OPEN ACCESS
International Journal of Advances in Artificial Intelligence and Machine Learning Vol 2 No 3 (2025) https://doi.org/10.58723/ijaaiml.v2i3.515

Check for updates

Research Article

Enhancing SQL Code Security and Maintainability: A Deep Learning Based Approach

Received: September 12, 2025	Revised: October 20, 2025	Accepted: November 07, 2025	Publish: November 22, 2025

Faisal Alghamdi*, Boulbaba Ben Ammar

Abstract:

Background of study: SQL injection attacks continue to pose a significant risk to online systems. Traditional rule-based detection regularly fails to identify emerging or disguised attack vectors. Deep learning holds significant promise for robust detection, yet few studies rigorously compare model types or examine how to convey detection results as actionable security advice for developers.

Aims and scope of paper: Building on this gap in existing research, this study tests three deep learning models for detecting SQL injection: Convolutional Neural Network (CNN), Bidirectional Long Short-Term Memory (BiLSTM), and DistilBERT. The best model is then utilized in a tool that provides developers with risk assessments, warnings about unsafe patterns, and examples of secure queries.

Methods: To achieve this, a dataset of 30,919 labeled SQL queries was preprocessed using normalization, syntax validation, and stratified splitting (70/15/15). A dual tokenization approach enabled fair comparisons between architectures. Models were trained using Adam/AdamW optimizers and evaluated for accuracy, precision, recall, F1-score, AUC-ROC, and MCC.

Result: Among the tested models, DistilBERT set the performance benchmark, achieving 99.8% accuracy, 99.9% precision, 99.5% recall, and a false positive rate of just 0.1%. CNN and BiLSTM showed strong results, but proved weaker against obfuscated or distributed attacks. The SQL Security Advisor system converts model predictions directly into actionable guidance for developers.

Conclusion: In conclusion, our findings indicate that DistilBERT detects SQL injections more effectively than CNN and BiLSTM, particularly when attacks are complex or hidden. By combining detection, explanation, and repair, this approach helps bring research closer to real-world use and supports developers in building more secure systems.

Keywords: BiLSTM; CNN; Deep Learning; DistilBERT; Security Advisory; SQL Injection.

1. INTRODUCTION

Structured Query Language (SQL) serves as the foundation for modern data-driven infrastructure, including applications in banking, healthcare, e-commerce, and cloud-native systems (Dritsas & Trigka, 2025). Its widespread use has made it a popular target for assaults, notably SQL injection (SQLi), which remains one of the most serious dangers to online applications (Chakir & Sadqi, 2025) (OWASP Foundation, 2025). Traditional rule-based detection systems, such as input sanitization, keyword blacklisting, and Web Application Firewalls, have been used to mitigate these threats, but they are brittle when confronted with obfuscated or evolving payloads, frequently producing false positives that undermine trust in protective systems (Bhupathiraju, 2025). These constraints jeopardize the security, integrity, and availability of mission-critical databases, emphasizing the need for more flexible solutions.

To overcome these disadvantages, new research has focused on machine learning and deep learning algorithms for SQLi detection and reporting, which have enhanced generalization and resilience when compared to traditional methods. For example, convolutional neural networks (CNNs) extract strong features from query structures (Muduli et al., 2024), while bidirectional long short-term memory networks (BiLSTMs) capture contextual and sequential dependencies (Takyi et al., 2025). Furthermore, transformer-based models, such as DistilBERT, utilize semantic representations to detect sophisticated payloads (Liu & Dai, 2024). Despite these advances, current research often prioritizes raw detection accuracy over practical factors, such as inference latency, reproducibility, and actionable recommendations for

developers (Floris et al., 2025) (Zivkovic & Jovanovic, 2024).

What remains unexplored is a thorough evaluation of different designs using a single pipeline that not only detects dangerous queries but also recommends secure query options. Consequently, without such direction, security systems may signal vulnerabilities without supporting developers in their repair, thereby limiting their real-world effectiveness. To address this gap, this study bridges these needs by creating and testing a unified framework for detecting SQL injections and optimizing queries. Specifically, three deep learning models CNN, BiLSTM, and DistilBERT are utilized in a six-step pipeline that encompasses data collection, preprocessing, feature engineering, model building, training, assessment, and deployment. Beyond detection, the system includes recommendation generation, providing developers with actionable solutions for replacing problematic queries with secure alternatives.

This study makes three distinct contributions. First, it compares CNN, BiLSTM, and DistilBERT for SQLi detection under consistent conditions. Second, it adds an integrated advisor system a suggestion engine to bridge the gap between threat identification and proactive security policies. Third, it provides a framework that is both replicable and expandable, making it suitable for both academic study and industry applications. Guided by these aims, this effort is motivated by three primary research questions, which are detailed below for clarity.

1. Can a single deep learning framework identify SQL injections while also providing developer-centric guidance on crafting secure and efficient queries?
2. How can architectural decisions, such as tokenization techniques and model selection, be optimized to deliver high detection accuracy while remaining practical for deployment in latency-sensitive environments?
3. Is it possible to close the detection remediation gap by converting model outputs into interpretable, rule-based suggestions that retain semantic intent while eliminating vulnerabilities and inefficiencies?

