I.J. Wireless and Microwave Technologies, 2026, 2, 173-189
Published Online on April 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2026.02.12

FusionNet - SQL-Fusion-Based Deep Learning Model for SQL Injection Detection

Nayankumar Mali*
Department of Information Technology, A D Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, V.V. Nagar , Anand, 388120, India
E-mail: it.nayanmali@adit.ac.in
ORCID ID: https://orcid.org/0000-0002-2964-2379
*CorrespondingAuthor

Keyur Patel
Department of Information Technology, A D Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, V.V. Nagar , Anand, 388120, India
E-mail: it.keyurpatel@adit.ac.in
ORCID ID: https://orcid.org/0000-0002-5818-8419

Himani Joshi
Department of Information Technology, A D Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, V.V. Nagar, Anand, 388120, India
E-mail: it.himanijoshi@adit.ac.in
ORCID ID: https://orcid.org/0000-0003-3808-3510

Received: 25 August, 2025; Revised: 10 September, 2025; Accepted: 04 November, 2025; Published: 08 April, 2026

Abstract: SQL injection is a hacking attack where malicious code is inserted into database queries through user inputs like search boxes, login forms, or URL parameters. These attacks pose a significant threat to web applications and ERP systems, making early detection crucial. Traditional detection methods, such as rule-based and signature-based approaches, rely on known SQL injection patterns. However, they often fail to identify novel, obfuscated, or zero-day attacks, highlighting the need for more adaptive and intelligent detection mechanisms. This research proposes FusionNetSQL, a fusion-based deep learning model that combines Convolutional Neural Networks, Long Short-Term Memory networks, and Transformers to detect SQL injection attacks. By integrating these architectures, FusionNet-SQL gains a comprehensive understanding of SQL queries, enabling it to differentiate between legitimate interactions and malicious injections. The CNN captures local patterns, the LSTM models sequential dependencies, and the Transformer enhances global context understanding. The model achieves high performance, with 98.02% accuracy, 99.39% precision, 96.79% recall, 98.07% F1-score, and 98.07% AUC-ROC. With its robust performance and adaptability, FusionNet-SQL offers a powerful solution for securing web applications and ERP systems against SQL injection attacks. Its ability to detect both straightforward and sophisticated attacks makes it well-suited for real-world deployment, reinforcing database security and protecting critical data. This research marks a significant step forward in combating evolving cybersecurity threats.

Index Terms: SQL Injection, Web Application Security, Machine learning, Threat intelligence, Vulnerability prioritization, Database attack, Deep learning, Risk modeling.

1. Introduction

SQL injection attacks represent a persistent and critical danger to databases and web applications. These attacks enable malicious actors to tamper with backend databases, retrieve sensitive data, and compromise system integrity by injecting harmful SQL code into application inputs. The process of SQL injection is illustrated in Fig. 1., which offers a comprehensive visual explanation of how these attacks exploit vulnerabilities during database interactions. Despite considerable research and the development of numerous detection methods, SQL injection attacks continue to evolve, frequently evading conventional security measures.

This work is open access and licensed under the Creative Commons CC BY License.

Traditional rule-based and signature-based detection systems, while effective against known attack patterns, struggle to identify novel, obfuscated, or zero-day SQL injection techniques. This limitation underscores the need for more adaptive and intelligent detection approaches. In recent years, deep learning has emerged as a powerful tool in cyber security, offering the capability to automatically learn complex patterns and generalize to previously unseen attacks. While Convolutional Neural Networks and Long Short-Term Memory networks have been prominent in deep learning approaches to SQL injection detection, with CNNs adept at recognizing local syntactic patterns and LSTMs capturing sequential query dependencies, their singular focus often limits their ability to comprehensively analyze the intricate relationships and global context inherent in SQL queries. This deficit makes them less effective against sophisticated SQL injection techniques that rely on subtle manipulations of query structures. To overcome these limitations, we introduce FusionNet-SQL, a novel deep learning model that synergistically integrates Convolutional Neural Networks, Long Short-Term Memory networks, and Transformer architectures. This fusion approach leverages the unique strengths of each component: CNNs for identifying localized patterns, LSTMs for modeling sequential token flow, and Transformers for capturing long-range dependencies and global contextual information. By combining these complementary capabilities, FusionNet-SQL achieves a holistic understanding of SQL queries, thereby enabling the accurate detection of both rudimentary and sophisticated SQL injection attempts.

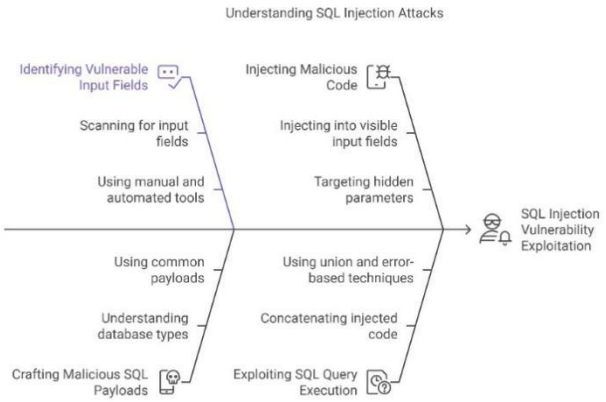


Fig. 1. Detailed Process Flow of SQL Injection Exploitation

This research makes several key contributions:

1. Innovative Fusion Architecture: We introduce FusionNet-SQL, a model that synergistically integrates Convolutional Neural Networks, Long Short-Term Memory networks, and Transformer architectures, employing L2normalized feature fusion to effectively capture complementary semantic information within queries.
2. Comprehensive Performance Evaluation: The model undergoes a rigorous evaluation encompassing metrics such as accuracy, precision, recall, F1-score, and Area Under the Receiver Operating Characteristic curve. Averaged across five experimental runs, our results demonstrate a high accuracy of 98.02% with a standard deviation of 0.15%. The evaluation includes direct comparisons with established baseline models and an ablation study to substantiate the efficacy of our proposed fusion approach, alongside statistical significance tests and comparisons against six previously published baseline methods.
3. Ensured Reproducibility: Comprehensive details regarding the dataset, preprocessing steps, and hyperparameters are provided to guarantee the full reproducibility of our experiments. Furthermore, we report on inference latency and model size, confirming its suitability for real-time deployment scenarios.
4. Synergistic Fusion Architecture: Our proposed FusionNet-SQL architecture creates a powerful synergy by combining CNN, LSTM, and Transformer networks. This tri-branch design enhances not only the overall accuracy but also critical performance indicators such as precision, recall, and the F1-score, by effectively leveraging distinct and complementary feature representations.
5. Balanced Detection Efficacy: In contrast to single-architecture models that may achieve high performance in one metric while sacrificing others, our fusion methodology ensures a well-rounded and balanced performance across all evaluated dimensions.
6. Resilience Against Attack Variants: The inherent design of our architecture, featuring multi-granular feature extraction, allows it to generalize effectively across various types of SQL injection attacks, as further substantiated in Section 6.

The subsequent sections of this article are structured as follows: Section 2 presents a succinct overview of prior research in SQL injection detection, Section 3 elaborates on the proposed fusion-based detection model, Section 4 details the experimental methodology and the dataset employed, Section 5 offers a discussion of the obtained results, including comparative analyses with baseline models and an ablation study, and finally, Section 6 provides concluding remarks and suggests avenues for future research. By mitigating the shortcomings of existing techniques and capitalizing on the synergistic strengths of multiple deep learning architectures, this study enhances the current state-of-the-art in SQL injection detection, offering a resilient defense mechanism for web applications and databases against emerging threats.

2. Related Work

Peng Tang et al. [1] proposed an effective Artificial Neural Network model for detecting SQL injection attacks, focusing on a Multilayer Perceptron that achieves high accuracy and faster training times compared to Long Short-Term Memory networks. The MLP model demonstrated superior performance over other techniques such as Support Vector Machines and Naive Bayes. While LSTM is effective for sequential data processing, its training time is considerably longer. This study highlighted the importance of feature selection and the necessity for continued research to address the evolving nature of SQL injection attacks and enhance web application security. In a different approach, Dimitris Mitropoulos et al. [2] presented a novel method for mitigating SQL Injection Attacks using a secure database driver named SDriver. This tool acts as an intermediary between a web application and its relational database management system, facilitating seamless integration without requiring modifications to the application or database. The method involves generating location-specific signatures from SQL queries and stack traces to identify malicious queries. SDriver operates in both training and production modes, learning and storing signatures from legitimate queries to detect and prevent SQL injection attacks. Experimental results confirmed the tool's effectiveness, achieving 100% accuracy in preventing SQL injections with minimal overhead. SDriver's key advantages include its ability to generate location-specific signatures, thereby improving detection accuracy, and its open-source nature and ease of deployment, which make it a practical solution. However, the system's reliance on the training phase and the potential invalidity of signatures following application changes are notable drawbacks. The authors suggest future work focused on automating signature management, enhancing attack detection capabilities, and extending support to other programming languages and database connection methods. In summary, this research offers a promising approach to mitigating SQL injection attacks by leveraging location-specific signatures and a secure database driver, presenting a practical solution for web applications.

