NTU Journal of Engineering and Technology (2025) 4 (4): 1 - 17
DOI: https://doi.org/10.56286/ntujet.v4i4

IRAQI Academic Scientific Journals

P-ISSN: 2788-9971 E-ISSN: 2788-998X
NTU Journal of Engineering and Technology
Available online at: https://journals.ntu.edu.iq/index.php/NTU-JET/index

Machine Learning and Deep Learning Approaches for SQL Injection Detection: A Review

Sahar Saadallah Ahmed[1], Mohand lokman Al dabag[2]
[1]Department of Computer Engineering Technology, Engineering Technical College, Northern Technical University, Iraq,
[2]Computer Center, Northern Technical University, Iraq.
sahar_saadallah@ntu.edu.iq, mohandaldabag@ntu.edu.iq

Article Informations

Received: 27-04- 2025,
Revised: 21-08-2025,
Accepted: 01-09-2025,
Published online: 28-12-2025

Corresponding author:
Name:Sahar Saadallah Ahmed
Affiliation: Department of Computer Engineering Technology, Engineering Technical College, Northern Technical University, Iraq
Email:
sahar_saadallah@ntu.edu.iq

Key Words:
Web Application Security,
SQL Injection (SQLi),
Machine Learning (ML),
Deep Learning (DL),
Natural Language Processing (NLP),

ABSTRACT

Structured Query Language Injection (SQLi) remains one of the most serious threats to web applications and has the ability to bypass traditional signature-based detection through obfuscation and zero-day payloads. This has driven the wider application of Machine Learning (ML) and Deep Learning (DL) techniques. This paper analyzes 50 peer-reviewed literatures published in the interval between 2015 and 2025, where the reported accuracy of detection ranged between 93 and 99.9%. Traditional ML methods include Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR), and Decision Tree (DT). DL approaches encompass Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and transformer-based models such as Bidirectional Encoder Representations from Transformers (BERT). Feature extraction methods include Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, and contextual embeddings. Evaluation of proposed models uncover new research opportunities in terms of lack of data availability, the problem of calss imbalance, real-time application, and excessive use of hardware resources.

1. Introduction

The fast growth of web-based systems has created major security problems, especially in injection attacks that take advantage of poor input handling. Structured Query Language Injection (SQLi) is one of the most common and harmful of these attacks. By sending specially designed input, an attacker can change how the database works, causing data leaks or even complete system compromise [1].

Fig. *1* shows a standard SQLi example in a web login form. In this case, weak input validation allows a malicious user to change the SQL query and get into the system without permission.

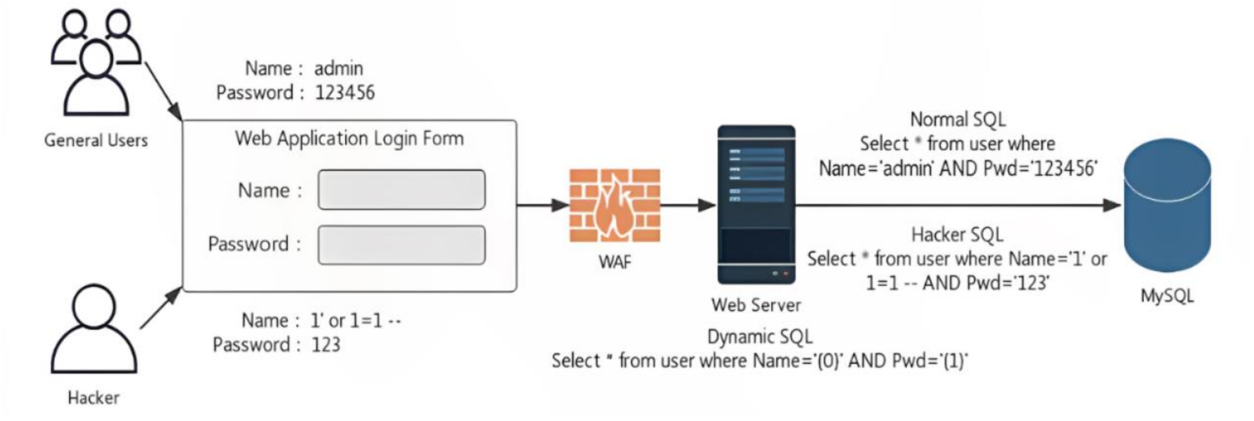
General Users
Name : admin
Password : 123456
Web Application Login Form
Name :
Password :
WAF
Web Server
Normal SQL
Select * from user where Name='admin' AND Pwd='123456'
Hacker SQL
Select * from user where Name='1' or 1=1 -- AND Pwd='123'
MySQL
Hacker
Name : 1' or 1=1 --
Password : 123
Dynamic SQL
Select * from user where Name='(0)' AND Pwd='(1)'

Fig. 1. SQL Injection Attack Process [1].

The annual reports of the Open Worldwide Application Security Project (OWASP) Foundation, between 2017 and 2021, ranked SQLi among the top ten most critical web application security risks. During this period, its rank fell from first to third place, despite that SQLi remained persistently mentioned in cases involving data breaches and infrastructure compromises [2]. Fig. 2 presents the OWASP Top 10 web application security risks, showing the position of SQL injection.

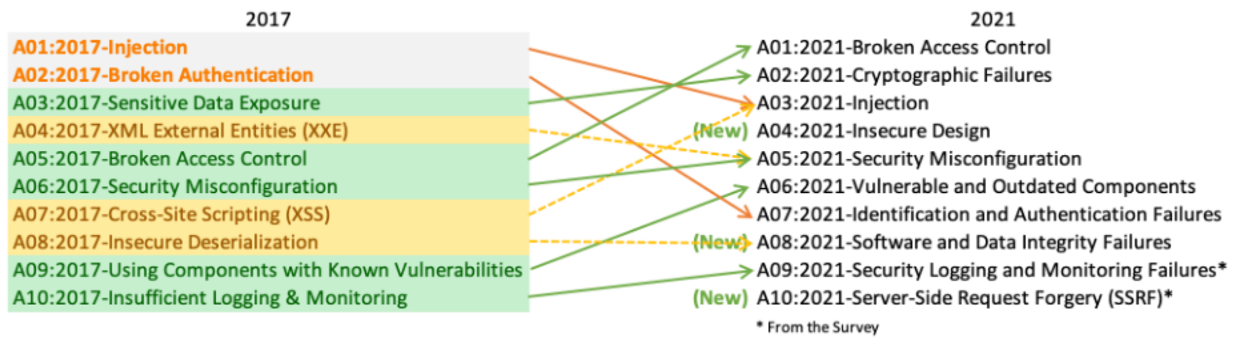
2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

Fig. 2. OWASP Top 10 risks in 2017 and 2021 [2].

Modern, obfuscated, or zero-day SQLi attacks have become increasingly difficult to detect using conventional signature-based or rule-driven defense mechanisms. These techniques rely on pre-established patterns that are inadequate to address new variations or payloads that are behaviorally hidden [3]. Consequently, the researchers became more interested in new and more intelligent solutions due to the wide gap between attacker evolution and stagnation of defenses.

Artificial intelligence (AI) has emerged as an essential part of the development of cybersecurity systems. AI uses approaches like pattern recognition, semantic analysis, and self-directed learning to detect and respond to complex attacks in real time. The machine learning (ML) component of AI allows models to recognize suspicious or unusual patterns in large datasets without requiring complex programming [4, 5]. Popular Algorithms such as Support Vector Machine (SVM), Random Forest (RF), and Neural Networks (NN) possess a high level of accuracy when classifying malicious inputs using their structural and semantic properties. Intuitively, it seems that deep models, including Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), could specialize in contextual embeddings and complex sequence dependencies related to query strings. For instance, it has been demonstrated that CNN-based detectors could identify and capture obfuscated payloads much more efficiently than traditional classifiers and extract multi-level features. [6]. Therefore, by incorporating such models, it will be feasible to develop a much more robust and scalable system to detect SQLi.