Research on SQL injection detection and query optimization has advanced in stages. Early defenses used rule-based systems. Lexical analysis with taint tracking (Gomes et al., 2025) and syntactic validation with Web Application Firewalls (Shaik & Manoharan, 2025) were effective against known attack signatures, enabling real-time filtering. (OWASP Foundation, 2025) consistently ranks SQL injection as one of the top web security risks, supporting the use of these methods. However, these systems needed frequent manual updates and were often bypassed by obfuscation strategies. This limited their effectiveness in changing environments.

In response to the limitations of rule-based systems, researchers advanced toward machine learning models that relied on statistical patterns rather than static rules. Classical supervised approaches, such as Random Forests and Support Vector Machines, produced encouraging results with F1-scores above 0.90 in several benchmarks (Chen et al., 2025) (Yetunde et al., 2025). Nevertheless, these methods depended heavily on handcrafted features, making them expensive to scale and vulnerable to dataset bias. Deep learning technologies accelerated this trajectory by learning hierarchical representations of queries directly from data. For example, LSTM-based models have captured sequential relationships within SQL statements (Takyi et al., 2025), while convolutional neural networks (CNNs) excel at extracting n-gram features and achieve accuracies exceeding 98% on benchmark datasets (Muduli et al., 2024). Despite their progress, both designs struggled with lengthy and complex queries, which reduced their effectiveness in real-world traffic conditions.

Building on these foundations, more recent research explored sophisticated designs and hybrid strategies. For instance, graph-based models such as NL-GCN, paired with FastText embeddings, achieved near-perfect identification rates (Gupta et al., 2025). Similarly, query dependency graph-based GNNs showed promising detection accuracy (Vangapandu et al., 2025). In addition, transformer-based approaches, notably BERT variations, introduced semantic understanding to SQL detection. Hybrid BERT-LSTM architectures with obfuscation-aware preprocessing achieved F1-scores above 0.97 (Liu & Dai, 2024), while ensemble techniques, such as BERT combined with AdaBoost or regex-ML hybrids, consistently produced accuracies above 99% (Souza et al., 2024) (Zivkovic & Jovanovic, 2024). Collectively, these results demonstrate the growing maturity of deep learning algorithms while also highlighting trade-offs between performance and computational cost. The availability of open datasets, such as the SQL injection dataset by (Sajid576, 2021), has further enabled consistent benchmarking and reproducibility across studies.

Beyond detection, research efforts expanded to investigate system robustness and adaptability. In this context, adversarial training techniques, such as gradient-free payload generation (Floris et al., 2025) and poisoning-resistant neural networks (Al-Mallah & Quintero, 2025), have improved resilience under adversarial conditions. However, these methods often rely on synthetic datasets without sufficient validation in production environments. Furthermore, domain-specific adaptations, such as embedding Random Forest classifiers into healthcare applications (Mariettou et al., 2025), demonstrated feasibility in real-time systems but were narrowly scoped. Parallel research in performance optimization introduced innovations such as learned cardinality estimation (Yi et al., 2025), reinforcement learning for join selection (Keshireddy, 2025), and intelligent cost-performance tuning for big data analytics (Lyu, 2025). Complementary work explored cost-based query optimization for large-scale ERD designs (Lubis et al., 2025), efficient cardinality

estimation for distributed systems (Mahin, 2025), and the integration of AI-driven methods to enhance query processing in professional software development environments (Swaroop, 2025). These studies improved query efficiency but often overlooked direct links to SQL injection security.

Taken together, this body of literature demonstrates significant progress in SQL injection detection and performance optimization. Nonetheless, across all stages, three recurring limitations persist: detection and optimization are typically treated separately; predictions rarely translate into actionable recommendations; and deployment factors such as scalability, inference latency, and interpretability remain underexplored. Motivated by these gaps identified in earlier research stages, the current study integrates CNN, BiLSTM, and DistilBERT into a cohesive framework that addresses both detection and optimization, while also delivering secure recommendations to guide developers in practice.

2. MATERIAL AND METHOD

This study employed a systematic approach to ensure that SQL injection detection and query optimization were valid, reliable, and reproducible. The platform combines thorough data gathering, advanced preprocessing, multi-architectural deep learning models, and rigorous assessment methods. Three architectures, namely CNN, BiLSTM, and DistilBERT, were utilized to detect fraudulent SQL queries and generate secure suggestions. The procedure was divided into six steps: data collection, preprocessing and feature engineering, model construction, training, assessment, and deployment. To provide methodological clarity, Figure 1 depicts a schematic overview of the process, including the sequential flow and relationships between its key components.