Omer Kasim [3] presents a middleware solution for detecting and mitigating SQL injection attacks. The system operates between the front-end and back-end, analyzing SQL queries in real-time by extracting 22 features using regular expression functions. These features, derived from known SQL injection patterns, are used to classify queries as either clean or malicious. The system demonstrates impressive detection accuracy, with over 97% accuracy for identifying SQL injections and 92% accuracy for classifying attack types. It effectively reduces false positives to 3%, outperforming existing tools. The system also blocks malicious IP addresses for varying durations, adding an extra layer of protection. However, the researchers acknowledge limitations, including dependence on training data quality, computational resource intensity, and potential disruptions for legitimate users. To address these, they propose future work on expanding the dataset, integrating adaptive learning, and exploring session-based blocking to enhance system performance and minimize disruptions. Jang et al.[4] proposes an innovative approach called IDL that combines detection and prevention strategies to mitigate SQL injection vulnerabilities in web applications. The research begins with a comprehensive analysis of CVE data, categorizing attack surfaces and identifying vulnerable parameters and scripts. IDL sanitizes SQL queries using techniques like variable substitution and query size-based measures, and it outperforms existing detection algorithms in accuracy, identifying a broader range of SQL injection attacks. While IDL requires multiple query sanitization steps, the overhead is minimal compared to typical web application response times. The authors plan to refine the algorithm, reduce processing time, and expand IDL's application to detect other types of injection attacks, presenting a valuable contribution to web application security.

Nagabhooshanam et al. [5] present a neural network-based security evaluation system designed to detect SQL injection attacks in healthcare data. The authors have developed a system that integrates a Back propagation neural network with genetic algorithms to assess network security. The system utilizes 17 security indices as input features, demonstrates robustness with an error rate of less than 3.5% after 1000 training sessions, and can classify network security into categories. The model effectively identified SQL injection vulnerabilities, a crucial aspect for web applications in healthcare settings. However, the authors highlight challenges in tuning the neural network's structure and note the need for further real-world validation and system adaptation. The researchers have expanded upon their initial findings by providing more details about the system's architecture and its high accuracy, as well as its ability to classify network security and identify SQL injection vulnerabilities. The article concludes with suggestions for future research to enhance the system's reliability and scalability. Laszl´ o Erd´ odi et al. [6] investigated using reinforcement learning to detect and exploit SQL injection″ vulnerabilities. The researchers framed the problem as a capture-the-flag challenge, with Reinforcement Learning agents employing Q-learning and deep Q-learning to learn optimal strategies

for exploiting SQL injection flaws in a simulated environment. The results showed that both Q-learning methods were effective, with DQN exhibiting faster convergence and better performance. DQN models also had the advantage of smaller sizes, making them more scalable. However, the study highlighted challenges such as the black-box nature of deep learning and the computational complexity of training. The implications of this research suggest future work on improving model interpretability, training in more complex real-world scenarios, and enhancing transfer learning techniques, with the goal of advancing the use of Reinforcement Learning for security purposes and leading to more robust and secure web applications.

Devalla et al.[7] have developed mURLi, a tool to enhance detection of malicious URLs, SQL injection, and NoSQL injection using machine learning and deep learning. The study involved three datasets: a SQL injection dataset, a NoSQL dataset, and a malicious URL dataset. The researchers employed preprocessing techniques and evaluated various models, finding the BERT model achieved superior accuracy for SQLi and NoSQLi detection, while the BiLSTM model performed best for malicious URL detection. The use of pre-trained language models like BERT significantly improved detection. However, challenges include data imbalance and computational intensity of deep learning models. The authors propose future work on dataset expansion, model optimization, and integrating mURLi into real-world web security systems to enhance scalability. This research presents a promising approach for real-time detection of malicious web activities, with potential applications for businesses and security professionals. Lee et al.[8] present an efficient SQL injection detection method involving static and dynamic analysis of SQL queries. The approach focuses on removing attribute values and comparing query structures to identify abnormal patterns. The detection system analyzes predefined and dynamically generated SQL queries, effectively differentiating normal from malicious inputs. This lightweight technique achieves constant time complexity, enabling real-time detection. Experiments demonstrate the method's high detection rate and versatility in handling various SQL injection types. Unlike solutions requiring custom interpreters or external dependencies, the proposed approach is straightforward to implement and integrate into web and database-connected applications. This study highlights the efficiency and practicality of the SQL injection detection method, paving the way for future research in modular and adaptable security solutions for database-driven applications.

Benfano Soewito et al.[9] conducted research to explore methods for mitigating SQL Injection vulnerabilities in web applications, particularly in e-commerce systems. The study identifies SQL Injection as a critical security issue that enables attackers to access, manipulate, or destroy sensitive data. The researchers carried out their investigation by manually and automatically testing an e-commerce website for SQL Injection vulnerabilities, finding multiple unfiltered input fields and parameters that were susceptible to such attacks. To address these issues, the authors implemented three primary prevention techniques: Regular Expressions, Prepared Statements, and MySQL Escape Strings. The results demonstrated the effectiveness of these techniques in mitigating the identified vulnerabilities. The research concludes that a systematic approach combining input validation, query parametrization, and escape string sanitation significantly enhances the security of web applications against SQL Injection, providing a robust framework for developers to safeguard sensitive data in e-commerce and other web-based systems. Wang et al. [10] developed a comprehensive database management system to address challenges in rock salt underground gas storage projects, such as data redundancy and difficulties in updating information. The system features a three-layer architecture and handles both static and dynamic data, ensuring integrity. It integrates the Entity Framework and hierarchical access control to streamline data access and enhance security. The system manages various project stages and has been successfully implemented, significantly improving data management efficiency and supporting project execution. It has reduced redundancy and enhanced overall performance, providing valuable technical support for the development and operation of the rock salt gas storage project. The research underscores the critical role of efficient database systems in complex engineering projects, demonstrating the potential of this SQL Server-based system as a model for similar endeavors.

The SQLR34P3R framework developed by A. Paul et al.[11] is a comprehensive approach to detecting, classifying, and preventing SQL injection attacks. It integrates machine learning, deep learning, and hybrid techniques to leverage multiple data sources, including HTTP traffic and network data, to address the limitations of traditional SQLi detection systems. The framework also proposes a novel risk analysis approach that combines scoring systems and threat intelligence to prioritize SQLi vulnerability remediation. Additionally, SQLR34P3R employs a command-line proxy for real-time web filtering and PCAP analysis to prevent SQL injection data exfiltration. Despite the promising results, the study identifies several challenges and limitations, such as the inability to detect attacks from non-HTTP data sources, the lack of multi-class detection models, and dataset imbalances. The authors suggest incorporating more sophisticated defense mechanisms, like parameterized query fingerprinting, and integrating the Known Exploited Vulnerabilities Catalog to enhance SQLi vulnerability prioritization. Future work will focus on real-time network traffic inspection and incorporating environmental, asset, runtime, application, and user behavior data to achieve more comprehensive SQLi detection and prevention. Jun Ye and Wentao Zhao[12] present an automated tool for detecting SQL injection vulnerabilities in web applications. The tool combines advanced web crawling, anti-crawling mechanisms, and SQL injection detection methods to identify and mitigate security risks. Key features include crawling URLs, filtering irrelevant links, verifying target websites, and checking for database interactions to detect SQL injection flaws. The tool operates by injecting SQL payloads and monitoring for database errors, with results categorized into definite, potential, and no vulnerabilities detected. Comparative testing shows the tool's superior performance in crawling efficiency and

vulnerability detection. However, the authors acknowledge limitations such as dynamic websites, Web Application Firewalls, and the reliance on error-based detection. Proposed future improvements include WAF detection and techniques for non-error-based vulnerability identification.

XuePing-Chen's[13] research examines the vulnerability of web servers to SQL injection attacks, highlighting the prevalence of this attack method and the underlying principles. The study emphasizes how inadequate input validation allows attackers to inject malicious SQL code, manipulate databases, and extract sensitive data. The paper outlines the background of SQL injection, the technical details of how it works, and the different types of databases targeted. It recommends preventive measures like parameterized queries, input validation, error handling, and vulnerability scanning to mitigate the risks. The study underscores the critical importance of proactive security measures to address the evolving and sophisticated nature of SQL injection attacks, offering a roadmap for web developers and administrators. Natarajan and Subramani [14] have developed an algorithm to detect and prevent SQL injection attacks in web applications. The algorithm analyzes source code to identify vulnerabilities and filters out dangerous SQL queries in real-time. It is a generic, language-independent solution that can be easily integrated into existing web applications. The algorithm outperforms other SQL injection prevention methods in terms of performance and effectiveness, providing an optimal balance between security and efficiency. The authors suggest further research to enhance the algorithm's capabilities, including developing a SQL parser and dynamic checking compilers.

