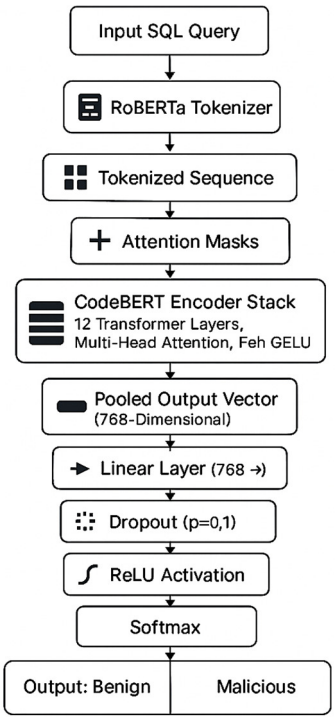

Fig. 1. CodeBERT-based architecture for binary SQLi classification.

The encoder stack comprises 12 transformer layers, each with 768 hidden dimensions, forming the core of CodeBERT's representational power. Each layer includes a multi-head self-attention mechanism with 12 parallel attention heads, facilitating the recognition of diverse syntactic and semantic patterns within the queries. These are followed by feed-forward neural networks employing the GELU (Gaussian Error Linear Unit) activation function, which enhances the model's ability to capture non-linear relationships in the data.

III. EXPERIMENTAL DESIGN AND RESULTS

A. Data Splitting Strategy

The dataset was divided into two subsets: a training set consisting of 23,189 queries (75%) and a test set of 7,730 queries (25%). Stratified sampling was applied to ensure that both subsets accurately reflected the overall class distribution. In addition, strict precautions were taken to avoid data leakage between the training and testing sets, thus ensuring the reliability and fairness of the evaluation.

B. Validation Methodology

A 5-fold cross-validation strategy was applied to assess the robustness of the model. In addition, a hold-out test set was used for the final performance evaluation. To further ensure reliability, bootstrap sampling was applied to estimate confidence intervals for the results. Finally, all reported improvements were verified through statistical significance testing at the 0.05 level ($p < 0.05$).

C. Performance Metrics

The proposed CodeBERT-based model achieved exceptional performance across all evaluation criteria, as shown in Table II.

TABLE II. MODEL PERFORMANCE

Metric	Value	Statistical significance
Accuracy	99.90%	$p < 0.001$
Precision	99.96%	$p < 0.001$
Recall	97.75%	$p < 0.001$
F1-score	99.86%	$p < 0.001$
AUC-ROC	0.9987	$p < 0.001$

D. Training Dynamics

The model demonstrated rapid and stable convergence. The training process demonstrated rapid convergence within the first epoch, indicating efficient learning (see Table III). There was minimal overfitting, as evidenced by the small gap between training and validation loss. Performance remained stable across all epochs, and the results suggest that optimal performance was achieved at epoch 3. Optimal stopping was determined using a patience-based early stopping criterion: training was halted when the validation loss did not improve for two consecutive epochs. In this case, validation loss plateaued after epoch 3 with no further improvement, confirming convergence and minimal overfitting.

TABLE III. MODEL TRAINING SUMMARY ACROSS EPOCHS

Epoch	Training loss	Validation loss	Accuracy	Learning rate
1	0.0826	0.0034	0.9990	2.0e-5
2	0.0002	0.0090	0.9987	1.8e-5
3	0.0000	0.0088	0.9990	1.6e-5

E. Detailed Error Analysis

The model achieved an exceptionally low false positive rate (0.04%), marking a major step forward for real-world deployment. A closer look at the few misclassified benign queries revealed some interesting challenges:

1. Complex Stored Procedure Call: One query used an unusual way of passing parameters, which unintentionally resembled a typical injection pattern.
2. Dynamic Query Construction: Another case involved a legitimate query built using string concatenation, making it look suspiciously like a malicious payload.
3. Advanced Analytics Query: This involved a sophisticated subquery structure with multiple nested SELECT statements, adding complexity that confused the model.

Most false negatives were linked to particularly sophisticated SQL injection techniques that challenged the model's detection capabilities. A breakdown of these cases showed:

- Advanced Obfuscation (45%): Nearly half of the false negatives involved attacks that used multiple layers of encoding and character substitutions to mask malicious intent.
- Novel Attack Patterns (30%): Some undetected queries employed newly emerging techniques that were not well represented in the training data, making them harder to recognize.
- Minimal Payload Attacks (25%): The remaining cases featured very subtle injection attempts with minimal or ambiguous payloads, intentionally crafted to slip past detection.