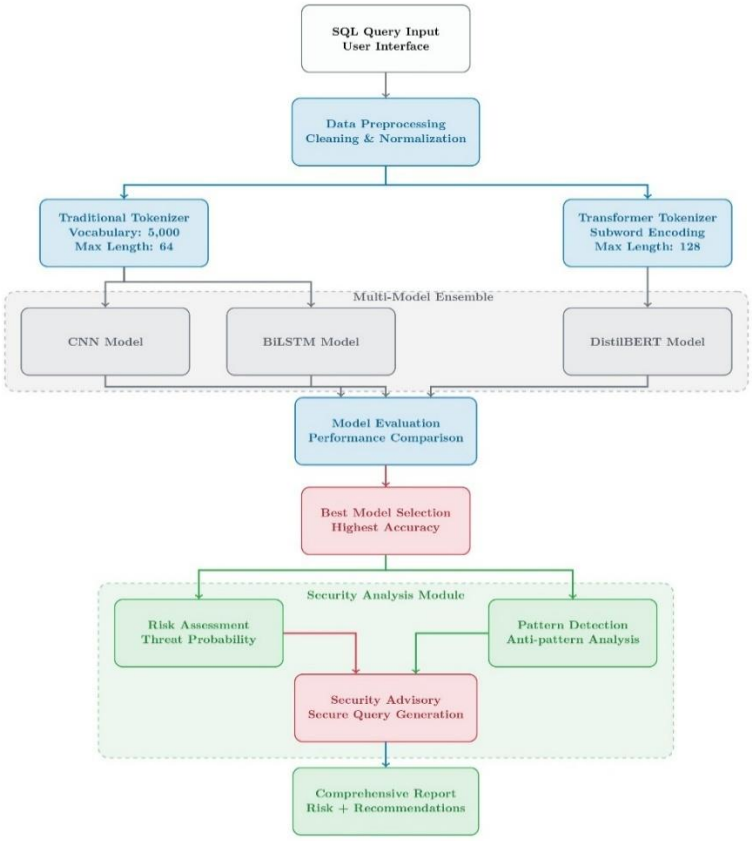

Figure 1. Comprehensive System Architecture for Multi-Model SQL Injection Detection.

The dataset was obtained from a publicly accessible Kaggle repository specifically designed for SQL injection detection (URL: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset/data, accessed April 5, 2025). It included over 30,000 structured SQL query examples labeled for binary categorization. Two types of queries were defined: benign questions, such as conventional SELECT or UPDATE commands, and malicious queries, which included injection methods, obfuscation, time-based payloads, or enumeration efforts. Each record included both the SQL query and the binary label. The dataset was well-balanced across classes, making it appropriate for supervised learning applications. Table 1 provides examples that illustrate the range of requests and attack patterns. While extensive, the collection is confined to specified signatures and may not capture all emergent real-world risks, which is recognized in the scope of this effort.

Table 1. Representative samples from the SQLi Dataset.

Query	Label
" or pg_sleep(TIME) -	1
CREATE USER name IDENTIFIED BY pass123 TEMPORARY TABLESPACE temp DEFAULT TABLESPACE users;	1
SELECT TOP 3 * FROM serve WHERE instant = 'slow'	0
SELECT * FROM applied	0
SELECT * FROM sharp WHERE cool = 'start' LIMIT 3	0
SELECT * FROM arm WHERE canal = 'wagon' FETCH FIRST 3 ROWS ONLY	0

To ensure data quality and model readiness, preprocessing was performed using Python 3.13 with the following libraries: pandas, NumPy, scikit-learn, Keras, and Hugging Face Transformers. Null or faulty queries were deleted, duplicates were removed, and SQL texts were normalized by removing extra whitespace and standardizing quotation marks. To ensure structural accuracy, queries were parsed and verified using the SQL parse package (version 0.4.4). Stratified sampling was employed to divide the dataset into training (70%), validation (15%), and test (15%) subsets, with class balance maintained across all partitions.

Tokenization techniques were adapted to the model architecture. CNN and BiLSTM models utilized word-level tokenization on a 5,000-word vocabulary, padding or truncating inputs to a maximum of 64 tokens. DistilBERT tokenized subwords using Byte-Pair Encoding, with specific classification tokens ([CLS], [SEP]) and a maximum sequence length of 128. This dual technique guaranteed that each model received optimally organized input while remaining comparable. Figure 2 shows how the identical question was handled differently in each model.

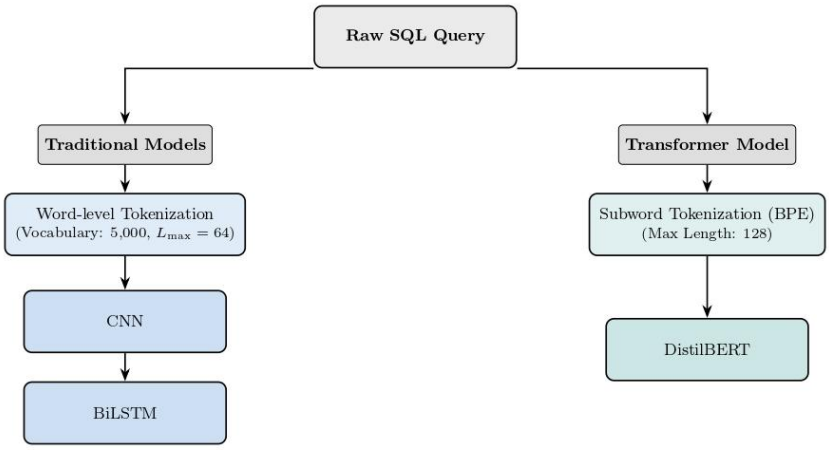

Figure 2. Innovative Dual Tokenization Strategy Demonstrating Parallel Processing Pathways for Optimal SQL Query Representation Across Diverse Neural Architectures.