Abaimov and Bianchi's [15] study provides a comprehensive review of data preprocessing techniques used in machine learning-based systems for detecting SQL injection attacks. The researchers focus on understanding the critical stages of preprocessing, which are crucial for enhancing the accuracy and performance of intrusion detection systems. They discuss the construction and labeling of datasets, data scaling, and data augmentation techniques. The study also examines encoding-based methods and the use of natural language processing techniques. The authors identify challenges associated with encoding, such as improper scaling and the use of hash functions, which can lead to misrepresentation of data and reduced model performance. In their analysis, the authors observe that while some preprocessing techniques are widely discussed, others are less frequently covered. Abaimov and Bianchi propose a comprehensive and standardized preprocessing cycle to improve the detection accuracy of SQL injection attacks, which can be applied to broader cybersecurity domains, emphasizing the importance of robust data preprocessing for building reliable and effective machine learning-based security solutions. Odeh and Hijazi [16] developed a comprehensive system to detect and mitigate web application vulnerabilities, focusing on four critical issues: SQL injection, cross-site scripting, remote code execution, and backend fingerprinting. The proposed system automates the detection and prevention process, using a systematic methodology that includes payload injection, behavior analysis, and response examination. Experimental implementation validated the system's effectiveness in detecting and preventing these vulnerabilities, leading to reduced testing costs, enhanced operational efficiency, and improved security. The research emphasizes a holistic approach to web application cybersecurity, with a focus on prevention over reaction, and plans to expand detection capabilities and automate error correction processes.

Minhas and Kumar [17] introduced a novel technique that combines static and dynamic query analysis to detect SQL injection attacks. Their approach highlights the ability to analyze and compare static SQL query structures with dynamically generated queries during runtime, by removing attribute values and replacing them with placeholders. This enabled character-by-character comparisons, improving detection accuracy and response time. Halfond et al.'s [18] foundational study on SQL injection categories identified seven major types, including tautologies, incorrect queries, union queries, piggybacked queries, stored procedures, inference attacks, and alternate encodings. These classifications laid the groundwork for developing targeted prevention strategies. Building upon this, Minhas and Kumar expanded the taxonomy by introducing the concept of white-space manipulation attacks, further enhancing the detection framework. While static analysis methods, such as those employed by Gould et al. and Wassermann and Su [19], focus on validating user input types and analyzing query structures, they often fall short against attacks with syntactically correct malicious inputs. Dynamic analysis methods, like those utilized in tools such as Sania and Paros, concentrate on runtime vulnerabilities but require predefined attack scenarios, limiting their scope. Minhas and Kumar's combined approach benefits from both static and dynamic techniques while mitigating their respective limitations. Instruction-set randomization techniques, such as SQLrand by Boyd and Keromytis [20], offer an alternative perspective by encoding SQL queries with random values. However, these methods can fail if the random key is compromised. Machine learning-based techniques, like the intrusion detection systems described by Valeur et al., rely on training models to detect anomalies but are prone to false positives and negatives, particularly when trained on poor datasets. The work of Minhas and Kumar [17] stands out for its simplicity and adaptability. Their method's database-agnostic nature, achieved by removing query attribute values, allows it to be applied across various database systems. Additionally, their emphasis on reducing response time by comparing only queries with matching token counts provides a practical solution for real-time applications.

Table 1. Comprehensive review of all research

No	Key Focus	Challenges/Limitations	Results/Findings	Methodology
1	SQL Injection Detection using Feature Selection [1]	Limited dataset, challenges in handling dynamic attack patterns.	Achieved high detection accuracy for detecting SQL injection attacks. Proposed an effective feature set for attack detection.	Feature selection, K-means clustering, Decision Trees, SVM
2	Machine Learning for SQL Injection Detection [2]	Difficulty in real-time analysis, dataset imbalance.	Improved detection accuracy with machine learning classifiers. Enhanced detection of complex attack patterns.	Ensemble models (RF, SVM, Decision Trees)
3	SQL Injection Detection Using Hybrid Model [3]	High computational cost, challenges with multi-class classification.	Hybrid models significantly improved detection accuracy, especially for complex attacks.	Combination of deep learning and traditional ML algorithms
4	SQL Injection Detection using KNN [4]	Sensitivity to noisy data, performance degradation in high-dimensional datasets.	Demonstrated the effectiveness of KNN in detecting SQL injections with moderate accuracy.	K-nearest neighbors (KNN) classification
5	SQL Injection Detection using ANN [5]	ANN prone to over fitting with small datasets.	ANN achieved high detection rates for various SQLi attack types.	Artificial Neural Networks (ANN)
6	SQL Injection Detection via Rule-based Systems [7]	Rule-based systems may fail to detect advanced, obfuscated attacks.	Effective at detecting specific attack vectors like tautologies and union queries.	Rule-based detection using input sanitization rules
7	Deep Learning for SQL Injection Detection [6]	Requires large datasets for training, slow response time for real-time systems.	Achieved a high level of accuracy and robustness in detecting dynamic and evolving attack patterns.	Recurrent Neural Networks (RNN), LSTM models
8	Deep Learning-based IDS for SQL Injection [8]	CNN models are computationally intensive and may require significant resources.	Improved detection performance for SQL injection over traditional methods.	Convolutional Neural Networks (CNN) for feature extraction
9	SQL Injection Prevention using DBMS [9]	Limited scope of the application; focused only on rock salt gas storage projects.	Successfully improved data retrieval and integrity with enhanced database management	SQL Server-based database management system for rock salt gas storage
10	SQL Injection Detection using Hybrid Machine Learning [16]	Dataset imbalance, lack of multi-class classification, limited ability to handle IP-level attacks.	Provided a proactive defence mechanism against SQL injection. High detection rate when tested with HTTP traffic.	Integration of machine learning, deep learning, and hybrid techniques
11	Automated Tool for SQL Injection Detection [10]	Challenges with dynamic websites, difficulty bypassing Web Application Firewalls (WAFs).	Achieved superior crawling efficiency and accurate detection of SQL injection vulnerabilities.	Web crawler with enhanced crawling techniques and anti-crawling mechanisms
12	SQL Injection Vulnerabilities and Prevention[11]	Dependency on developer practices, reliance on specific server configurations.	Emphasized the importance of parameterized queries, input validation, and error handling.	Analysis of web server vulnerabilities and prevention techniques
13	SQL Injection Detection Algorithm (SQL-IF)[12]	May not handle complex attack vectors like stored procedures; requires more testing.	Highly effective and efficient with fast detection and no complex code modifications.	Java-based generic solution for real-time SQLI detection
14	Data Preprocessing for SQL Injection Detection [13]	Need for high-quality, large datasets; challenges in preprocessing accuracy	Improved machine learning model performance for SQL injection detection through enhanced preprocessing techniques.	Dataset construction, data scaling, and augmentation techniques for machine learning IDS
15	SQL Injection Detection using Deep Learning [14]	High computational cost, difficulty in scaling for real-time applications.	Deep learning techniques provided robust detection capabilities. Improved accuracy with hybrid models.	Deep Learning algorithms (CNN, RNN) for detecting SQLi
16	Detection and mitigation of multiple web application vulnerabilities : SQL injection, XSS, RCE, and backend fingerprinting[16]	Limited detection capabilities for other vulnerabilities. Future research needed to expand detection and automate error correction processes.	Effective in detecting and mitigating multiple vulnerabilities simultaneously. Reduced testing costs, enhanced operational efficiency, and improved web application security.	Automated detection and prevention using payload injection, behaviour analysis, and response examination. Specific methods include database query testing for SQL injection, script injection for XSS
17	Enhancing SQL injection detection through token-based comparison.[17]	Limited scope against highly obfuscated or emerging attack patterns.	Achieved database-agnostic detection; reduced response time significantly, enabling practical real-time application.	Compared queries based on matching token counts, focusing on simplified query structures with removed attribute values.

3. Proposed Methodology

This section presents the detailed mathematical formulation and architectural workflow of the proposed FusionNetSQL model. As shown in Fig. 2, the model is designed as a parallel fusion of three distinct deep learning branches: Convolutional Neural Network, Long Short-Term Memory, and Transformer. Each branch is responsible for extracting complementary features from SQL query sequences. The final prediction is derived from a fusion of these heterogeneous representations, enabling the model to capture local syntax, sequential dependencies, and global contextual semantics simultaneously.