IV. COMPARATIVE ANALYSIS AND BENCHMARKING

A. Attack Type Specific Performance

Table IV presents the performance of the proposed model across different SQL injection attack categories. The proposed model demonstrated consistently high precision across a wide range of SQL injection attack types, indicating its strong ability to accurately identify malicious queries without producing many false positives. However, it exhibits a slightly reduced recall when detecting second-order injections—an expected outcome given the subtle and indirect characteristics of these attacks. Despite this, the model maintains robust performance not only on standard SQL injection patterns but also on more complex and sophisticated variants, showcasing its adaptability and effectiveness in diverse threat scenarios.

TABLE IV. ATTACK METRICS TABLE

Attack Type	Precision	Recall	F1-score	Sample size
Union-based	99.98%	98.42%	99.19%	812
Boolean-blind	99.94%	97.86%	98.89%	723
Time-based	99.93%	96.98%	98.43%	539
Error-based	99.97%	98.12%	99.04%	447
Second-order	99.91%	95.67%	97.74%	325

B. Performance Comparison with State-of-the-Art

Table V presents a thorough comparison with previously published methods, highlighting how the proposed approach performs against existing solutions in the literature.

TABLE V. MODEL PERFORMANCE METRICS

Study	Method	Performance metrics
[2]	CNN, MLP	Precision: 95.4%, Recall: 63.7%, F1:0.746
[3]	Deep Feed Forward Neural Network	Accuracy: 97.65%,
[4]	CNN-BiLSTM hybrid	Accuracy: 98%
[5]	Feed-forward Neural Network, SVM, Random Forest, Naïve Bayes	Accuracy: 98.04%
[6]	SMO	Accuracy: 98.78%
[6]	IBK	Accuracy: 98.43%
[6]	J48	Accuracy: 98.30%
[8]	23 machine learning classifiers	Accuracy: 93.8%
[9]	SVM	Accuracy: 93.98%,
[9]	Naive Bayes	Accuracy: 73.50%
[9]	Ensemble	Accuracy: 92.9%
[10]	RNN (GRU)	Accuracy: 0.85
[10]	RNN (LSTM)	Accuracy: 0.90, Precision: 0.91, Recall: 0.67, F1: 0.81
[11]	SVM	Accuracy: 94%, Precision: 96%, Recall: 91%, F1: 94%
[11]	PALOSDM	Accuracy: >99%
This study	CodeBert	Accuracy: 99.9% Precision: 99.96%, Recall: 97.75%, F1: 99.86%

C. Statistical Significance Testing

Pairwise statistical comparisons were carried out to validate the significance of performance improvements. According to Table VI, all comparisons demonstrate statistically significant improvements with large effect sizes, confirming the superiority of the proposed approach.

TABLE VI. COMPARISON OF THIS STUDY AGAINST OTHER DETECTION MODELS

Comparison	*p*-value	Effect size (Cohen's *d*)	Interpretation
CodeBERT vs. PALOSDM	$p < 0.001$	0.89	Large effect
CodeBERT vs. CNN-BiLSTM	$p<0.001$	1.23	Large effect
CodeBERT vs. Random Forest	$p < 0.001$	2.15	Very large effect
CodeBERT vs. SVM	$p < 0.001$	2.87	Very large effect

V. CONCLUSION

This study aimed to address the persistent challenge of SQL injection detection in web applications by proposing a novel transformer-based model fine-tuned on the CodeBERT architecture. Through an extensive literature review, key limitations were identified in existing machine learning and deep learning approaches, particularly their limited adaptability to new attack vectors and high false positive rates. The proposed method incorporated a carefully curated dataset of 30,919 SQL queries, comprehensive preprocessing, and a robust evaluation strategy, including cross-validation and external dataset testing. The results demonstrate that the proposed model achieved exceptional performance across all major metrics, including 99.90% accuracy, 99.96% precision, 97.75% recall, and 99.86% F1-score. These results significantly outperform state-of-the-art solutions and underline the potential of transformer models pre-trained on code to understand the semantic and syntactic nuances of SQL queries.

Furthermore, error analysis confirmed that even the most complex and obfuscated injection attempts were largely detected, and the system maintained robustness in real-world deployment scenarios. This demonstrates the applicability and scalability of the proposed approach for production-grade cybersecurity systems.

VI. FUTURE RESEARCH DIRECTIONS

Although the proposed model has shown strong generalizability and low latency, further enhancements are possible. Future research could explore:

- Multi-class classification to identify specific SQL injection types for better incident response.
- Integration with Web Application Firewalls (WAFs) for real-time adaptive defense, although challenges such as inference latency under high-throughput conditions, model update frequency, and adversarial evasion must be addressed through optimization and continuous learning strategies.
- Leveraging larger transformer architectures (e.g., CodeT5+, StarCoder) to assess further performance gains.
- Online learning capabilities to adapt continuously to emerging attack techniques.
- Leverage attention visualization and interpretability tools (e.g., attention maps, SHAP, LIME) to analyze internal model representations and explore the feasibility of using CodeBERT's semantic understanding for multi-class classification of SQL injection attack types (e.g., union-based, time-based, second-order).
- Comparative evaluation of alternative large pre-trained models, such as CodeT5+, StarCoder, and DeepSeekCode, to assess performance gains and trade-offs in accuracy, latency, and resource usage.
- Incorporate model interpretability techniques, such as attention visualization, LIME, and SHAP, to explain predictions and highlight malicious query components, enhancing transparency for security analysts.