To further enhance the detection of SQLi attacks, researchers have increasingly adopted Natural Language Processing (NLP) methods alongside ML techniques. Text representation strategies such as Term Frequency–Inverse Document Frequency (TF-IDF), Word2Vec, and Bidirectional Encoder Representations from Transformers (BERT) have proven effective in capturing the syntactic and semantic structures of SQL queries, allowing classifiers to better distinguish between malicious and benign statements. These embeddings assist in capturing contextual meaning, which is crucial for detecting subtle injection patterns [4]. In addition, recent studies have highlighted the advantages of hybrid models and ensemble learning techniques that combine multiple algorithms to enhance robustness and accuracy, especially in identifying obfuscated payloads and managing imbalanced datasets in real-time environments [7, 8].

1.1 Research gaps

While the progress made in the studies is commendable, the studies highlighted in the research still showcase gaps in real world implementations of SQL injection detection. First, there remains a strong reliance on classical ML paired with Bag-of-Words (BoW), and TF-IDF rather than contextual embeddings or hybrid deep architectures, which constrains generalization to obfuscated or previously unseen payloads [9-11].

Second, the input scope is frequently narrow; most works operate on query strings or textual payloads, with few explicitly handling richer vectors. For example, several studies use only textual inputs or Kaggle-style SQL query corpora without broader protocol/context features [10, 12, 13].

Third, evidence of online or production-grade validation is limited: a handful demonstrate deployment or real-time feasibility (e.g., a Flask app or firewall and edge suitability), but most evaluations remain offline [10, 12, 14, 15].

Fourth, explainability is underutilized, apart from a few works employing SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME), most models lack integrated Explainable Artificial Intelligence (XAI) to support auditing in security-critical settings [16, 17].

Fifth, data limitations persist, including small or proprietary datasets and reliance on single public corpora, which can bias reported metrics; concrete examples include modest-sized or single-dataset evaluations and limited external validation [18-21].

Finally, only a few frameworks consider multiple web attacks (e.g., SQLi and cross-site scripting (XSS)) within a unified model, while most approaches remain narrowly scoped to SQLi [22, 23].

1.2 Problem statement

Although numerous studies have explored SQL Injection detection using ML and DL techniques, the development of robust and practically deployable solutions remains limited. Current detection frameworks frequently limit their analysis to query strings or POST parameters, overlooking other potentially exploitable vectors, including Hypertext Transfer Protocol (HTTP) headers (e.g., User-Agent, Cookies) and Domain Name System (DNS)-level payloads. DL models have shown good accuracy in controlled settings, but they can't be used in real time because they take up too much processing power and aren't flexible enough to handle new or hidden payloads. This is especially true for Web Application Firewalls (WAFs) and Intrusion Detection Systems (IDSs).

Furthermore, the opaque nature of numerous deep learning architectures constrains interpretability, inhibiting security analysts from validating or auditing detection results, which is a crucial necessity for security-sensitive systems. Class imbalance in SQLi datasets skews models even more toward benign queries, making them less sensitive to rare but dangerous attack patterns. Lastly, many of the proposed detection methods haven't been tested in real-world settings, raising concerns about scalability, operational feasibility, and long-term adaptability.

1.3 Research questions

This review tries to answer some basic questions about the current state and future direction of SQLi detection research.

First, it tries to find and compare the different machine learning (ML) and deep learning (DL) methods that have been used to detect SQLi. It focuses on how accurate, efficient, and useful they might be in the real world.

Second, the review looks at the extent to which recent detection models being able to adapt in real time, being able to analyze multiple input vectors, and being able to explain themselves using XAI techniques. Third, it looks at the big research gaps and technical problems that still exist in the current literature.

Finally, it seeks to establish a compilation of optimal practices and prospective research trajectories that can facilitate the advancement of more resilient, comprehensible, and scalable SQL injection detection systems.

1.4 Research objectives

This review provides a critical, comprehensive synthesis of machine learning (ML) and deep learning (DL) based SQL injection detection from peer-reviewed studies published between 2015 and 2025. First, it shows how model families, like classical, deep, and hybrid, can be designed and what preprocessing and feature-representation options they have. Second, it looks at how well things work in real life, with accuracy as the main way to compare them. Third, it points out common methodological problems, such as limited generalizability, class imbalance, XAI, computational overhead, and real-time constraints. Fourth, it looks at practical issues that come up when trying to make something scalable and deploy it in a security setting. Lastly, it puts together evidence-based suggestions and design rules for making SQLi detection systems that are strong, easy to understand, and can grow to fit real-world situations like WAFs and IDSs.

2. Research Methodology

This review follows a set paln for reviewing literatures. We restricted the scope to peer-reviewed studies on SQLi detection using ML and DL published between 2015 and 2025. We searched IEEE Xplore, SpringerLink, MDPI, and Google Scholar using combinations of the terms SQL injection, detection, classification, ML, and DL. Duplicates were removed, records were screened by title and abstract, then assessed in full text against predefined inclusion criteria (English, peer-reviewed, ML and DL for SQLi with quantitative evaluation) and exclusion criteria (non-ML approaches, no dataset or metrics, non-English, duplicates, outside the timeframe). We retrieved 100 records; after deduplication, 71 records remained for title–abstract screening and full-text assessment, of which 50 studies met the inclusion criteria and were included in the final review. For each included study, we extracted fields aligned with our summary table: Authors, Objective, Method, Accuracy (was employed as the primary evaluation metric to assess the performance of the applied models. Accuracy is formally defined as the proportion of correctly classified instances (both attack and benign queries) over the total number of instances, as shown in Equation (1):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (1)$$

where TP denotes true positives, TN true negatives, FP false positives, and FN false negatives. A higher accuracy value indicates that the model can correctly distinguish between malicious and legitimate SQL queries with greater reliability [30]), Strength Points, and Limitations, and complementary details (model family, preprocessing and feature representation, use of explainability, and any deployment or real-time evidence such as Flask, microservices, or firewall or edge settings). Owing to heterogeneity across datasets and experimental setups, findings were synthesized narratively, grouped by model family and representation choice; no reimplementation or additional quantitative testing was performed.

3. Related Work

Prior research on SQL injection detection spans two broad strands: classical machine-learning pipelines driven by handcrafted textual features, and deep and hybrid architectures that learn hierarchical or contextual representations directly from payloads.

Classical ML with handcrafted features. Numerous studies rely on BoW and TF-IDF (or variants such as Improved Term Frequency–Inverse Document

Frequency(ITFIDF)) with classifiers including SVM, Logistic Regression (LR), Naive Bayes (NB), k-Nearest Neighbors (KNN), Decision Tree (DT), RF, and boosted ensembles, often reporting competitive accuracy under controlled settings [9, 15, 24, 25].

Work has also examined data imbalance and sampling effects on SQLi detection performance, highlighting the sensitivity of results to class skew and evaluation protocol [8]. Feature-selection methods also appear in several papers to streamline models or improve generalization [18, 20, 26].

Deep and hybrid models. CNN-based detectors trained on network or HTTP payloads have shown strong results without manual feature engineering [27, 28], while hybrids combining CNN with Bidirectional Long Short-Term Memory(BiLSTM) capture both local n-gram patterns and long-range dependencies [13, 22, 29].

Representation-learning approaches include autoencoder-derived features used with downstream classifiers [30]

Deep Forest as a non-neural deep alternative [31]. Lightweight attention models designed for low-latency inference [12].

Contextual and sentence-level embeddings. Several works move beyond sparse vectors toward dense, semantics-aware representations. Comparative studies evaluate contextualized embeddings against BoW and TF-IDF with classical learners [5], while BERT-based or BERT-hybrid architectures (e.g., BERT–LSTM, Syntactic Bidirectional Encoder Representations from Transformers(synBERT)) aim to capture obfuscation-resilient semantics at the token or sentence level [23, 32-35].

Explainability and deployment aspects. A subset of studies integrates XAI tooling, such as SHAP or LIME, to enhance the auditability of model decisions, particularly in operational settings [16, 17].

On the deployment side, prior work demonstrates practical prototypes or settings including Flask-based applications, microservice-oriented designs, and firewall, edge-style integrations, indicating a path from offline evaluation to applied use cases [10, 12, 14, 15, 36]. Other efforts aggregate multi-source traffic or Internet Service Provider(ISP) data to approximate realistic conditions [37-39].

Beyond a single-attack scope. While much of the literature targets SQLi specifically, some frameworks broaden the scope to multiple web threats (e.g., joint SQLi and XSS detection) or related query-injection families, underscoring the importance of generalizable defenses in heterogeneous environments [15, 22, 35].