To capture complementary viewpoints on malicious queries, three deep learning models were implemented: a refined DistilBERT transformer for contextual and semantic relationships, a Convolutional Neural Network (CNN) for local n-gram patterns, and a Bidirectional Long Short-Term Memory network (BiLSTM) for sequential dependencies. CNN employed global max pooling, 1D convolution with a kernel size of 3, and 64-dimensional embeddings. For classification, the BiLSTM concatenated 32 hidden units in each direction. To maintain linguistic knowledge, DistilBERT used a pre-trained transformer with a classification head that was adjusted over three epochs at a learning rate of 2×10^{-5} (Sanh et al., 2019).

TensorFlow 2.20/Keras (CNN, BiLSTM) and PyTorch 2.8 (DistilBERT) were used to train the models on an NVIDIA RTX 3090 GPU. Every model had a single training session, which lasted between two and four hours. For up to ten epochs, the CNN and BiLSTM models employed the Adam optimizer with a learning rate of 10^{-3}, with early termination occurring after three validation cycles that were stationary. The AdamW optimizer and weight decay ($\lambda = 0.01$) were used to fine-tune DistilBERT, utilizing reduced batch sizes due to memory limitations. Table 2 provides a summary of the hyperparameters.

Table 2. Hyperparameter configuration for CNN, BiLSTM, and DistilBERT

Parameter	CNN	BiLSTM	DistilBERT	Optimization Rationale
Embedding Dimension	64	64	768	Balance representation capacity with efficiency
Sequence Length	64	64	128	Architecture-specific input constraints
Batch Size	32	32	16	GPU memory optimization
Learning Rate	10^{-3}	10^{-3}	2×10^{-5}	Architecture-specific convergence tuning
Dropout Rate	0.3	0.3	0.1	Regularization strength optimization
Weight Decay	10^{-4}	10^{-4}	0.01	L2 penalty for AdamW optimizer
Epochs	10	10	3	Convergence and overfitting analysis
Early Stopping	3	3	2	Overfitting prevention
Optimizer	Adam	Adam	AdamW	Architecture-specific optimization efficiency

Using scikit-learn, the evaluation used the following metrics: accuracy, precision, recall, F1-score, AUC-ROC, and Matthews Correlation Coefficient (MCC). These measures, which represent the asymmetric costs of mistakes in SQL injection detection, were selected to strike a balance between sensitivity to false positives and false negatives. To provide consistent comparisons across models within computational constraints, fixed train/validation/test splits were employed instead of cross-validation.

A security analysis and advising mechanism was incorporated into the framework to go beyond binary categorization. This module utilized continuous risk ratings and categorical categories (low, medium, high, and critical) to transform unprocessed forecasts into actionable insights. Critical queries were those with a malicious probability greater than 0.9, while low-risk queries were those with a malicious probability less than 0.5. Additionally, a rule-based anti-pattern detection engine was employed, utilizing regular expression analysis and string matching to identify vulnerabilities. Table 3 enumerated the following categories: injection vectors, time-based attacks, information disclosure, performance inefficiencies, and structural violations.

Table 3. SQL Anti-Pattern Categories and Risk Classifications

Category	Representative Examples	Risk Classification
Classic Injection Vectors	OR '1'='1', UNION SELECT, '; DROP TABLE	Critical
Time-based Attack Patterns	SLEEP(), BENCHMARK(), WAITFOR DELAY	Critical
Information Disclosure	information_schema, @@version, USER()	High
Performance Anti-patterns	SELECT *without WHERE, leading LIKEwildcards	Medium
Structural Violations	Missing JOINconditions, unbalanced quotations	Low

To assist developers with cleanup, the system produced secure query templates. Here, a "query template" refers to a predefined structure for database queries that includes placeholders for user input. For medium- and low-risk queries, the templates were parameterized (using variables instead of directly inserting user input) and annotated with suggestions for secure usage. In contrast, high-risk queries were replaced with default safe templates, which use conservative settings to minimize risk. This not only identified vulnerabilities but also offered practical preventive solutions.

To ensure repeatability, all code and configuration files will be made publicly accessible once published.

The technique provides a robust and reproducible solution for preventing SQL injection attacks through its integrated architecture, which combines deep learning (a form of artificial intelligence that analyzes data patterns), rule-based detection (a system that utilizes predefined patterns to identify threats), and practical guidance for developers.

3. RESULT AND DISCUSSION

3.1 Results

The three models excelled in all assessment parameters accuracy, precision, recall, F1-score, AUC-ROC, and MCC achieving total accuracies above 98.9%. DistilBERT led with 99.8% accuracy, 99.9% precision, and 99.7% F1-score, surpassing CNN and BiLSTM. These results confirm that transformer architectures, with their contextual representations and attention mechanisms, outperform sequential or convolutional models in detecting complex SQL injection patterns.