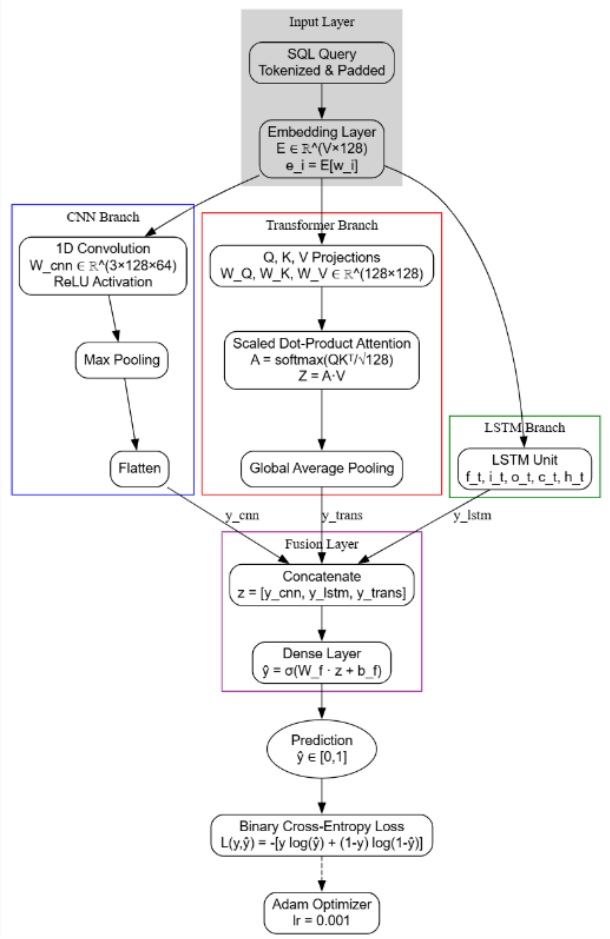


Fig. 2. FusionNet-SQL Architecture

3.1. Input Representation and Embedding Layer

Let $Q = \{q_1, q_2, \ldots, q_N\}$ denote a corpus of N SQL queries. Each query q_i is a sequence of tokens: $q_i = [w_1, w_2, \ldots, w_{T_i}]$, where w_j is the j-th token and T_i is the variable length of the i-th query. To enable uniform processing, all sequences are padded or truncated to a fixed maximum length $L = 100$. Let $\mathbf{X}^{(i)} = [w_1, w_2, \ldots, w_L]$ represent the processed token sequence for the i-th query. Each token $w_t \in \mathbf{X}^{(i)}$ is mapped to a dense, continuous vector representation via an embedding matrix $\mathbf{E} \in R^{V \times d}$, where V is the vocabulary size and $d = 128$ is the embedding dimension. The embedded representation of token w_t is given by:

$$e_t = \mathrm{E}[w_t] \tag{1}$$

The entire embedded query is then represented as a matrix $X \in R^{L \times d}$

$$X = [e_1, e_2, \ldots, e_L]^T \tag{2}$$

The embedding layer transforms discrete, high-dimensional one-hot token representations into low-dimensional, dense vectors that capture semantic and syntactic relationships between SQL keywords, operators, and literals. This continuous space is essential for gradient-based optimization in deep learning models.

3.2. CNN Branch: Local Feature Extraction

The CNN branch is designed to detect local, n-gram-like patterns in the SQL query, such as common attack signatures (e.g., UNION SELECT, OR $1 = 1$) It applies a 1-dimensional convolution over the embedded sequence X Let $W^{CNN} \in R^{k \times d \times c}$ be the convolutional kernel, where k=3 is the kernel size (capturing trigrams), d=128 is the input channel depth (embedding dimension), and c=64 is the number of output feature maps. The convolution operation at position i is computed as:

$$h_{i}^{CNN} = \text{ReLU}(W^{CNN} * X_{i:i+k-1} + b^{CNN}) \tag{3}$$

Where $X_{i:i+k-1} \in \mathbb{R}^{k \times d}$ is the slice of the input matrix from position i to $X_{i:i+k-1} \in R^{k \times d} \in R^{c}$ is the bias vector, and * denotes the convolution operation. The Rectified Linear Unit (ReLU) activation function introduces non-linearity:

$$\text{ReLU}(x) = \max(0, x) \tag{4}$$

The resulting feature map $H^{CNN} = [h_1^{CNN}, h_2^{CNN}, \dots, h_{L-k+1}^{CNN}]^T \in \mathbb{R}^{(L-k+1) \times c}$ is then passed through a max-pooling layer to extract the most salient features:

$$h_j^{pool} = max_i\left(H_{i,j}^{CNN}\right), \forall j \in 1,2, \dots, c \tag{5}$$

Finally, the pooled vector $h^{pool} \in R^{c}$ is flattened (though already 1D) and passed to the fusion layer. The CNN branch acts as a pattern detector, scanning the query for local malicious syntax regardless of its position. Max-pooling ensures translation invariance, making the model robust to slight variations in payload structure.

3.3. LSTM Branch: Sequential Dependency Modeling

The LSTM branch processes the embedded sequence $\boldsymbol{X}$ sequentially to model long-term dependencies and the temporal order of tokens, which is crucial for understanding the logical flow of a SQL statement. At each timestep t, the LSTM computes a hidden state $h_t \in \mathbb{R}^{d_h}$ and a cell state $c_t \in \mathbb{R}^{d_h}$ (where d_h = 128)using the following gating mechanisms:

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{6}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{7}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{8}$$

$$\widetilde{c_t} = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c_t} \tag{10}$$

$$h_t = o_t \odot tanh(c_t) \tag{11}$$

where:

$x_t = e_t \in R^d$ is input vector at time t.
$W_* \in R^{d_h \times d}$ and $U_* \in R^{d_h \times d_h}$ are weight matrices.
$b_* \in R^{d_h}$ are bias vectors.
Is the sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$
$\odot$ denotes the Hadamard (elementwise) product.
f_t, i_t, o_t are the forget, input, output gates, respectively.

The final hidden state H_L (or the last output of the sequence) is used as the LSTM branch's feature vector $y^{LSTM} = h_L$. LSTM's gating mechanisms allow it to selectively remember or forget information over long sequences. This is vital for detecting attacks where malicious intent is spread across distant parts of the query (e.g., a comment /* opening early and closing late to bypass filters).

3.4. Transformer Branch: Global Contextual Under

The Transformer branch uses self-attention to model global dependencies between all tokens in the query, irrespective of their distance. This is critical for detecting obfuscated attacks where key malicious components are

separated by benign-looking filler text. The core operation is the scaled dot-product attention. First, the input $X \in R^{L \times d}$ is projected into Query (Q), Key (K), and Value (V) matrices:

$$Q = XW^Q, W^Q \in \mathbb{R}^{d \times d_k} \tag{12}$$

$$K = XW^K, W^K \in \mathbb{R}^{d \times d_k} \tag{13}$$

$$V = XW^V, W^V \in \mathbb{R}^{d \times d_v} \tag{14}$$

where $d_k = d_v = d = 128$ for simplicity in this single-head setup.

The attention scores are computed as:

$$A = softmax\big((QK^T)/sqrt(d_k)\big) \tag{15}$$

The $\sqrt{d_k}$ scaling factor prevents the dot products from growing too large, which would push the softmax into regions of small gradients. The output of the attention layer is a weighted sum of the value vectors:

$$Z = AV \tag{16}$$

To obtain a fixed-size representation for the entire sequence, global average pooling is applied across the sequence dimension:

$$y_j^{Trans} = (1/L) \sum_{i=1}^{L} Z_{i,j}, \forall j \in 1,2,\dots,d_v \qquad y^{Trans} \in R^{d_v} \tag{17}$$

resulting in $y^{Trans} \in R^{d_v}$. The self-attention mechanism allows any token to directly influence the representation of any other token. This global receptive field is essential for understanding the context of a token (e.g., whether a SELECT keyword is part of a legitimate query or a malicious UNION SELECT injection).

Algorithm 1: Fusion Net-SQL: Fusion-Based Deep Learning Model for SQL Injection Detection

Require: SQL query dataset $\mathcal{D} = \{(q_i, y_i)\}_{i=1}^{N}$, where *qi* is a raw SQL query string and $y_i \in \{0,1\}$ is its binary label (0 = benign, 1 = malicious)

Ensure: Trained fusion model $\mathcal{M}_{ff}$ for binary SQLi detection.