Fig. 3 Conceptual pipeline summarizing the main stages of SQL injection detection techniques, based on prior research themes including feature extraction, imbalance handling, classification, explainability, and deployment.

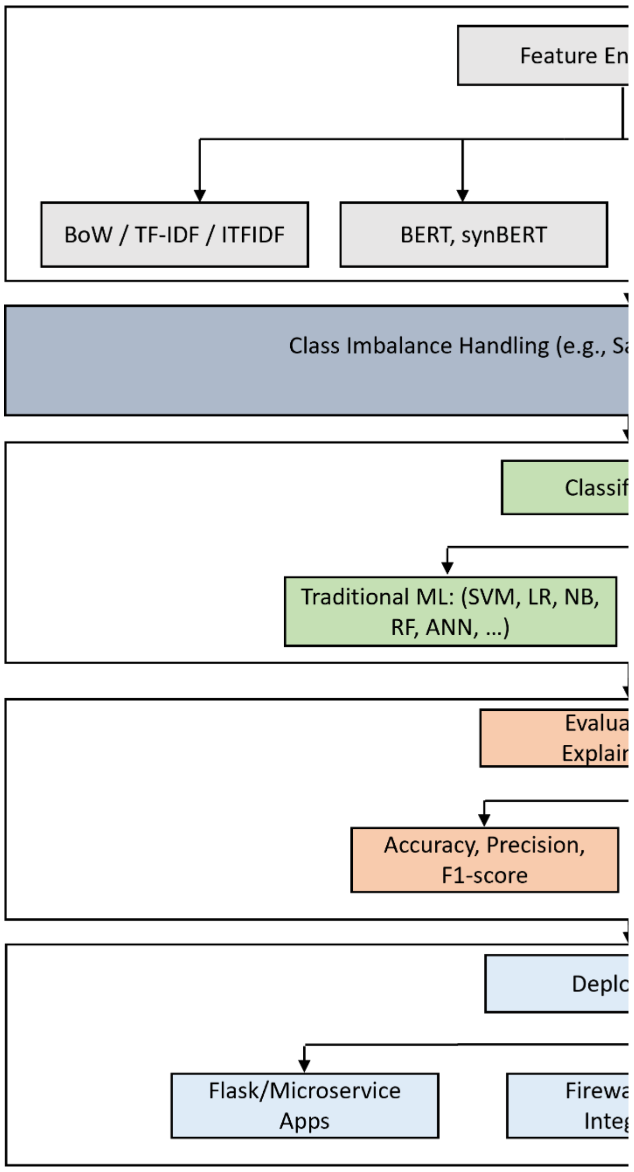
Feature En
BoW / TF-IDF / ITFIDF
BERT, synBERT
Class Imbalance Handling (e.g., Sa
Classif
Traditional ML: (SVM, LR, NB, RF, ANN, …)
Evalua
Explair
Accuracy, Precision, F1-score
Deplo
Flask/Microservice Apps
Firewa
Integ

Fig. 3. Conceptual workflow of SQL injection detection researches.

Table *1.* Summary of related works evaluation. presents a focused comparison of selected representative studies from the reviewed literature. Each entry highlights the key technique used by the authors and outlines the most critical limitations of their proposed approaches. This table aims to emphasize not only the methodological contributions but also the gaps that hinder real-world applicability, such as a lack of explainability, the absence of feature selection or hybridization, and limited scalability or deployment.

Fig. ***3*** Conceptual pipeline summarizing the main stages of SQL injection detection techniques, based on prior research themes including feature extraction, imbalance handling, classification, explainability, and deployment.

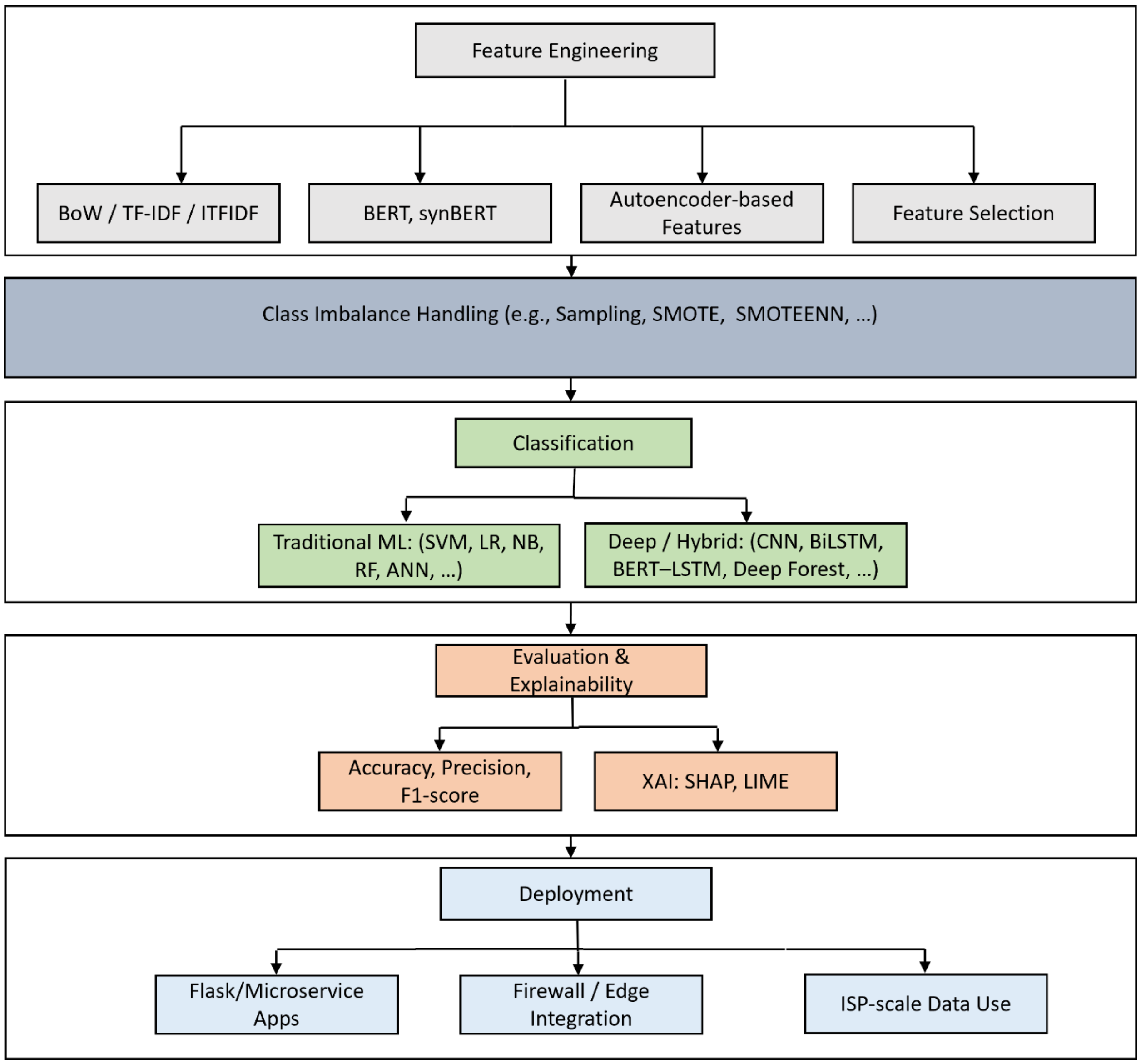
Feature Engineering
BoW / TF-IDF / ITFIDF
BERT, synBERT
Autoencoder-based Features
Feature Selection
Class Imbalance Handling (e.g., Sampling, SMOTE, SMOTEENN, …)
Classification
Traditional ML: (SVM, LR, NB, RF, ANN, …)
Deep / Hybrid: (CNN, BiLSTM, BERT–LSTM, Deep Forest, …)
Evaluation & Explainability
Accuracy, Precision, F1-score
XAI: SHAP, LIME
Deployment
Flask/Microservice Apps
Firewall / Edge Integration
ISP-scale Data Use

Fig. 3. Conceptual workflow of SQL injection detection researches.

Table 1. Summary of related works evaluation.