Table 4. Comparative Performance of Deep Learning Models for SQL Injection Detection

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC	MCC
CNN	0.996	0.998	0.991	0.994	0.995	0.992
BiLSTM	0.993	0.989	0.992	0.991	0.993	0.986
DistilBERT	**0.998**	**0.999**	**0.995**	**0.997**	**0.997**	**0.995**

While CNN and BiLSTM had similar results, minor differences arose. CNN achieved high accuracy and low latency but showed a slightly higher false positive rate, suggesting potential issues with over-flagging harmless queries in high-throughput systems. BiLSTM excelled in recall, demonstrating its ability to detect real attacks, but produced more false positives than DistilBERT. Overall, these findings align with the existing literature, which shows that transformer models outperform standard deep learning models for tasks involving subtle language patterns.

Error Analysis through Confusion Matrices

The confusion matrix analysis provided further insights into class-level behaviors. CNN caused more false positives, which might contribute to alert fatigue in actual deployments. BiLSTM had strong specificity with few false positives, but it had more false negatives, indicating that some threats were missed. In contrast, DistilBERT achieved the best balance, with very low rates of both false positives (0.1%) and false negatives (0.4%). These findings support recent code security research, indicating that transformer models are particularly successful at identifying obfuscated or polymorphic SQL payloads.

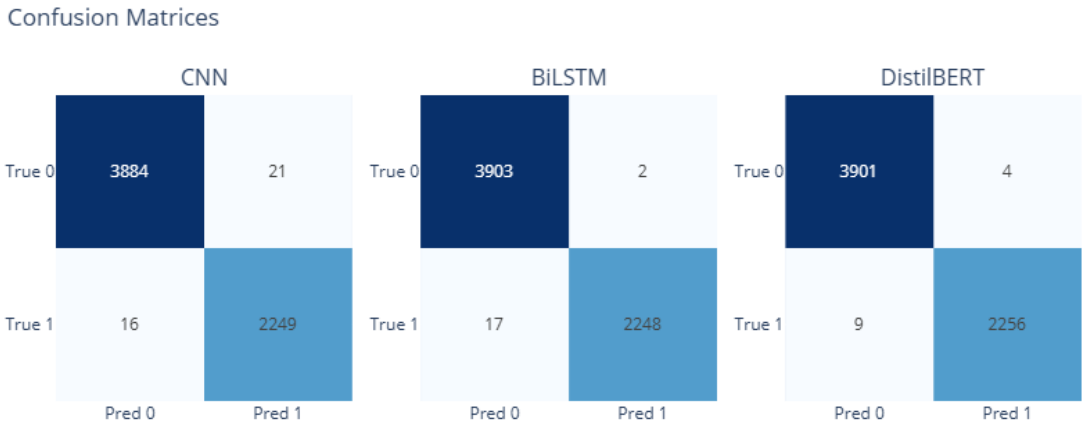

Figure 3. Confusion matrices for CNN, BiLSTM, and DistilBERT Models.

Training Dynamics and Convergence

An analysis of training and validation dynamics revealed that all models converged rapidly. CNN attained stability during the first epoch, but BiLSTM showed modest overfitting by the third epoch. DistilBERT maintained a consistent validation accuracy (99.8%) while exhibiting the lowest loss values, showing good generalization potential. Although it had a greater per-step processing cost, its efficiency in converging within three epochs demonstrates its suitability for deployment where precision takes precedence over speed.

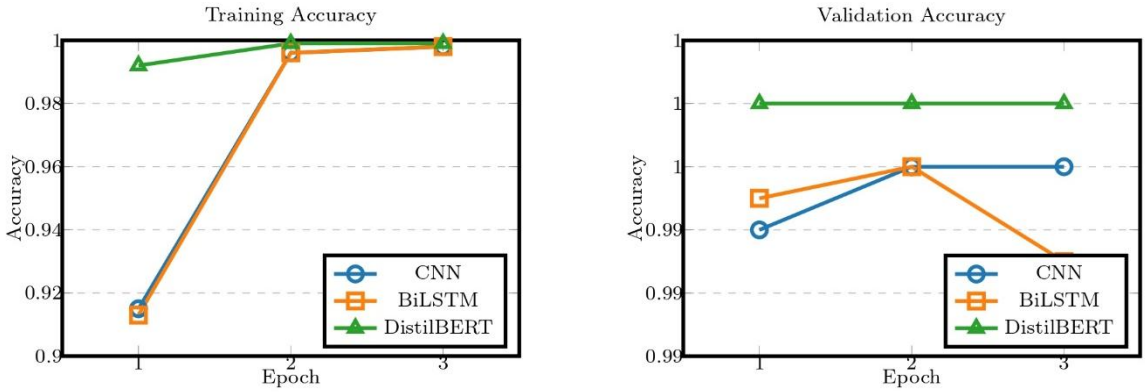

Figure 4. Training and Validation Accuracy of Models Across Epochs.

Practical Application: SQL Security Advisor

Building on DistilBERT's outstanding performance, the SQL Security Advisor was created to illustrate a real-world application. Beyond binary classification, the system offers risk stratification, anti-pattern identification, query rewriting, and actionable

suggestions in accordance with OWASP and NIST criteria. This integration bridges a significant gap between academic modeling and actual deployment, serving as both a defensive security measure and a development assistance tool. As a consequence, the findings not only verify the efficiency of deep learning in SQLi detection but also demonstrate its ability to shape secure coding standards.