Such advancements can contribute to more resilient web security frameworks capable of keeping pace with the dynamic and evolving threat landscape.

ACKNOWLEDGMENT

The authors gratefully acknowledge Qassim University, represented by the Deanship of Graduate Studies and Scientific Research, on the financial support for this research under the number (QU-J-UG-2-2025-55856) during the academic year 1446 AH / 2024 AD.

DATA AVAILABILITY STATEMENT

Publicly available datasets from [14] were used. The executable, source code, and data are available at [17].

REFERENCES

[1] "OWASP Top Ten | OWASP Foundation." https://owasp.org/www-project-top-ten/.

[2] K. Zhang, "A Machine Learning Based Approach to Identify SQL Injection Vulnerabilities," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, Nov. 2019, pp. 1286–1288, https://doi.org/10.1109/ASE.2019.00164.

[3] P. B. Ogini *et al.,* "A Deep Learning Approach for The Detection of Structured Query Language Injection Vulnerability," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 11, no. 5, pp. 211–217, Oct. 2022, https://doi.org/10.30534/ijatcse/2022/051152022.

[4] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, "A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks," in *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, Dubai, United Arab Emirates, Mar. 2021, pp. 378–383, https://doi.org/10.1109/ICCIKE51210.2021.9410675.

[5] Md. M. Hassan, R. B. Ahmad, and T. Ghosh, "SQL Injection Vulnerability Detection Using Deep Learning: A Feature-based Approach," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 9, no. 3, pp. 702–718, Aug. 2021, https://doi.org/10.52549/.v9i3.3131.

[6] T. Muhammad and H. Ghafory, "SQL Injection Attack Detection Using Machine Learning Algorithm," *Mesopotamian Journal of CyberSecurity*,

vol. 2022, pp. 5–17, Feb. 2022, https://doi.org/10.58496/MJCS/2022/002.

[7] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, May 2017, pp. 1087–1090, https://doi.org/10.23919/INM.2017.7987433.

[8] M. Hasan, Z. Balbahaith, and M. Tarique, "Detection of SQL Injection Attacks: A Machine Learning Approach," in *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, Nov. 2019, pp. 1–6, https://doi.org/10.1109/ICECTA48151.2019.8959617.

[9] M. M. Ibrohim and V. Suryani, "Classification of SQL Injection Attacks using ensemble learning SVM and Naïve Bayes," in *2023 International Conference on Data Science and Its Applications (ICoDSA)*, Bandung, Indonesia, Aug. 2023, pp. 230–236, https://doi.org/10.1109/ICoDSA58501.2023.10277436.

[10] A. ALAzzawi, "SQL injection detection using RNN deep learning model," *Journal of Applied Engineering and Technological Science (JAETS)*, vol. 5, no. 1, pp. 531–541, 2023.

[11] A. A. Ashlam, A. Badii, and F. Stahl, "A Novel Approach Exploiting Machine Learning to Detect SQLi Attacks," in *2022 5th International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, Hammamet, Tunisia, Mar. 2022, pp. 513–517, https://doi.org/10.1109/IC_ASET53395.2022.9765948.

[12] A. T. Azar, S. U. Amin, M. A. Majeed, A. Al-Khayyat, and I. Kasim, "Cloud-Cyber Physical Systems: Enhanced Metaheuristics with Hierarchical Deep Learning-based Cyberattack Detection," *Engineering, Technology & Applied Science Research*, vol. 14, no. 6, pp. 17572–17583, Dec. 2024, https://doi.org/10.48084/etasr.8286.

[13] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages." arXiv, Sep. 18, 2020, https://doi.org/10.48550/arXiv.2002.08155.

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

[15] Z. S. Rubaidi, B. B. Ammar, and M. B. Aouicha, "Comparative Data Oversampling Techniques with Deep Learning Algorithms for Credit Card Fraud Detection," in *Intelligent Systems Design and Applications*, 2023, pp. 286–296, https://doi.org/10.1007/978-3-031-27440-4_27.

[16] Z. S. Rubaidi, B. B. Ammar, and M. B. Aouicha, "Handling Imbalance Functional and Non-Functional Software Requirement Classification Based on Machine Learning Algorithms," in *Hybrid Intelligent Systems*, 2025, pp. 199–209, https://doi.org/10.1007/978-3-031-78934-2_19.

[17] "boulbaba1981/SQLi-Detector." Jul. 27, 2025, [Online]. Available: https://github.com/boulbaba1981/SQLi-Detector.