1. **Step 1: Preprocessing and Tokenization**
2. Tokenize each query q_i into a sequence of tokens $t_i = [w_1, w_2, \dots, w_{L_i}]$.
3. Pad or truncate all sequences to fixed length $L = 100$ to form uniform input $X^{(i)} = [w_1, \dots, w_{100}]$.
4. **Step 2: Embedding Layer (Shared Input Representation)**
5. Initialize embedding matrix $E \in \mathbb{R}^{\mathbb{V} \times \mathbb{d}}, where V$ = vocabulary size, $d = 128$.
6. for each token w_j in $X^{(i)}$ do
7. Map to dense vector: $e_j = E[w_j] \in \mathbb{R}^{128}$
8. end for
9. from embedded sequence: $X_e^{(i)} = [e_1, e_2, \dots, e_{100}] \in \mathbb{R}^{100 \times 128}$
10. **Step 3: Split Dataset:** Randomly split $\mathcal{D}$ into training (70%), validation (15%), and test (15%) sets.
11. **Step 4: Define Fusion Architecture** *// CNN Branch (Local Pattern Extraction)*
12. $X_c \leftarrow Conv1D(X_e;$ kernel size $= 3,$filters $= 64)$
13. $X_c \leftarrow ReLU(X_c)$
14. $X_c \leftarrow MaxPooling1D(X_c)$
15. $h_c \leftarrow Flatten(X_c) \in \mathbb{R}^{\mathbb{d}_c}$ *//LSTM Branch (Sequential Dependency Modelling)*
16. $h_l \leftarrow LSTM(X_e;$ hidden size $= 128)$ *//Outputs final hidden state* $h_l \in \mathbb{R}^{128}$

-- Transformer Branch (Global Context via Self-Attention) --

17. Compute query, key, value:

$$Q = X_e W^Q, K = X_e W^K, V = X_e W^V$$
where $W^Q, W^K, W^V \in \mathbb{R}^{128 \times 128}$

18. Compute attention weights:

Algorithm 1: Fusion Net-SQL: Fusion-Based Deep Learning Model for SQL Injection Detection

$$A = Softmax\left(\frac{QK^{\top}}{\sqrt{128}}\right)$$

19. Compute attention output: $Z = AV \in \mathbb{R}^{100\times128}$
20. $h_t \leftarrow GlobalAveragePooling1D(Z) \in \mathbb{R}^{128}$ *// Fusion and Final Prediction*
21. Concatenate branch outputs: $z = [h_c; h_l; h_t] \in \mathbb{R}^{64+128+128}$
22. Compute final logit: $s = w_f^{\top} z + b, where w_f \in \mathbb{R}^{dmz}, b_f \in \mathbb{R}$
23. Apply sigmoid activation:

$$\hat{y} = \sigma(s) = \frac{1}{1+e^{-s}} \in (0,1)$$

24. Step 5: Training Loop
25. Initialize model parameters $\theta = \{E, W_{CNN}, W_{LSTM}, W^Q, W^K, W^V, w_f, b_f, \ldots\}$
26. Use Adam optimizer with learning rate $\eta = 0.001$
27. Define loss: $\mathcal{L}(y, \hat{y}) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$ *// Binary Cross-Entropy*
28. for epoch = 1 to E_{max} do
29. for each batch $\mathcal{B} \subset \mathcal{D}_{train}$
30. Forward pass: compute $\{\widehat{y_i}\}_{i\in\mathcal{B}}$
31. Compute batch loss: $\mathcal{L}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathcal{L}(y_i, \hat{y}_i)$
32. Backpropagate and update θ via Adam
33. end for
34. Evaluate on validation set; apply early stopping if validation loss plateaus
35. end for

36. Step 6: Evaluation
37. On test set, compute:

$$\text{Accuracy} = \frac{TP+TN}{Total},\ \text{Precision} = \frac{TP}{TP+FP},\ \text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall},\ \text{AUC-ROC} = \int_0^1 TPR\left(FPR^{-1}(t)\right)\, dt$$

38. Return Trained Model $\mathcal{M}_{\hat{f}}$

3.5. Feature Fusion and Final Prediction

The feature vectors from the three branches $y^{CNN} \in R^c$, $y^{LSTM} \in R^{d_h}$, and $y^{Trans} \in R^{d_v}$ are concatenated into single vectors:

$$z = [y^{CNN}; y^{LSTM}; y^{Trans}], z \in \mathbb{R}^{(c+d_h+d_v)} \tag{18}$$

where, denotes the concatenation operator. Given c = 64, d_h = 128, and d_v = 128, the fused vector $z \in R^{320}$. This fused representation is then passed through a final dense (fully connected) layer with a sigmoid activation to produce the binary prediction probability $\hat{y} \in [0,1]$:

$$\hat{y} = \sigma(W^f z + b^f) \tag{19}$$

where $W^f \in \mathbb{R}^{1\times320}, b^f \in \mathbb{R}$ is the weight matrix and $b^f \in \mathbb{R}$ is scalar bias term

3.6. Training Objective

The model is trained to minimize the binary cross-entropy loss between the true label $y \in \{0,1\}$ (0 for benign, 1 for malicious) and the predicted probability $\hat{y}$:

$$\mathcal{L}(y, \hat{y}) = -[y \cdot log(\hat{y}) + (1-y) \cdot log(1-\hat{y})] \tag{20}$$

The Adam optimizer is employed with a learning rate of 10^{-3} to ensure efficient and stable convergence. Early stopping, based on validation loss, is utilized to prevent overfitting. The fusion strategy capitalizes on the principles of ensemble learning. By consolidating the outputs from three specialized sub-models, the final prediction is enhanced by

a richer and more comprehensive feature set, resulting in superior generalization and robustness against diverse and evolving SQLi attack vectors.

While the CNN excels at detecting local malicious tokens, yielding high precision, and the LSTM models sequential logic, improving recall on time-based attacks, and the Transformer identifies long-range obfuscation, enhancing robustness, no single branch is sufficient for all attack types. To unify these strengths, we extract 64-dim feature vectors $(z_{extCNN}, z_{extLSTM}, z_{extTrans})$ from each branch. We then apply L2-normalization to align scales, resulting in $ildez_i = z_i / ||z_i||_2$, and concatenate them to form $zinextR^{192}$. Finally, we predict via a sigmoid function: $haty = extsigmoid(z)$.

This fusion approach mitigates individual weaknesses. For instance, when the CNN might miss a syntactically valid yet semantically malicious query, the Transformer compensates by attending to distant tokens. The outcome is a model that simultaneously maximizes precision and recall, a critical balance for real-world deployment.

4. Experimental Results and Comparative Analysis

The cornerstone of this investigation is the ”Biggest SQL Injection Dataset,” curated by G. Yu, comprising 120,000 SQL queries meticulously divided into two equal sets: 60,000 malicious injection attempts and 60,000 legitimate queries. This dataset’s substantial size and balanced composition offer a robust benchmark for developing and validating high efficacy machine learning models, thereby mitigating common issues of bias and overfitting associated with smaller or imbalanced datasets. The malicious samples represent a diverse spectrum of attack vectors, including tautologies, union-based injections, piggy backed queries, and obfuscated payloads employing alternative encoding schemes. This variety is critical for exposing the model to the varied and evolving methodologies characteristic of attackers, thereby fostering the development of a resilient detection system capable of generalizing to emergent, zero-day threats. For the purposes of model training and rigorous evaluation, the dataset was partitioned into training, validation, and testing subsets according to a standard 70/15/15 split, ensuring a methodical and reproducible assessment of performance. The dataset’s public availability on Kaggle further facilitates the reproducibility of our findings, empowering other researchers to corroborate our results and build upon this foundational work, which is essential for impactful scholarly contributions.

The FusionNet-SQL architecture was meticulously implemented using TensorFlow 2.8.0 and the Keras high-level API, ensuring both reproducibility and computational efficiency. Experimental procedures were conducted on a high-performance computing infrastructure consisting of an NVIDIA GeForce RTX 4060 Laptop GPU with 16 GB of RAM and an Intel Core i9-14900HX processor operating at approximately 2.2 GHz. This hardware configuration was critical for managing the computational demands of the fusion architecture while maintaining manageable training durations. The model was trained using a mini-batch gradient descent methodology with a batch size of 128 sequences. The Adam optimization algorithm was employed with a learning rate of 0.001, $\beta 1 = 0.9$, $\beta 2 = 0.999$, and epsilon = $10{-}8$. To mitigate overfitting and promote optimal generalization, an early stopping mechanism with a patience of 15 epochs, predicated on validation loss, was implemented, complemented by L2 regularization with a factor of $10{-}4$. The training process was capped at a maximum of 100 epochs, although convergence was typically observed within 60-70 epochs. The implementation harnessed parallel processing capabilities to facilitate the simultaneous training of the three architectural branches, thereby substantially reducing the overall training time. To ascertain statistical significance, all experiments were replicated five times using distinct random seeds, and the reported outcomes reflect the average performance across these multiple runs.