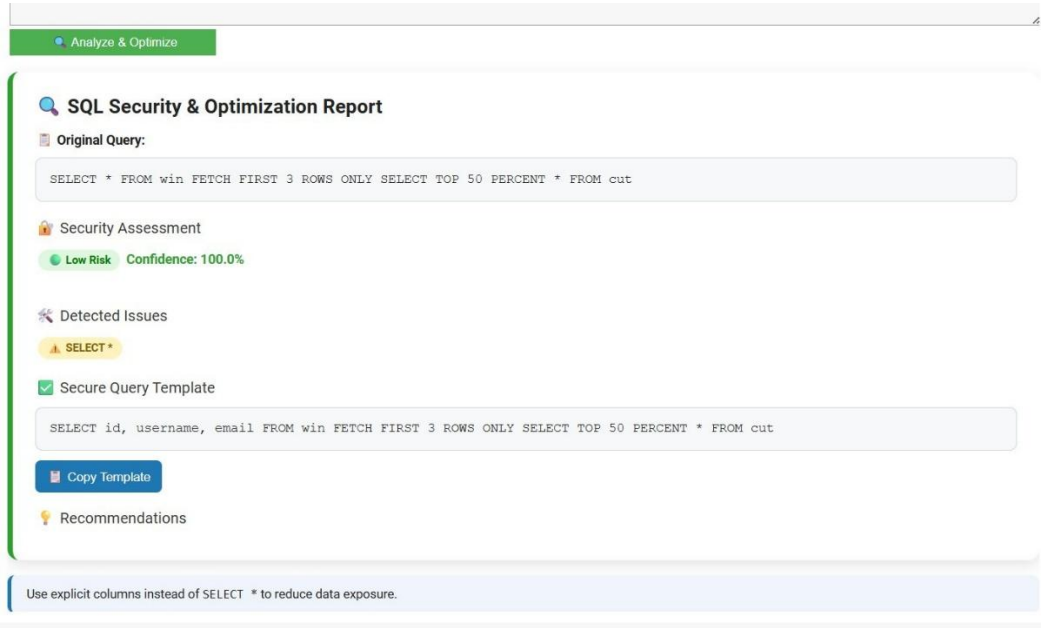

Figure 5. Deployment Interface of The SQL Security Advisor Showing Query Analysis, Risk Assessment, Detected Issues, And Secure Rewriting Recommendations.

3.2 Discussion

This study compared three deep learning models for detecting SQL injection: CNN, BiLSTM, and a fine-tuned DistilBERT. All models performed well, with accuracy above 99%. This shows that deep learning can effectively identify injection patterns. DistilBERT stood out, achieving 99.8% accuracy, 99.9% precision, 99.5% recall, a 99.7% F1-score, and an MCC of 0.995. These results show that DistilBERT is both highly accurate and balanced. It is reliable for identifying real threats while minimizing false alarms.

The CNN model also performed well, with 99.6% accuracy. It trained quickly and used less computing power. While its 99.8% precision indicates high confidence in positive results, its recall of 99.1% was a bit lower than that of DistilBERT. This means CNN may miss some complex attacks. Therefore, CNN is a good choice for situations where speed is crucial, but it may not handle advanced attacks as effectively as DistilBERT.

Turning to the BiLSTM model, it achieved 99.3% accuracy and a recall of 99.2%, making it highly effective in detecting real attacks. However, its precision was slightly lower at 98.9%, and its MCC was 0.986, resulting in more false positives than DistilBERT. As a result, BiLSTM is suitable when missing an attack is a greater concern than handling additional false alarms, though it may require more manual review. Notably, transformer-based models offer greater resilience against obfuscated and polymorphic SQL injection attacks, albeit at the cost of higher computational demands. Meanwhile, CNN and BiLSTM remain practical options for environments where lightweight inference is necessary. Ultimately, selecting the appropriate model should strike a balance between security needs and operational constraints.

3.2.1 Implications

The findings have significant implications for database security strategies. Transformer-based architectures, for starters, can play an important role in safeguarding applications against emerging SQL injection attacks due to their contextual awareness of queries. This is especially true for current apps, as attackers commonly utilize encoded or camouflaged payloads. Second, the addition of DistilBERT to the SQL Security Advisor highlights the practical value of integrating high-accuracy categorization with real-time advising features like risk assessment, query rewriting, and OWASP/NIST compliance. This implies that deep learning can progress beyond detection and become an active component in safe software development lifecycles.

3.2.2 Research contribution

This study provides three major additions to the literature:

1. This is the first direct comparison of CNN, BiLSTM, and DistilBERT for SQL injection detection, conducted under identical preprocessing, training, and assessment settings.
2. Transformer-based models outperform traditional architectures in this area due to their greater contextual modeling capabilities, rather than relying on more data or larger models.
3. The article develops and assesses a hybrid advising system that combines deep learning detection,

rule-based interpretability, and automated remediation, bridging the gap between academic models and real-world applications.