Author	Objective	Method	Accuracy	Strength Points	Limitation
(Chen et al.2021) [1]	To develop DL based SQLi detection model that avoids rule-based systems.	Preprocessed HTTP traffic through recursive decoding and generalization, then applied Word2Vec for embedding; trained CNN and Multilayer Perceptron(MLP) models; evaluated using confusion matrices and F1-score.	98.58%	Combines NLP and DL; effective use of Word2Vec, CNN, and MLP; evaluated on real HTTP data with detailed metrics.	Focuses only on first-order SQLi; limited to offline evaluation; second-order and hybrid SQLi attacks not addressed.
(Farooq. 2021) [4]	To develop an effective SQL injection detection system using	Created a labeled dataset of 35,198 queries (normal, malicious, plain text) with 21 statistical and semantic features.	99.34%	Manually constructed, balanced, and labeled dataset with detailed feature engineering;	No DL models used, dataset limited to structured queries; performance in real-world

	ensemble ML techniques and a manually constructed feature-rich dataset.	Applied four ensemble classifiers (Gradient Boosting Machine(GBM), Adaptive Boosting(AdaBoost), Extreme Gradient Boosting(XGBoost), Light Gradient Boosting Machine(LGBM)) and evaluated using 3-fold and 5-fold cross-validation.		extensive evaluation using multiple metrics, comparative analysis with prior methods showed improvement.	unstructured web logs not tested.
(Zulu et al.2024)[5]	Compare contextualized vs. non-contextualized word embeddings for SQLi detection using ML models	Applied BoW and Robustly Optimized BERT Pretraining Approach (RoBERTa). embeddings; trained MLP, RF, KNN, and LR classifiers on Kaggle SQLi dataset	above 99%	High accuracy across models, significant reduction in training time, better model calibration with contextual embeddings	Training excludes pretraining cost of RoBERTa; higher memory needed for RoBERTa embeddings; BoW models consumed high memory and showed poor generalization in some classifiers
(Zuech et al 2021) [8]	Investigate the impact of extreme class imbalance and rarity on the detection of SQL Injection web attacks in the CSE-CIC-IDS2018 dataset.	Evaluated 7 classifiers (DT, RF, LGB, XGBoost, Categorical Boosting (CatBoost), NB, LR) across different Random Under-Sampling (RUS) random undersampling) ratios; Used stratified 5-fold CV repeated 10 times (total 2800 runs); Statistical analysis via Analysis of Variance(ANOVA) and Tukey's HSD.	-	Rigorous data preparation; Deep statistical validation; Explored classifier and sampling interactions; First to deeply analyze SQLi rarity in CSE-CIC-IDS2018.	Did not explore oversampling or hybrid sampling; Only focused on Area Under the Curve(AUC) metric; Evaluation limited to SQL Injection attacks only from two days in the dataset.
(Pramono et al, 2024).[9]	Compare the effectiveness of NB and SVM in classifying SQLi.	Preprocessing TF-IDF and NB vs. SVM using the labeled Kaggle dataset.	96.67%	Simple yet effective comparison; tested 3 data split scenarios.	Limited to Kaggle dataset; does not apply ensemble or DL methods.
(Ahmed et al.2021) [10]	Detect SQL and NoSQL injection attacks using a hybrid two-layer firewall.	Layer 1: Pattern matching. Layer 2: ML classifiers (SVM, DT, AdaBoost, RF, LR, NB), Synthetic Minority Over-sampling Technique with Edited Nearest Neighbors(SMOTEENN), GridSearchCV.	100%	Real-time firewall; high accuracy; handles both SQL and NoSQL; dataset published.	Only textual input; limited NoSQL samples; no file support; no DL used.
(Ahmed and Uddin 2020) [11]	To enhance SQL injection detection accuracy using ML and NLP techniques.	Collected SQLi and normal payloads using tools such as LibInjection and SQLMap; used token pattern via regex and CountVectorizer to extract BoW features; applied RF classifier with bagging; compared with DT, NB, SVM, and KNN.	98.15%	Integrated NLP-based BoW feature extraction, large dataset, high classification metrics, and robust comparison with four classifiers.	Limited to classical ML, lacks DL or word embedding; no real-time or online system evaluation implemented.
(Lo et al.2025) [12]	Develop a lightweight and efficient neural network model for SQL injection detection.	SQL-specific tokenizer using command expression and symbol categories combined with CNN and multi-head self-attention, followed by a sigmoid output layer trained on the Kaggle SQL injection dataset.	98.98%	Compact model with 69,269 parameters, fast inference suitable for edge devices, competitive with large pretrained models like A Lite	Slightly lower accuracy than DistilBERT and Efficiently Learning an Encoder that Classifies Token Replacements Accurately

				BERT(ALBERT) and Distilled BERT(DistilBxERT).	(ELECTRA), limited to SQL injection detection, depends on the quality of the SQL-specific tokenizer.
(Gandhi et al.2021) [13]	To develop a hybrid CNN-BiLSTM model for accurate detection of SQL injection attacks.	4200 labeled queries processed through cleaning, tokenization, and embedding, classified using CNN for feature extraction and Bi-LSTM for sequence modeling.	98%	High detection accuracy, effective hybrid architecture, robust comparison with multiple models.	Moderate execution time, limited dataset size, no discussion of real-time deployment or adversarial threats.
(Ogini, et al.2022) [14]	To develop a DL based Feed-Forward Neural Network(FFNN) model for detecting SQL injection attacks.	Used a dataset of 30,635 SQL queries from Kaggle. Preprocessed with CountVectorizer and TF-IDF. Trained FFNN in TensorFlow, Keras, over 20 epochs, and deployed using Flask.	97.65%	High detection accuracy; real-time Flask deployment; thorough preprocessing.	Limited to FFNN; only evaluated on Kaggle data; no hybrid model comparison or overfitting discussion.
(Pallam et al.2021) [15]	Detect SQLi & XSS using ML ensemble methods.	Used TF-IDF with LGBM, AdaBoost, XGBoost , and GBM; deployed on a Flask app.	99.59%	Practical deployment, high accuracy, and an IP ban feature.	Limited XSS data; no real-time and adversarial testing.
(Kakisim 2024) [16]	To improve SQL injection (SQLi) detection using a DL system leveraging multiple semantic representations.	Introduced Multi-View Convolution-Bidirectional Convolutional Neural Network(MVC-BiCNN), combining BiLSTM and CNN; applied multi-view learning with tokenized, converted, and enriched representations; used 21 semantic tags; evaluated on 5 datasets.	99.96%	Effective multi-view consensus strategy; strong generalization across datasets; robust feature representation; XAI explainability with LIME.	Focused only on SQLi; does not address XSS or other web attacks; limited real-time deployment evaluation.
(Le et al., 2024).[17]	Compare ensemble and boosting models and enhance SQLi detection transparency using SHAP and LIME.	Trained DT, RF, XGBoost, AdaBoost, Gradient Boosting Decision Tree(GBDT), and Histogram-based Gradient Boosting Decision Tree(HGBDT) on an SQL injection dataset; evaluated using accuracy, F1-score, SHAP, and LIME explainability techniques.	99.50%	High detection accuracy; explainability through SHAP and LIME; suitable for real-world deployment; thorough comparative evaluation of multiple models.	SHAP is computationally expensive; XGBoost showed limited performance in this use case; evaluation was limited to a controlled dataset, not real-time or adversarial settings.
(Alarfaj & Khan 2023).[18]	To enhance the detection performance of SQL injection attacks using DL	A Probabilistic Neural Network (PNN) optimized via the BAT algorithm; features were extracted using tokenization and regular expressions, and selected via Chi-Square test. The dataset included 6000 SQLi and 3500 benign queries. Evaluated using 10-fold cross-validation and compared with SVM, Artificial Neural Network(ANN), and DT.	99.19%	Utilized an optimization algorithm, BAT, to fine-tune the smoothing parameter; high detection accuracy; comprehensive evaluation with multiple classifiers and validation setups.	High model complexity; sensitivity to noisy or irrelevant features; a custom dataset may limit generalizability.