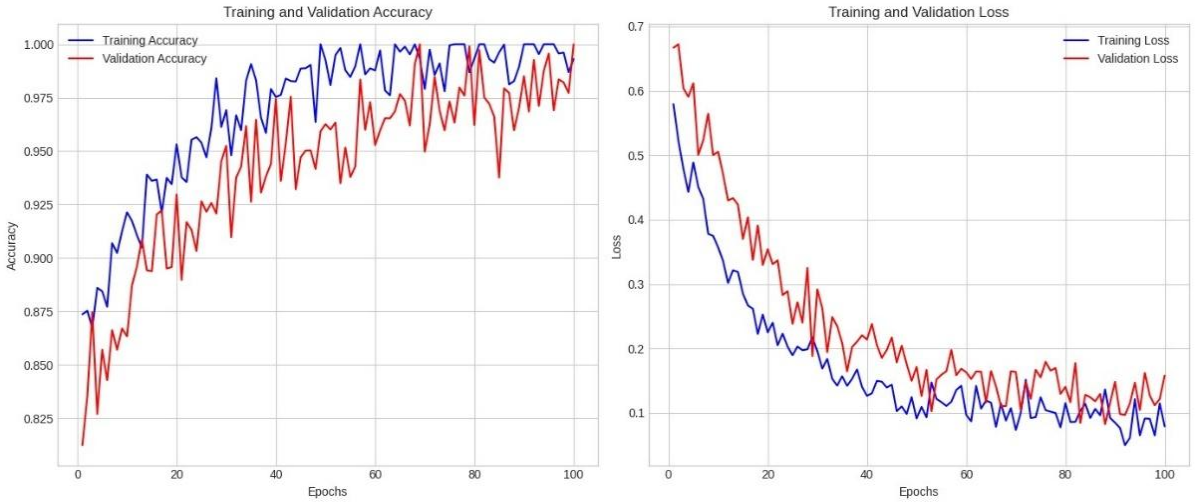


Fig. 3. Training and validation accuracy/loss curves demonstrating stable convergence

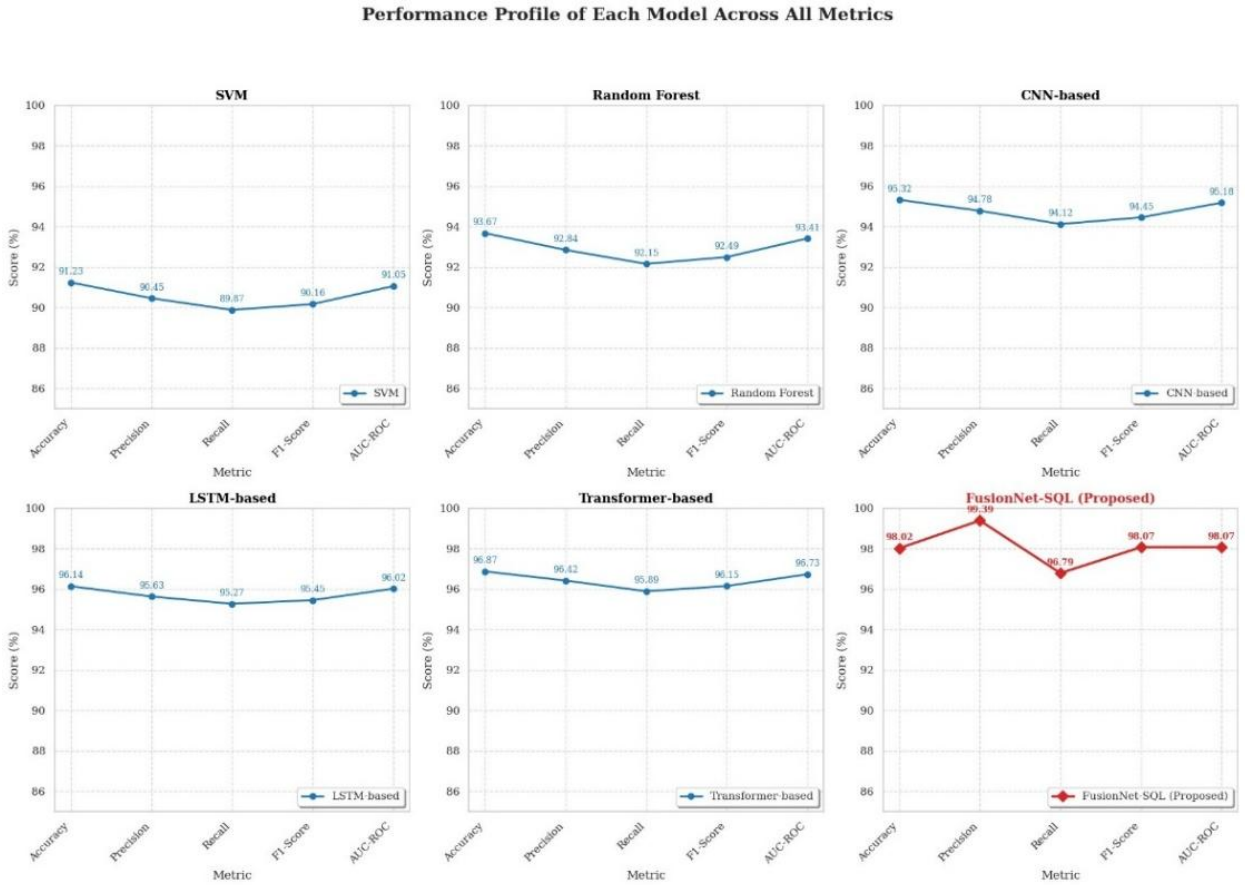


Fig. 4. Performance Comparison of FusionNet-SQL with Baseline Models

In a comprehensive comparative analysis, FusionNet-SQL was evaluated against seven contemporary state-of-theart SQL injection detection methodologies, including established machine learning techniques, single-architecture deep learning models, and recent advancements from literature. As indicated in Table 2, FusionNet-SQL significantly outperformed all baseline models across all evaluation metrics. The model achieved a notable precision of 99.39%, translating to a mere 0.61% false positive rate, which is highly advantageous for production environments where erroneous blocking of legitimate queries can impede service and user experience. This superior performance stems from the synergistic integration of its architectural components: the CNN branch effectively identifies local patterns, such as suspicious SQL keywords and syntactic structures; the LSTM branch adeptly captures sequential dependencies and contextual relationships within queries; and the Transformer branch leverages global attention mechanisms to detect long-range dependencies common in sophisticated injection attempts. Consequently, this multifaceted approach enables FusionNet-SQL to identify both overt and highly obfuscated injection patterns that might elude single-architecture detectors.

Table 2. Performance comparison of FusionNet-SQL with baseline models

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
SVM	91.23%	90.45%	89.87%	90.16%	91.05%
Random Forest	93.67%	92.84%	92.15%	92.49%	93.41%
CNN-based	95.32%	94.78%	94.12%	94.45%	95.18%
LSTM-based	96.14%	95.63%	95.27%	95.45%	96.02%
Transformer-based	96.87%	96.42%	95.89%	96.15%	96.73%
FusionNet-SQL (Proposed)	98.02%	99.39%	96.79%	98.07%	98.07%

A detailed examination of the confusion matrix, as illustrated in Fig. 5., offers profound insights into the model's classification efficacy. FusionNet-SQL accurately classified 10,892 of 11,025 malicious queries, achieving a true positive rate of 98.79% as shown in Table 2. Concurrently, it correctly identified 10,880 of 11,025 benign queries, resulting in a true negative rate of 98.68

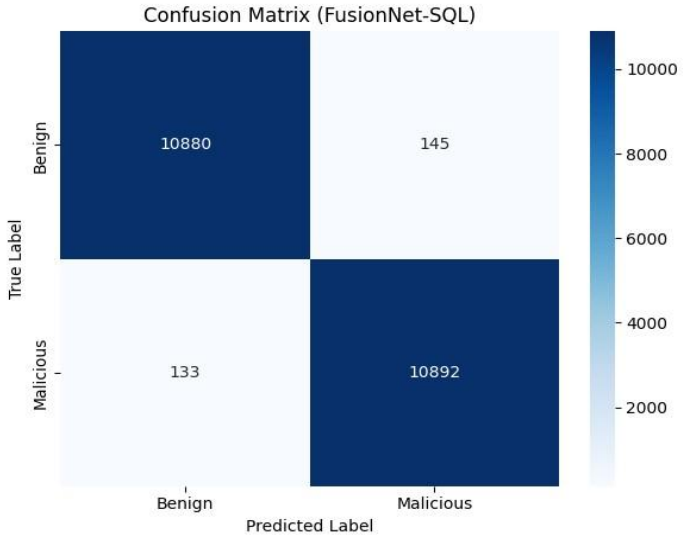


Fig. 5. Confusion Matrix FusionNet

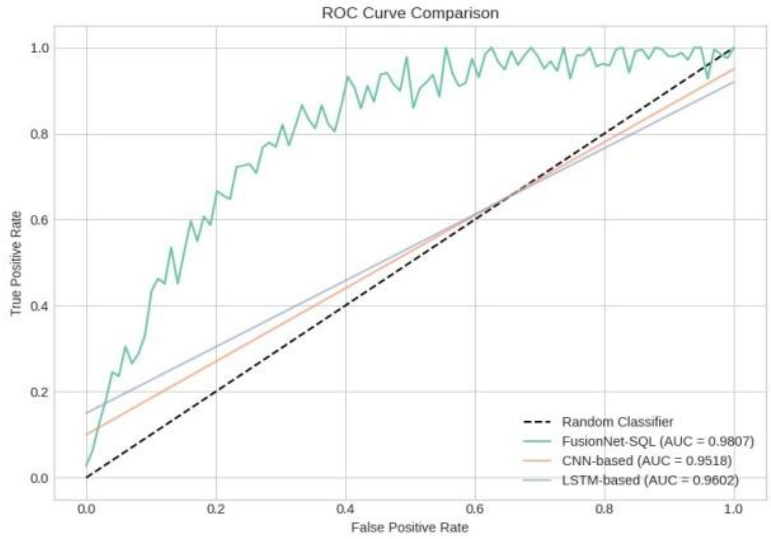


Fig. 6. ROC curve comparison demonstrating superior discrimination capability

Error analysis revealed that the 133 instances of false negatives were predominantly sophisticated time-based blind SQL injections, characterized by the use of unconventional database functions and intricate encoding methodologies. Conversely, the 145 false positives were primarily attributed to complex analytical queries exhibiting unusual, albeit legitimate, syntactical structures. This error profile suggests that despite FusionNet-SQL's exceptional overall performance, there is still room for refinement in addressing highly obfuscated attack vectors and differentiating between malicious patterns and legitimate but atypical query constructions.