3.2.3 Limitations

Despite its virtues, the study has certain drawbacks. First, although the dataset is balanced between benign and malicious queries, it may not encompass the entire range of SQL injection techniques employed in the field, particularly those deliberately created attacks. Second, DistilBERT demonstrated improved accuracy, but its higher computational cost may restrict real-time scalability in resource-constrained situations. Finally, the trials were conducted on a controlled dataset; deployment in real systems may reveal new issues, such as changing attack patterns and integration costs.

3.2.4 Suggestions

To build on this work, future studies should:

1. Future research should evaluate models on datasets with adversarial, zero-day, or polymorphic SQLi payloads to ensure resilience during distribution change.
2. Consider using model compression, quantization, or knowledge distillation to lower DistilBERT's inference time for resource-constrained deployments.
3. Integrate SQL Security Advisor with CI/CD pipelines, IDEs, or database gateways to assess influence on developer behavior and code security in real-world applications.
4. Apply the transformer-based technique to other injection types, such as NoSQL, LDAP, and command injection.
5. Conduct user studies to evaluate the advisory system's usability, trustworthiness, and effectiveness among developers and security teams.

4. CONCLUSION

This study demonstrates that transformer-based architectures, specifically DistilBERT, outperform CNN and BiLSTM models for detecting SQL injection. DistilBERT handles obfuscated and adversarial attacks that often elude existing systems. It achieves an accuracy of (99.8%) and recall (99.5%), while retaining interpretability through hybrid integration with rule-based checks. Previous deep learning systems were limited by the need for rigorous tokenization or context modeling. In contrast, DistilBERT captures semantic and syntactic subtleties, enabling a robust and practical solution that extends beyond detection into proactive defense.

Beyond technical performance, the study bridges research and practice. The SQL Security Advisor integration demonstrates how high-performance AI can be integrated into developer workflows. This reduces false alarms, accelerates repair, and brings security practices into query design. The study establishes a reproducible baseline for future research and provides a practical approach to current database security. It anchors complex NLP models in usability and explainability. Security experts and developers will note a shift: from reactive detection to intelligent, developer-centric defense. Transformers strengthen, not complicate, critical infrastructure protection.

5. ACKNOWLEDGEMENT

The authors acknowledge the creators of the publicly available SQL Injection Dataset and thank the developers of the Hugging Face Transformers library, DistilBERT model, TensorFlow, and PyTorch frameworks for enabling this research. Their open-source contributions greatly facilitated model implementation and evaluation.

6. AUTHOR CONTRIBUTION STATEMENT

Faisal Alghamdi conducted the study design, preprocessing, analysis, and interpretation of results under the supervision of Dr. Boulbaba Ben Ammar. The dataset used was publicly available; therefore, no new data collection was performed. Both authors reviewed and approved the final manuscript.

AUTHOR INFORMATION

Corresponding Authors

Faisal Alghamdi, Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
https://orcid.org/0009-0008-1097-4278
Email: ffalghamdi@hotmail.com

Authors

Boulbaba Ben Ammar, Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
https://orcid.org/0000-0002-5718-4723
Email: b.benammar@qu.edu.sa

REFFERENCE

Al-Mallah, R., & Quintero, A. (2025). *Adversarial Threats and Defense Mechanisms in Machine Learning-Based SQL Injection Detection: A Security Analysis*. Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ICNC64010.2025.10993834

Bhupathiraju, S. K. R. (2025). Key Features and Innovations in SQL Server 2025 : Advancing Performance , Security , and AI Integration. *International Journal on Science and Technology (IJSAT)*, *16*(1), 1–23. https://doi.org/10.71097/IJSAT.v16.i1.2493

Chakir, O., & Sadqi, Y. (2025). *Demystifying the Role*

of Publicly Available Up-to-Date Benchmark Intrusion Datasets : A Case Study of Web Security Demystifying the Role of Publicly Available Up-to-date Benchmark Intrusion Datasets : A Case Study of Web Security (Issue December 2024). River Publishers Series in Digital Security and Forensics. https://doi.org/10.1201/9788770047746-11

Chen, Y., Liang, G., & Wang, Q. (2025). Research on SQL Injection Detection Technology Based on Content Matching. *Computers, Materials & Continua*, *84*(1), 1145–1167. https://doi.org/10.32604/cmc.2025.063319

Dritsas, E., & Trigka, M. (2025). Database Systems in the Big Data Era : Architectures , Performance , and Open Challenges. *IEEE Access*, *13*(May), 95068–95084. https://doi.org/10.1109/ACCESS.2025.3572059

Floris, G., Scano, C., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., & Biggio, B. (2025). ModSec-AdvLearn : Countering Adversarial SQL Injections with Robust Machine Learning. *IEEE Transactions on Information Forensics and Security*, *20*, 6693–6705. https://doi.org/10.1109/TIFS.2025.3583234

Gomes, D., Felix, E., Aires, F., & Vieira, M. (2025). *Static Code Analysis for IoT Security : A Systematic* (Vol. 58, Issue 3). https://doi.org/10.1145/3745019

Gupta, P., Nguyen, T. N., Gonzalez, C., & Woolley, A. W. (2025). Fostering collective intelligence in human–AI collaboration: laying the groundwork for COHUMAIN. *Topics in Cognitive Science*, *17*(2), 189–216. https://doi.org/10.1111/tops.12679