(Hernawan et al, 2020) [19]	To prevent SQL Injection attacks using a hybrid intelligent detection system.	A hybrid model combining SQL Injection Free Secure (SQL-IF) and NB, implemented as a proxy.	90%	Integrates pattern matching and ML; works as a real-time proxy; evaluated using real attacks.	Increased web load time due to dual processing; dataset limited to 250 queries; lacks external validation.
(Arasteh et al.2024) [20]	Improve SQL injection detection by selecting optimal features and classification techniques.	Developed a Binary Olympiad Optimization Algorithm (BOOA) for feature selection; trained classifiers (ANN, DT, SVM, KNN) on 13 numerical features; compared performance with and without BOOA-selected features.	99.35	Effective feature reduction boosting accuracy; high stability across runs; combination of BOOA and ANN yields top performance.	Does not explore DL; limited to a modest dataset size; only 13 hand-engineered features used.
(Alghawazi et al.2023) [21]	To develop an RNN autoencoder architecture for detecting SQL injection attacks.	They trained an RNN autoencoder consisting of an encoder–decoder pair to compress and reconstruct SQL queries and added an LSTM-based classifier on the encoded representations.	94%	The architecture captures sequential patterns effectively and was benchmarked against seven other classifiers.	Evaluation used a single public dataset, without real-world deployment or larger, more diverse data samples.
(Tadhani et al.2024) [22]	Develop a unified DL model to detect both SQLi and XSS web attacks effectively.	Hybrid CNN–LSTM model with preprocessing (decoding, standardization, tokenization) and Word2Vec embedding.	99.84%	Unified model detects multiple attack types; high accuracy across three datasets; effective preprocessing and embedding strategy.	High training time; performance may be dataset-dependent; not tested on other attack types like Zero-day or phishing.
(Bakır 2025) [23]	Unified detection of XSS and SQL injection using fused embeddings.	UniEmbed: fuse Universal Sentence Encoder(USE) sentence-level, Word2Vec word-level, FastText subword-level; train ML classifiers LR, SVM, GNB, DT, KNN, MLP, RF; use hard voting and soft voting.	99.82%	Multi-level feature fusion yields top performance; consistent results across datasets and classifiers; efficient inference.	Tested only on benchmark datasets; no live deployment; limited to text inputs; not evaluated on novel attack types.
(Li & Bin 2019) [24]	To enhance SQL injection attack detection by improving the traditional TF-IDF algorithm through distribution-aware feature weighting.	Proposed an ITFIDF algorithm considering feature distribution across SQL statement types, extracted 34 features including keyword frequency and ITFIDF of 32 sensitive characters, used SVM for classification, and compared against existing methods and classifiers.(KNN and DT).	99.08%	Higher feature representation accuracy, superior classification performance with SVM, and improved metrics over baseline methods.	Limited dataset diversity, no real-time system evaluation, lacks contextual or semantic analysis of SQL statements.
(Zhang et al.2022) [25]	Propose a deep neural network (SQLNN) for accurate SQL injection detection.	SQL Neural Network (SQLNN) model using TF-IDF for feature extraction, Rectified Linear Unit(ReLU) activation, Adam optimizer, and Dropout regularization; compared with KNN, DT, and LSTM.	96.16%	Integrates DL with automated feature extraction, avoids overfitting using Dropout, robust against evasion attacks.	Lacks real-time deployment validation; relies on a single dataset from Kaggle.
(Purbawa et al.2023) [26]	Enhance detection accuracy of SQLi using an ML ensemble.	Preprocessing (stemming, lemmatization, TF-IDF), models: LR, LDA, GNB, RF, and Voting Classifier (soft voting).	97.07%	Combined multiple ML algorithms with feature selection (ANOVA) and vectorization (TF-IDF), using Kaggle data.	Limited to classical ML models only, no comparison with DL, used only one small dataset version.

(Shahbaz et al, 2024).[27]	To propose a CNN-based model for detecting SQL injection attacks without manual feature engineering.	Built a CNN model trained on a dataset of 109,520 SQL queries (80% train, 20% test) using embedding, with one-dimensional convolution (Conv1D), pooling, and dense layers for feature extraction and classification.	98.16%	High detection accuracy; automatic feature learning; minimized false positives and negatives.	Requires diverse data; high computational resources; slightly lower performance on malicious queries (FN=484).
(Luo et al.2019) [28]	To detect SQL injection attacks using a CNN-based model and compare it with traditional rule-based detection.	Collected SQL injection payloads from real HTTP traffic, applied data sanitization and vectorization using Gensim, and trained a CNN model consisting of three convolutional and pooling layers with a fully connected and hidden layer.	99.50%	Utilized real-world traffic, applied thorough preprocessing, and demonstrated robustness against obfuscated attacks.	Focused only on CNN without comparing other DL models, limited to binary classification, and lacked broader dataset diversity.
(Alshammari 2023) [29]	To evaluate and compare the performance of ANNs, CNNs, and RNNs for detecting SQL injection attacks.	Applied ANN with TF-IDF vectors, CNN with embedding, convolution, pooling, and RNN with LSTM on SQL query dataset.	99.70%	Clear comparison between different neural network architectures using the same dataset and evaluation metrics.	Limited dataset; no real-world deployment; lacks comparison with traditional ML or hybrid models.
(Thalji et al. 2023) [30]	To develop an automated, AI-based method for detecting SQLi attacks without human intervention.	Proposed Autoencoder Network (AE-Net) to extract high-level deep features from SQL queries. Evaluated with ML and DL models: KNN, LR, RF, XGBoost (for BoW & TF-IDF), and KNN, RF, XGBoost, LSTM (for AE-Net features). Hyperparameter tuning and k-fold validation applied.	99%	Novel deep features improved detection; robust evaluation.	LSTM underperformed; high runtime for XGBoost; no real-time Graphical User Interface(GUI).
(Q. Li et al. 2019) [31]	Propose an effective SQLi detection method for complex environments.	Adaptive Deep Forest (ADF) with AdaBoost uses multi-grained scanning and cascade forest architecture.	98.00%	Automatically adjusts parameters; low overfitting; high accuracy with small datasets; better than Deep Neural Network(DNN) and ML models.	Slightly outperformed by DNN when training samples exceed 16,000; limited dataset diversity.
(Liu and Dai 2024)[32]	Propose a hybrid BERT–LSTM model for detecting SQLi attacks.	BERT for contextual embeddings, and LSTM for sequence modeling. Dataset: HttpParams (30,156 samples).	97.3%	High accuracy; robust to obfuscation; effective semantic modeling.	Limited performance on encoded/XSS attacks; needs decoder integration.
(Lu et al 2023).[33]	Propose an accurate and generalizable model for SQL injection detection.	Developed synBERT, a semantic learning-based model that embeds SQL statement semantics; trained and evaluated on multiple datasets.	99.74%	Introduces semantic understanding via syntax trees; strong generalization, BERT-based deep architecture.	Missing details on dataset balance and real-world scalability; limited ablation and interpretability analysis.