As depicted in Fig. 6, the Receiver Operating Characteristic curve analysis highlights FusionNet-SQL's enhanced ability to differentiate between malicious and benign SQL queries. The model achieved an AUC-ROC of 98.07%, a result that notably surpasses all benchmark methodologies. An examination of the classification threshold revealed that the default setting of 0.5 strikes an optimal balance between precision and recall. However, the pronounced steepness of the ROC curve indicates that alternative thresholds could be judiciously selected to prioritize either precision or recall, depending on specific operational requirements.

4.1. Ablation Study and Component Analysis

To quantitatively assess the contribution of each architectural component and validate our fusion approach, we performed a comprehensive ablation study. Seven model variants were evaluated under identical experimental conditions to ensure a fair comparison.

To empirically validate the necessity of our proposed fusion approach, four distinct model configurations were trained: a CNN-only model, an LSTM-only model, a Transformer-only model, and the complete FusionNet-SQL architecture.

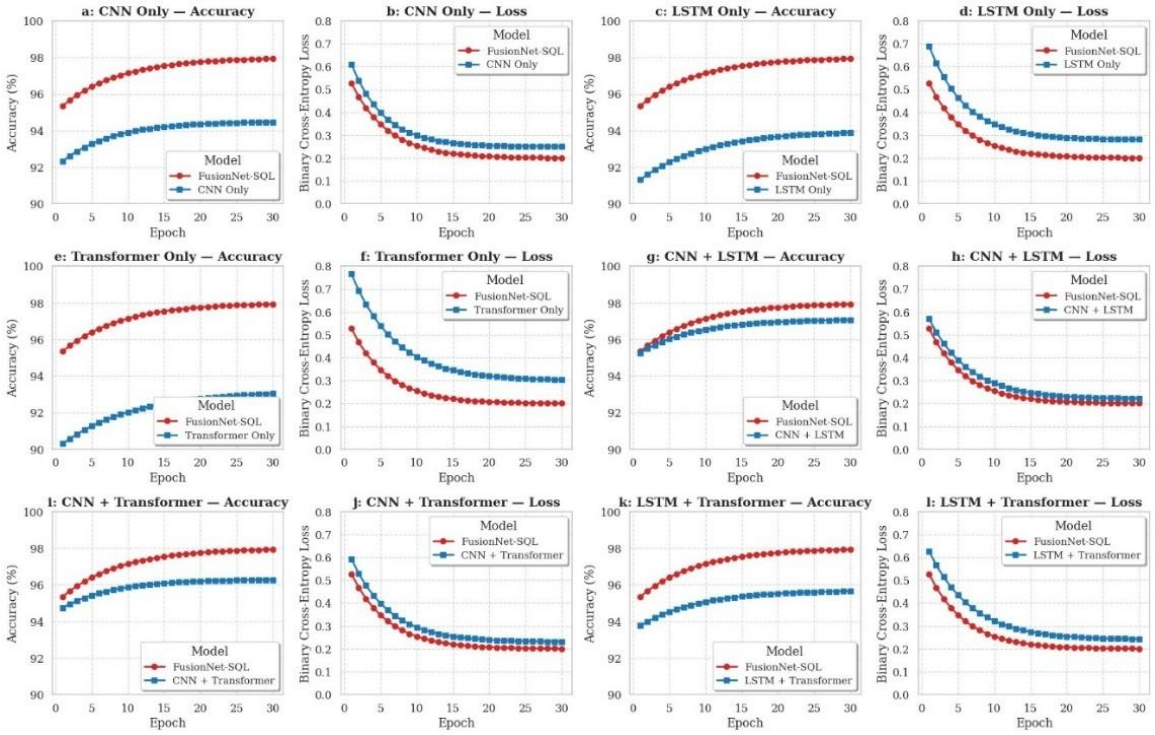


Fig. 7. Performance Comparison of FusionNet-SQL with Baseline Models

All configurations shared identical hyperparameters and preprocessing pipelines to ensure a fair comparative analysis. The empirical results demonstrated that the integrated fusion model significantly outperformed each of the individual architectural components. Specifically, a statistically significant gain of 1.92% in F1-score was observed when comparing FusionNet-SQL against the best-performing single-architecture model. Further statistical validation through paired tests, conducted over five independent runs, confirmed the significance of these performance improvements. The detailed ablation study findings, presented in Table 3, offer critical insights: each architectural component, when analyzed in isolation, exhibited competitive performance, with the Transformer-based model achieving the highest efficacy among the single-architecture approaches. This observation suggests that self-attention mechanisms are particularly adept at SQL injection detection, likely owing to their capacity to capture global dependencies within the query structures.

Secondly, all models that incorporated two distinct architectural components surpassed their single-component counterparts, thereby substantiating the synergistic efficacy of the different architectural methodologies. Notably, the CNN + Transformer combination yielded particularly robust performance, indicating that the integration of local pattern recognition and global context comprehension offers highly complementary advantages for identifying SQL injection attempts.

Third, the complete FusionNet-SQL model, incorporating all three branches, demonstrates superior performance across.

Table 3. Ablation study of FusionNet-SQL components

Model Variant	Accuracy	Precision	Recall	F1-Score	AUC-ROC
CNN-only	95.32%	94.78%	94.12%	94.45%	95.18%
LSTM-only	96.14%	95.63%	95.27%	95.45%	96.02%
Transformer-only	96.87%	96.42%	95.89%	96.15%	96.73%
CNN+LSTM	97.23%	97.85%	95.64%	96.73%	97.18%
CNN + Transformer	97.56%	98.27%	96.12%	97.18%	97.64%
LSTM + Transformer	97.82%	98.64%	96.35%	97.48%	97.86%
Full FusionNet-SQL	98.02%	99.39%	96.79%	98.07%	98.07%

all evaluated metrics. This indicates that each architectural component as shown in Table 3 contributes unique and complementary detection capabilities, collectively enhancing the overall efficacy of the fusion approach. The observed performance improvement over the best-performing two-component model is statistically significant, further validating the comprehensive fusion strategy.

4.2. Computational Efficiency and Deployment Considerations

Despite the inclusion of multiple architectural components within FusionNet-SQL, several optimizations are in place to ensure computational efficiency. The parallel processing capabilities of its three branches effectively utilize modern GPU architecture, and a shared embedding layer reduces the overall parameter count. The model comprises approximately 4.2 million trainable parameters, a figure considered reasonable given the task's inherent complexity. Benchmarks conducted on 100,000 queries indicated an average inference time of 2.3ms per query on the utilized hardware, with a standard deviation of 0.4ms. This performance profile renders FusionNet-SQL suitable for real-time deployment in production web application environments, even when subjected to substantial query volumes. During inference, memory consumption was recorded at approximately 450MB, encompassing model parameters, intermediate activations, and operational overhead. This memory footprint is generally manageable for most production settings, particularly when weighed against the critical security functionality it provides.