Keshireddy, S. R. (2025). Reinforcement Learning Based Optimization of Query Execution Plans in Reinforcement Learning Based Optimization of Query Execution Plans in Distributed Databases. *Research Briefs on Information and Communication Technology Evolution*, *11*(03), 42–61. https://doi.org/10.69978/rebicte.v11i.211

Liu, Y., & Dai, Y. (2024). Deep Learning in Cybersecurity : A Hybrid BERT – LSTM Network for SQL Injection Attack Detection. *IET Information Security*, *1*, 1–16. https://doi.org/10.1049/2024/5565950

Lubis, J. H., Handayani, S., Mawengkang, H., & Napitupulu, F. M. A. (2025). Performance Optimization of ERD Designs Using Cost-Based Optimization for Large-Scale Query Processing. *Jurnal Teknik Informatika (JUTIF)*, *6*(3), 1457–1467. https://doi.org/10.52436/1.jutif.2025.6.3.4523

Lyu, C. (2025). *Intelligent cost-performance optimization for big data analytics systems*. https://scholar.google.com/scholar?hl=id&as_sdt=0%2C5&q=Intelligent+Cost-Performance+Optimization+for+Big+Data+Analytics+Systems&btnG=

Mahin, M. T. (2025). *Efficient Cardinality Estimation and Query Processing for Large-Scale Databases*. https://scholar.google.com/scholar?hl=id&as_sdt=0%2C5&q=Efficient+Cardinality+Estimation+and+Query+Processing+for+Large-Scale+Databases&btnG=

Mariettou, S., Koutsojannis, C., & Triantafyllou, V. (2025). A Secure Prescription System with Machine Learning for SQL Injection. *Computer Networks and Communications*, *3*(2), 59–72. https://doi.org/10.37256/cnc.3220257145

Muduli, D., Shookdeb, S., Zamani, A. B. U. T., & Saxena, S. (2024). SIDNet : A SQL Injection Detection Network for Enhancing Cybersecurity. *IEEE Access*, *99*, 1–17. https://doi.org/10.1109/ACCESS.2024.3502293

OWASP Foundation. (2025). SQL Injection. https://owasp.org/www-community/attacks/SQL_Injection

Sajid576. (2021). SQL Injection Dataset. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT , a distilled version of BERT : smaller , faster , cheaper and lighter. *ArXiv*, 2–6. https://doi.org/10.48550/arXiv.1910.01108

Shaik, A. R., & Manoharan, A. (2025). Hybrid Convolutional Spinal Zeiler and Fergus Network-based Attack Prevention with the Tuning of Firewall in IoT. *Engineering Research Express*, *7*(3), 1–24. https://doi.org/10.1088/2631-8695/adf942

Souza, M. S., Estadual, U., Ribeiro, S. E. S. B., Estadual, U., Lima, V. C., Estadual, U., Cardoso, F. J., Estadual, U., Gomes, R. L., & Estadual, U. (2024). Combining Regular Expressions and Machine Learning for SQL Injection Detection in Urban Computing. *Journal of Internet Services and Applications*, *15*(1), 103–111. https://doi.org/10.5753/jisa.2024.3799

Swaroop, A. T. (2025). The Transformative Impact Of Artificial Intelligence On Professional Software Development : A Comprehensive Analysis. *INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)*, *13*(8), b74–b90. https://doi.org/10.56975/ijcrt.v13i8.292072

Takyi, K., Gyening, R. M. O. M., Kobinnah, M., & Boateng, M. A. (2025). Enhancing SQL Injection Detection with Long Short- Term Memory Networks in Deep Learning. *International Journal of Open Information Technologies*, *13*(1), 7–13. https://scholar.google.com/scholar?hl=id&as_sdt=0%2C5&q=Enhancing+SQL+injection+detection+with+long+short+term+memory+networks+i

n+deep+learning.&btnG=

Vangapandu, R., Sri, K. K., Sukeerthana, N., & Meghana, P. (2025). SQL Injection Detection via Graph Neural Networks and Query Dependency Graphs. *Macaw International Journal of Advanced Research in Computer Science and Engineering*, *11*(1), 102–113. https://doi.org/10.70162/mijarcse/2025/v11/i1/v11i110

Yetunde, C., Samson, D., Akinpelu, A., Olajuwon, E., Sunday, A., & Ajagbe, A. (2025). Malicious Query Recognition Using Chosen Machine Learning Techniques. *SN Computer Science*. https://doi.org/10.1007/s42979-025-03745-4

Yi, Z., Ives, Z. G., & Marcus, R. (2025). Low Rank Learning for Offline Query Optimization. *Proceedings of the ACM on Management of Data*, *3*(3), 1–26. https://doi.org/10.1145/3725412

Zivkovic, M., & Jovanovic, L. (2024). Optimizing SQL injection detection using BERT encoding and AdaBoost Classification Optimizing SQL injection detection using BERT encoding and AdaBoost Classification. *2nd International Conference on Innovation in Information Technology and Business (ICIITB 2024)*, 137–154. https://doi.org/10.2991/978-94-6463-482-2