(Sun et al.2023) [34]	To propose a DL-based detection method for SQL injection attacks.	Employed an enhanced TextCNN for local feature extraction, followed by Bi-LSTM for sequential learning. Integrated an attention mechanism to improve long-sequence handling and incorporated BERT for transfer learning.	above 99.57%	A combination of CNN, Bi-LSTM, attention, and BERT; effective for complex and evolving SQLi patterns.	Dataset type and size not specified; lacks exact metrics (e.g., accuracy, F1-score); not validated in real-world deployment.
(Devalla et al. 2022) [35]	To detect SQLi, NoSQLi, and malicious URL attacks using intelligent models, including ML and DL methods.	Proposed the mURLi tool integrating RF, KNN, XGBoost, AdaBoost, ANN, BiLSTM, and BERT; applied feature engineering and Synthetic Minority Over-sampling Technique (SMOTE) balancing.	99.84%	Extensive feature engineering, effective model comparison, strong BERT performance on textual inputs, and effective SMOTE application.	A small NoSQLi dataset reduced BiLSTM accuracy; BERT underperformed on a numeric-based malicious URL dataset.
(Garcia, et al.2024) [36]	To compare ML algorithms for SQLi detection in web microservices.	Trained SVM, RF, and DT using TF-IDF and CountVectorizer; deployed in a microservices-based architecture.	99%,	Realistic microservices deployment; balanced dataset; high detection performance; practical evaluation.	Limited prior studies on SQLi in microservices; challenges in handling complex query structures; not tested in real-time adversarial settings.
(Tang et al.2020)[37]	To develop a neural network-based model for accurate SQL injection detection using both statistical and sequential URL features.	Extracted 8 statistical features for MLP and used American Standard Code for Information Interchange(ASCII)-encoded sequences for LSTM; models trained using PyTorch.	99.67%	Proposed dual-model comparison using both MLP (fast, feature-based) and LSTM (sequential, scalable); real ISP data used.	LSTM showed lower accuracy and significantly higher detection time; ASCII encoding limited the distinction between characters.
(Ross et al.2018) [38]	Propose a multi-source data analysis system to improve SQL Injection detection accuracy.	Designed a vulnerable enterprise chat web application; captured HTTP and MySQL traffic using Snort and Datiphy appliances; generated normal and attack traffic (manual SQLi and SQLMap); used Waikato Environment for Knowledge Analysis (WEKA) with feature selection (Correlation-based Feature Selection(CFS) and Genetic Search); evaluated classifiers (JRip, J48, RF, SVM, ANN) on three datasets: Webapp, Datiphy, and Correlated.	98.06%	Multi-source data improves detection, systematic feature selection, detailed time and performance metrics.	Simulated dataset (not real-world traffic); ANN is accurate but slow; no generalization to other attack types yet.
(Daramola et al.2025) [39]	Proactive detection and classification of malicious SQL queries.	Collected & preprocessed 88,213 labeled queries; engineered features (entropy, keywords, tokenization); trained and compared RF, MLP, SVM, NB.	98.4%,	Very large, diverse dataset; end-to-end pipeline; comparative evaluation of four ML models; open data.	No hyperparameter tuning; limited to SQLi; not evaluated on non-SQLi attacks or real-time deployment.

(Muhammad et al.2022) [40]	To implement predictive analytics for detecting and classifying SQL injection attacks using ML classifiers.	Multiple ML algorithms (LR, Sequential Minimal Optimization (SMO), J48, Instance-Based K (IBK), Stochastic Gradient Descent (SDG), NB, and Bayesian Network Classifier(BNK)) were trained and evaluated using WEKA with 10-fold cross-validation and hold-out. Features extracted from Damn Vulnerable Web Application(DVWA) access logs.	98.77%	Comprehensive metric-based evaluation; realistic simulation environment; robust model via hybrid detection.	Limited dataset size from DVWA; lacks real-time detection; no use of contextual embeddings.
(Triloka et al.2022) [41]	To detect SQLi attacks using ML and NLP-based feature engineering.	Tested five classifiers (SVM, KNN, LR, GB, NB) using text-based features and corpus processing with Python NLTK.	99.77%	Used dual datasets (training and challenge), NLP-based preprocessing, evaluated Time of Process, and accuracy.	Limited to specific corpus parameters; real-world generalizability not validated.
(Hirani et al.2020) [42]	To detect and compare SQLi attack detection performance across CNN and traditional ML algorithms.	Compiled a dataset with various SQLi types (union, blind, error-based); preprocessed and labeled payloads; evaluated DT, NB, SVM, KNN, and CNN classifiers.	94.84%	Wide dataset coverage, including multiple database(DB) types; strong CNN performance; thorough comparison with four ML models; robust metric analysis.	Architecture details of CNN not provided; dataset size not explicitly stated; real-time deployment and generalizability not evaluated
(Li et al 2019) [43]	To develop an effective SQL injection detection method tailored for intelligent transportation systems using DL.	Proposed an LSTM-based classifier combined with a SQL injection sample generation method to address data imbalance and overfitting. Word2Vec was used for feature embedding. The model was trained and evaluated on six datasets (DS1–DS6).	93.47%	Automatic feature learning via LSTM, effective positive sample generation reduced overfitting; strong generalization and high accuracy.	Generated samples may not cover all real-world SQLi variations; limited validation in diverse real-world deployment environments.
(Jothi et al 2021) [44]	To design an efficient SQL injection detection system using DL.	Used an MLP model with an embedding layer trained on SQLi queries from the Lib-Injection dataset and normal plain text. Data preprocessing included tokenization, stop-word removal, lemmatization, character filtering, word indexing, and padding.	98%	The model can detect all types of SQL injection attacks, does not rely on manually defined features, supports generalization, and can be extended to URL-based input detection.	The model does not use n-gram features, which limits its understanding of sentence context, potentially causing misclassification in rare cases.
(Ladole and Phalke 2016) [45]	To detect SQL Injection attacks and classify users (normal or attacker) based on queries.	Query tree generation using PostgreSQL logs; feature extraction via Fisher Score; classification via SVM implemented through the WEKA library.	94.12%	Utilized structural query trees for better detection; combined query analysis with user behavior classification.	Evaluation on a limited dataset lacks DL models; only binary classification.
(Hasan et al. 2019) [46]	To develop an intelligent system for detecting SQL injection attacks using ML techniques and	Evaluated 23 ML classifiers using MATLAB; selected the top five based on accuracy. Collected SQL statement datasets from w3schools and OWASP. Extracted numerical features for	93.8%	Comprehensive evaluation of multiple classifiers, integration of a user-friendly GUI, and high detection	Limited number of benign (non-injected) statements affected overall accuracy, relied on basic numerical features, and the system

	enhance detection accuracy via a GUI interface.	classification and implemented a GUI-based detection system.		accuracy for injected queries.	was not validated in a real-world environment.
(Sommervoll et al. 2024) [47]	Develop RL agents to simulate all types of SQL injection.	Trained Q-learning agents in a synthetic environment with 5 SQLi types and improved preprocessing.	96%	Covers all SQLi types; enables transfer learning; reduces expert input; realistic simulation.	Lower accuracy in uncertain cases; Q-learning scalability limits; synthetic setup lacks real-world complexity.
(Ismail et al., 2024) [48]	To identify a lightweight ML model for detecting cyber-attacks in blockchain-enabled industrial supply chains.	Compared NB, KNN, RF, DT, Bagging, Stacking, and CatBoost on the WUSTL-IIOT-2021 dataset using Mutual Information (MI) and Extra-trees (ET) for feature selection and undersampling for class balance.	-	Stacking outperformed in most metrics; CatBoost had highest precision; NB was fastest to train, DT fastest to predict.	Stacking had high training time; the study focused on only four types of attacks, and no DL models were included for comparison.
(Muduli et al. 2024) [49]	Develop two customized convolutional neural network models (SIDNet-1 and SIDNet-2) for high-accuracy SQL injection detection and prevention, with the ability to integrate into web protection systems.	Utilized datasets containing both benign and malicious SQL queries, converting them into numeric arrays (64×64×1), followed by a multi-layer CNN architecture (convolutional, MaxPooling, Dropout, Dense). SIDNet-2 includes additional Dropout layers after each convolutional and pooling to improve generalization. Performance compared with ML models (SVM, KNN, DT, NB) and previous DL models.	98.02%	Custom architecture tailored for SQL data, incorporation of generalization improvement strategies (Dropout), deployable in WAF or post-firewall, supports continuous learning to adapt to evolving threats.	Not tested extensively in real-world environments, did not evaluate other web attacks (e.g., XSS), performance depends on the quality and diversity of the dataset.
(Abaimov and Bianchi, 2019) [50]	Detect code-injection attacks (SQL and XSS) with DL.	CNN with a tailored pre-processing stage that encodes symbols as type and value pairs; local search to optimize network configuration; optional static signature check.	Up to 94%	Reduces training requirements through semantic encoding; modular and easily reconfigurable framework.	Relies on domain-specific pre-processing knowledge; performance varies with data and necessitates retraining when data distributions change.
(Kim et al. 2020) [51]	Apply DL (CNN-LSTM with normalized Unicode Transformation Format – 8-bit (UTF-8) encoding of spatial features) to detect unknown or obfuscated web attacks and to improve Snort rules in real time.	Run a signature NIDS (Snort) in parallel with a Traffic Analysis System; learn on HTTP payloads at the application layer; fast preprocessing via UTF-8 encoding; CNN-LSTM model tuned for scalability (Docker).	93%	Detects encoded or obfuscated payloads that Snort misses; assists in authoring and refining Snort rules (approximately five new rules per month) while avoiding duplicates; flexible, scalable design with daily labeling for continual retraining.	Requires analyst verification initially due to false positives; public benchmark sets are small, which can induce overfitting; focuses on HTTP and excludes Secure Sockets Layer(SSL) and User Datagram Protocol (UDP); depends on high-quality labeled data.