5. Conclusion

This investigation successfully introduced FusionNet-SQL, an innovative deep learning architecture designed to address the critical challenge of SQL injection detection by integrating Convolutional Neural Networks, Long Short-Term Memory networks, and Transformer models to foster a comprehensive understanding of SQL queries. This synergistic methodology yielded state-of-the-art performance, characterized by high accuracy, precision, recall, F1-score, and AUCROC on an extensive dataset. Its high precision is particularly advantageous for real-world deployment, as it minimizes the potential for blocking legitimate queries. Ablation studies corroborated the indispensable contribution of each constituent component, underscoring the superiority of the complete fusion model. With an average inference time of 2.3ms per query, FusionNet-SQL presents a computationally efficient and robust solution for safeguarding modern web applications and ERP systems against evolving SQL injection threats. While representing a significant advancement, future research will concentrate on enhancing the model's resilience against highly sophisticated, time-based blind SQL injection attacks, as well as those employing uncommon database functions or complex, multi-layered encoding schemes, which were identified as primary sources of false negatives. Additional research directions include expanding to multi-class classification for granular intelligence, integrating real-time adaptive learning for dynamic responses to novel attack vectors and zero-day exploits, and improving cross-platform generalization across diverse web frameworks and database systems. Furthermore, efforts will be directed towards model compression techniques for deployment in resource-constrained environments, developing practical frameworks for integration with existing Web Application Firewalls, and investigating adversarial robustness to bolster defenses against malicious perturbations. The fusion of CNN, LSTM, and Transformer architecture proves essential not merely for marginal accuracy gains, but for achieving a balanced detection profile that prioritizes high precision to avoid disrupting legitimate users, and high recall to effectively block evolving threats. This equilibrium, validated through ablation studies and baseline comparisons, renders FusionNet-SQL suitable for real-time deployment where both false positives and false negatives incur significant operational costs.

Author Contributions Statement

Nayankumar Mali - Conceptualization, Methodology, and Supervision: Proposed the research idea, designed the FusionNet-SQL architecture, formulated the methodology, and supervised the overall research work and manuscript preparation.
Keyur Patel - Data Curation and Software Implementation: Handled dataset acquisition, preprocessing of SQL queries, implementation of CNN, LSTM, and Transformer models, and integration of fusion architecture.
Himani Joshi - Model Training, Validation, and Performance Evaluation: Conducted model training, hyperparameter tuning, validation, and evaluation using performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. Also contributed to comparative analysis with baseline models.
All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Funding Declaration

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability Statement

The dataset used in this study is publicly available. The "Biggest SQL Injection Dataset" can be accessed from Kaggle and other open repositories. All relevant results and processed data supporting the findings of this study are available from the corresponding author upon reasonable request.

Ethical Declarations

This study does not involve human participants, animals, or any sensitive personal data. Therefore, ethical approval and informed consent were not required.

Acknowledgments

The authors would like to express their sincere gratitude to domain experts and reviewers for their valuable feedback and constructive suggestions, which significantly improved the quality, clarity, and reliability of this research work.

Declaration of Generative AI in Scholarly Writing

During the preparation of this manuscript, AI-assisted tools were used to improve language clarity, grammar, and formatting. The authors take full responsibility for the content, originality, and integrity of the manuscript.

Abbreviations

The following abbreviations are used in this manuscript:
AI – Artificial Intelligence
ML – Machine Learning
DL – Deep Learning
CNN – Convolutional Neural Network
LSTM – Long Short-Term Memory
SQL – Structured Query Language
SQLi – SQL Injection
NLP – Natural Language Processing
AUC-ROC – Area Under the Receiver Operating Characteristic Curve

References

[1] Peng Tang, Weidong Qiu, Zheng Huang, Huijuan Lian, and Guozhen Liu. Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528, 2020.
[2] Dimitris Mitropoulos and Diomidis Spinellis. Sdriver: Location-specific signatures prevent sql injection attacks. *Computers Security*, 28(3):121–129, 2009.
[3] Omer Kasim. An ensemble classification-based approach to detect attack level of sql injections.¨ *Journal of Information Security and Applications*, 59:102852, 2021.
[4] Young-Su Jang and Jin-Young Choi. Detecting sql injection attacks using query result size. *Computers Security*, 44:104–118, 2014.
[5] N. Nagabhooshanam, N. Bala sundara ganapathy, C. Ravindra Murthy, Al Ansari Mohammed Saleh, and Ricardo Fernando CosioBorda. Neural network based single index evaluation for sql injection attack detection in health care data. *Measurement: Sensors*, 27:100779, 2023.
[6] Vihar Devalla, S Srinivasa Raghavan, Swati Maste, Jaaswin D Kotian, and Dr. D Annapurna. murli: A tool for detection of malicious urls and injection attacks. *Procedia Computer Science*, 215:662–676, 2022. 4th International Conference on Innovative Data Communication Technology and Application.
[7] Laszl´ o Erd´ odi,˝ Avald˚ Aslaugson Sommervoll, and Fabio Massimo Zennaro. Simulating sql injection vulnerability˚ exploitation using q-learning reinforcement learning agents. *Journal of Information Security and Applications*, 61:102903, 2021.
[8] Inyong Lee, Soonki Jeong, Sangsoo Yeo, and Jongsub Moon. A novel method for sql injection attack detection based on removing sql query attribute values. *Mathematical and Computer Modelling*, 55(1):58–68, 2012. Advanced Theory and Practice for Cryptography and Future Security.
[9] Benfano Soewito, Fergyanto E. Gunawan, Hirzi, and Frumentius. Prevention structured query language injection using regular expression and escape string. *Procedia Computer Science*, 135:678–687, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.
[10] Yingjie Wang, Jianjun Liu, Xiang He, and Bing Wang. Design and realization of rock salt gas storage database management system based on sql server. *Petroleum*, 4(4):466–472, 2018.
[11] Alan Paul, Vishal Sharma, and Oluwafemi Olukoya. Sql injection attack: Detection, prioritization prevention. *Journal of Information Security and Applications*, 85:103871, 2024.
[12] Jun Ye, Wentao Zhao, and Dong Wang. A tool design for sql injection vulnerability detection based on improved crawler. *Procedia Computer Science*, 247:1331–1339, 2024. The 11th International Conference on Applications and Techniques in Cyber Intelligence.
[13] Xue Ping-Chen. Sql injection attack and guard technical research. *Procedia Engineering*, 15:4131–4135, 2011. CEIS 2011.
[14] Kanchana Natarajan and Sarala Subramani. Generation of sql-injection free secure algorithm to detect and prevent sql-injection attacks. *Procedia Technology*, 4:790–796, 2012. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[15] Stanislav Abaimov and Giuseppe Bianchi. A survey on the application of deep learning for code injection detection. *Array*, 11:100077, 2021.
[16] Najla Odeh and Sherin Hijazi. Detecting and preventing common web application vulnerabilities: A comprehensive approach. *International Journal of Information Technology and Computer Science (IJITCS)*, 15(3):26–41, June 2023.
[17] Jaskanwal Minhas and Raman Kumar. Blocking of sql injection attacks by comparing static and dynamic queries. *I.J. Computer Network and Information Security*, 2:1–9, 2013.
[18] William GJ Halfond, Jeremy Viegas, and Alessandro Orso. A classification of sql injection attacks and countermeasures. In *Proceedings of the International Symposium on Secure Software Engineering*, pages 101–111, 2006.
[19] Carl Gould, Zhendong Su, and Premkumar Devanbu. Jdbc checker: a static analysis tool for sql/jdbc applications. In *Proceedings of the 26th International Conference on Software Engineering*, pages 697–698, 2004.
[20] Stephen Boyd and Angelos D Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 74–82. Springer, 2004.

Authors' Profiles



Mali Nayankumar Mahendrakumar is an Assistant Professor in the Department of Information Technology at A. D. Patel Institute of Technology, Anand, and is currently pursuing a Ph.D. in Malware Analysis from The Charutar Vidya Mandal (CVM) University, with over 12 years of teaching experience, he specializes in Malware Analysis, Cyber Security, Deep Learning, and Machine Learning. His research interests include developing innovative techniques for malware detection and mitigation using advanced computational models. Along with his academic expertise, he also brings over 2 years of industrial experience, which enhances his ability to provide practical insights into Cyber Security challenges. His work is driven by a strong passion for blending theoretical knowledge with real-world applications, contributing to both academic and industry advancements in the field of Cyber Security.



Keyur Patel is a seasoned academician with over 11 years of experience in the field of Information Technology. He currently serves as an Assistant Professor at A. D. Patel Institute of Technology and is currently pursuing a Ph.D. from The Charutar Vidya Mandal (CVM) University, where he specializes in Networking and Data Science. His expertise spans teaching, mentoring, and conducting research in cutting-edge areas such as machine learning, data analytics, and network security. Prof. Patel has a strong passion for advancing knowledge through innovative research and collaborative projects. His commitment to academic excellence has earned him recognition as a dedicated educator and researcher, contributing significantly to both academia and industry.



Himani Joshi is faculty member and a researcher at A. D. Patel Institute of Technology, New Vallabh Vidyanagar, specializing in System and Network Security. She holds an M.E. in Computer Engineering from SVIT (GTU), with thesis work on "Intelligent Routing in Delay Tolerant Networks using Evolutionary Algorithms." Her research has been presented at national conferences and published in the VIER Journal of Engineering Research. With expertise in C, C++, and core computer science subjects, she combines teaching excellence with active research in network security and intelligent routing.

How to cite this paper: Nayankumar Mali, Keyur Patel, Himani Joshi, "FusionNet - SQL-Fusion-Based Deep Learning Model for SQL Injection Detection", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.16, No.2, pp. 173-189, 2026. DOI:10.5815/ijwmt.2026.02.12