The fifty studies reviewed show how SQL injection detection has changed from traditional machine-learning pipelines with hand-crafted features to deep and hybrid architectures that can learn contextual or hierarchical representations directly from payloads. Classical methods, mainly using BoW, TF-IDF, or similar methods like ITFIDF with classifiers like SVM, RF, and LR, still work well in controlled settings [9, 15, 24, 25], but they are often specific to one dataset and can't handle inputs that are hidden or meant to trick the system. Deep learning techniques, especially CNN, Bi-LSTM, and their hybrids [13, 21, 27, 34], can extract features and model sequences better, but they may need more computing power, which makes them less useful in real time. Contextual embeddings, such as BERT-based variants [5, 12, 33, 35], improve semantic resilience but still have problems with memory efficiency and inference latency.

From a deployment perspective, only a subset of work transitions from offline experiments to operational prototypes. Examples include Flask-based services [14, 15], and firewall or edge-device integrations [10, 12, 49], highlighting a persistent gap in real-world validation. Similarly, explainability via SHAP or LIME is present in select studies [16, 17], but remains underutilized, reducing auditability in security-critical contexts. Data-related constraints are also prevalent, with many evaluations relying on small, single-source, or synthetic datasets [26, 31, 35], undermining generalizability. Furthermore, multi-attack detection frameworks remain rare, with most solutions narrowly scoped to SQLi [42, 44, 47].

These patterns underscore the primary gaps addressed by the present work: employing enriched and multi-source feature representations to improve robustness across diverse payload formats, integrating deep contextual modeling with resource-aware design to balance accuracy and deployment feasibility, embedding explainability into the detection pipeline for enhanced transparency, and extending detection capabilities to broader web-attack families within a unified, real-time framework.

4. Results and Discussion

The included studies report high accuracies under controlled evaluations across classical ML pipelines, deep and hybrid architectures, and contextual-embedding approaches. Classical NLP and ML baselines can be strong: TF-IDF and other NLP features with NB and SVM or ensembles reach 96–99% in several works, including 99.77% with NLP-based preprocessing and multiple classical classifiers, and 98.15% with BoW and RF against four ML baselines [11]

Deep-learning advances show comparable or higher performance while reducing manual feature engineering. Pure CNN detectors achieve 98.16% and 99.50% on large or real-traffic datasets [27, 28], whereas a CNN–BiLSTM hybrid reaches 98% on labeled queries [13]. A lightweight CNN with multi-head self-attention attains 98.98% with 69,269 parameters and fast inference suitable for edge devices [12].

Contextual and sentence-level semantics further improve robustness. A BERT–LSTM hybrid reports 97.3% on HttpParams [32]; synBERT reaches 99.74% with semantic learning [33], and a TextCNN followed by Bi-LSTM with an attention mechanism exceeds 99.57% [34]. Comparative work finds that contextualized embeddings deliver over 99% accuracy with better calibration and reduced training time, although they require more memory and typically exclude the pretraining cost [5].

Evidence of practical deployment exists but remains limited. Examples include a Flask-based ensemble system [15]. , microservice-oriented deployment with TF-IDF and CountVectorizer [36], and a two-layer firewall combining pattern matching with ML [10], The lightweight attention model targets edge scenarios [12]. However, many studies still lack online evaluation or adversarial testing.

On explainability, one ensemble study integrates SHAP and LIME with 99.50% accuracy, supporting auditability, although such tooling is not yet widespread [17].

Recurring limitations include small or single-source datasets, scarce external or real-time validation, and computational demands for some deep models. Reported issues include limited or simulated data, missing dataset details, resource requirements, or slightly lower performance on certain malicious subsets [12, 15, 41].

Ranking summary. Contextual-embedding and BERT-hybrid methods generally top accuracy (often 99%) [5, 33, 34]; CNN and BiLSTM hybrids follow closely (98–99.5%) [13, 27, 28]; classical TF-IDF and BoW pipelines provide strong baselines (96–99%) with lower complexity[9, 11, 25].

Accuracy complexity trade-off. Contextual embeddings improve calibration but increase memory footprint [5]; lightweight attention models trade a minor accuracy margin for fast edge-side inference [12].

5. Conclusion

This review indicates that SQL injection detection attains high performance across three main families of approaches: classical pipelines based on textual representations, such as TF-IDF [9, 25, 36]; deep and hybrid models built on convolutional and recurrent networks, including CNN and CNN–BiLSTM [13, 27, 28]; and contextual-embedding and modern language–representation methods [5, 33, 34].

In general, contextual approaches lead in accuracy but require more memory [5], lightweight attention- and CNN-based models are better suited to edge, low-latency scenarios [12], and classical pipelines remain strong, low-complexity baselines[11].

Deployability is evident yet still limited: examples include Flask-based prototypes [15]. , microservice-oriented deployments [36], and firewall or edge integrations [10, 12]. Nevertheless, live-traffic evaluation and adversarial testing remain scarce [10, 12, 15] Explainability is only sparsely adopted [17], and dataset constraints together with heterogeneous protocols hinder external validity [12, 15, 41].

Operational notes. In terms of computational complexity, a lightweight attention model contains about 69,269 parameters with fast inference appropriate for edge devices [12]. Contextual embeddings generally increase memory requirements, and pretraining cost is typically not included in reported training time [5]; BoW representations can also inflate memory as vocabularies grow [5]. Regarding error profiles, one CNN study reported a notable number of false negatives on the malicious subset, underscoring the need for careful threshold calibration to balance missed attacks against false alarms [27]. On the resource side, deployments span traditional servers via Flask [15], cloud microservices [36], firewall-style integrations [10], and edge execution for lightweight models [12].

6. Limitations

While this literature review comprehensively synthesizes fifty peer-reviewed studies on SQL injection detection using ML and DL techniques, several constraints remain: In terms of the review's scope, it has to be noted that the literature review only covers studies published between the years 2015 and 2025, and only within the bounds of reputable, peer-reviewed journals, including IEEE, Springer, Elsevier, ACM, and MDPI. While some relevant indexed literature may be grey and non-published, which holds some relevant emerging perspectives, these have also been excluded. This review also focuses on SQLi specifically, including only studies where SQLi was the primary focus. Research concerning broader web-attack detection was left out, unless SQLi was explicitly considered, which may lead to the omission of relevant methodology found within adjacent fields, including Cross-Site Scripting (XSS) or Denial of Service (DoS). This review also relies on the measures provided within the studies. The review may be able to generate an analysis based on performance metrics and methodological attributes provided by authors, but any inconsistencies concerning reporting standards or dataset descriptions may impact the comparison of results across the studies. There is also the question of pace concerning the fast movement of AI-driven security research, where the relevant techniques or datasets may outpace the review's closing date, leading to revisions of some of the trends or gaps that have been identified.

7. Future Work.

Firstly, broadening inclusion criteria to incorporate high-quality but non-indexed sources, technical reports, and industry white papers to capture state-of-practice alongside academic research.

Second, extending the scope to multi-attack detection literature to identify architectures adaptable beyond SQLi.

Third, conducting a meta-analysis to statistically compare performance across studies and identify effect sizes of different feature extraction or model types.

References

[1] D. Chen et al., "SQL injection attack detection and prevention techniques using deep learning," J. Phys. Conf. Ser., vol. 2021, IOP Publishing, 2021.

[2] OWASP Foundation, "OWASP Top 10 – 2021: The ten most critical web application security risks," OWASP, 2021. [Online]. Available: https://owasp.org/Top10/.[Accessed: Jul. 25, 2024].

[3] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, 2015.

[4] U. Farooq, "Ensemble machine learning approaches for detection of SQL injection attack," Tehnički Glasnik, vol. 15, no. 1, pp. 112–120, 2021.

[5] J. Zulu et al., "Enhancing machine learning-based SQL injection detection using contextualized word embedding," in Proc. 2024 ACM Southeast Conf., 2024.

[6] I. S. Crespo-Martínez et al., "SQL injection attack detection in network flow data," Comput. Secur., vol. 127, p. 103093, 2023.

[7] M. Zivkovic et al., "Optimizing SQL injection detection using BERT encoding and AdaBoost classification," in Proc. 2nd Int. Conf. Innov. Inf. Technol. Bus. (ICIITB 2024), Atlantis Press, 2024.

[8] R. Zuech, J. Hancock, and T. M. Khoshgoftaar, "Detecting SQL injection web attacks using ensemble learners and data sampling," in Proc. 2021 IEEE Int. Conf. Cyber Secur. Resil. (CSR), IEEE, 2021.

[9] P. Pramono, R. D. Irawan, and A. A. Sari, "Comparative analysis of SQL injection attack classification using Naïve Bayes method and support vector machine (SVM)," 2024.

[10] A. S. S. Ahmed et al., "A hybrid approach to detect injection attacks on server-side applications using data mining techniques," in Proc. 3rd Int. Conf. Sustainable Technol. Ind. 4.0 (STI), IEEE, 2021.

[11] M. Ahmed and M. N. Uddin, "Cyber attack detection method based on NLP and ensemble learning approach," in Proc. 23rd Int. Conf. Comput. Inf. Technol. (ICCIT), IEEE, 2020.

[12] R.-T. Lo, W.-J. Hwang, and T.-M. Tai, "SQL injection detection based on lightweight multi-head self-attention," Appl. Sci., vol. 15, no. 2, p. 571, 2025.

[13] N. Gandhi et al., "A CNN-BiLSTM based approach for detection of SQL injection attacks," in Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE), IEEE, 2021.

[14] P. Ogini, E. Taylor, and N. Nwiabu, "A deep learning approach for the detection of structured query language injection vulnerability," Int. J. Inf. Secur., vol. 11, no. 5, 2022.

[15] R. Pallam et al., "Detection of web attacks using ensemble learning," Learning, vol. 3, no. 4, p. 5, 2021.

[16] A. G. Kakisim, "A deep learning approach based on multi-view consensus for SQL injection detection," Int. J. Inf. Secur., vol. 23, no. 2, pp. 1541–1556, 2024.

[17] T.-T.-H. Le et al., "Enhancing structured query language injection detection with trustworthy ensemble learning and boosting models using local explanation techniques," Electronics, vol. 13, no. 22, p. 4350, 2024.

[18] F. K. Alarfaj and N. A. Khan, "Enhancing the performance of SQL injection attack detection through probabilistic neural networks," Appl. Sci., vol. 13, no. 7, p. 4365, 2023.

[19] F. Y. Hernawan, I. Hidayatulloh, and I. F. Adam, "Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance," J. Eng. Appl. Technol., vol. 1, no. 2, 2020.

[20] B. Arasteh et al., "Effective SQL injection detection: A fusion of binary Olympiad optimizer and classification algorithm," Mathematics, vol. 12, no. 18, p. 2917, 2024.

[21] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model," Mathematics, vol. 11, no. 15, p. 3286, 2023.

[22] J. R. Tadhani et al., "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," Sci. Rep., vol. 14, no. 1, p. 1803, 2024.

[23] R. Bakır, "UniEmbed: A novel approach to detect XSS and SQL injection attacks leveraging multiple feature fusion with machine learning techniques," Arab. J. Sci. Eng., pp. 1–14, 2025.

[24] Y. Li and B. Zhang, "Detection of SQL injection attacks based on improved TFIDF algorithm," J. Phys. Conf. Ser., IOP Publishing, 2019.

[25] W. Zhang et al., "Deep neural network-based SQL injection detection method," Secur. Commun. Netw., vol. 2022, no. 1, p. 4836289, 2022.

[26] D. P. Purbawa et al., "An enhanced SQL injection detection using ensemble method," JUTI: J. Ilm. Teknol. Inf., vol. 21, no. 1, pp. 1–9, 2023.

[27] M. Shahbaz et al., "Evaluating CNN effectiveness in SQL injection attack detection," J. Comput. Biomed. Inform., vol. 7, no. 2, 2024.

[28] A. Luo, W. Huang, and W. Fan, "A CNN-based approach to the detection of SQL injection attacks," in Proc. IEEE/ACIS 18th Int. Conf. Comput. Inf. Sci. (ICIS), IEEE, 2019.

[29] M. Alshammari, "Deep learning approaches to SQL injection detection: evaluating ANNs, CNNs, and RNNs," in Proc. Int. Conf. Math. Stat. Phys. Comput. Sci. Educ. Commun. (ICMSCE 2023), SPIE, 2023.

[30] N. Thalji et al., "AE-Net: Novel autoencoder-based deep features for SQL injection attack detection," IEEE Access, vol. 11, pp. 135507–135516, 2023.

[31] Q. Li et al., "A SQL injection detection method based on adaptive deep forest," IEEE Access, vol. 7, pp. 145385–145394, 2019.

[32] Y. Liu and Y. Dai, "Deep learning in cybersecurity: A hybrid BERT–LSTM network for SQL injection attack detection," IET Inf. Secur., vol. 2024, no. 1, p. 5565950, 2024.

[33] D. Lu, J. Fei, and L. Liu, "A semantic learning-based SQL injection attack detection technology," Electronics, vol. 12, no. 6, p. 1344, 2023.

[34] H. Sun, Y. Du, and Q. J. A. S. Li, "Deep learning-based detection technology for SQL injection research and implementation," Appl. Sci., vol. 13, no. 16, p. 9466, 2023.

[35] V. Devalla et al., "Murli: A tool for detection of malicious URLs and injection attacks," Procedia Comput. Sci., vol. 215, pp. 662–676, 2022.

[36] E. Peralta-Garcia et al., "Detecting structured query language injections in web microservices using machine learning," in Informatics, MDPI, 2024.

[37] P. Tang et al., "Detection of SQL injection based on artificial neural network," Knowl.-Based Syst., vol. 190, p. 105528, 2020.

[38] K. Ross et al., "Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection," in Proc. 2018 ACM Southeast Conf., 2018.

[39] C. Y. Daramola et al., "Malicious query recognition using chosen machine learning techniques," SN Comput. Sci., vol. 6, no. 3, p. 281, 2025.

[40] T. Muhammad and H. Ghafory, "SQL injection attack detection using machine learning algorithm," Mesopotamian J. Cybersecur., vol. 2022, p. 5–17, 2022.

[41] J. Triloka, H. Hartono, and S. Sutedi, "Detection of SQL injection attack using machine learning based on

natural language processing," Int. J. Artif. Intell. Res., vol. 6, no. 2, 2022.

[42] M. Hirani et al., "A deep learning approach for detection of SQL injection attacks using convolutional neural networks," Dept. Comput. Eng., MPSTME, NMIMS Univ., Mumbai, India, 2020.

[43] Q. Li et al., "LSTM-based SQL injection detection method for intelligent transportation system," IEEE Trans. Veh. Technol., vol. 68, no. 5, pp. 4182–4191, 2019.

[44] K. Jothi et al., "An efficient SQL injection detection system using deep learning," in Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE), IEEE, 2021.

[45] A. Ladole and M. Phalke, "SQL injection attack and user behavior detection by using query tree, Fisher score and SVM classification," Int. Res. J. Eng. Technol., vol. 3, no. 6, pp. 1505–1509, 2016.

[46] M. Hasan, Z. Balbahaith, and M. Tarique, "Detection of SQL injection attacks: A machine learning approach," in Proc. Int. Conf. Electr. Comput. Technol. Appl. (ICECTA), IEEE, 2019.

[47] Å. Å. Sommervoll, L. Erdődi, and F. M. Zennaro, "Simulating all archetypes of SQL injection vulnerability exploitation using reinforcement learning agents," Int. J. Inf. Secur., vol. 23, no. 1, pp. 225–246, 2024.

[48] S. Ismail et al., "A comparative study of lightweight machine learning techniques for cyber-attack detection in blockchain-enabled industrial supply chain," IEEE Access, 2024.

[49] D. Muduli et al., "SIDNet: A SQL injection detection network for enhancing cybersecurity," IEEE Access, 2024.

[50] S. Abaimov and G. Bianchi, "CODDLE: Code-injection detection with deep learning," IEEE Access, vol. 7, pp. 128617–128627, 2019.

[51] A. Kim, M. Park, and D. H. Lee, "AI-IDS: Application of deep learning to real-time web intrusion detection," IEEE Access, vol. 8, pp. 70245–70261, 2020.