IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 December 2025, accepted 2 January 2026, date of publication 12 January 2026, date of current version 15 January 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3651991

RESEARCH ARTICLE

A Hybrid Deep Learning Model for SQL Injection Attack Detection

EMANUEL N. CASMIRY, RAMADHANI S. SINDE, (Member, IEEE), AND NEEMA M. MDUMA, (Member, IEEE)
The Nelson Mandela African Institution of Science and Technology, Arusha 23306, Tanzania
Corresponding author: Emanuel N. Casmiry (casmirye@nm-aist.ac.tz)

This work was supported in part by the Mkwawa University College of Education, a constituent college of the University of Dar es Salaam.

ABSTRACT An increasing number of web application services raises significant security concerns. Online access to these applications exposes them to multiple cyberattacks. The Open Web Application Security Project has consistently reported SQL injection attacks among the top 10 cyber threats for over a decade, highlighting the need for more effective detection and prevention strategies. Traditional detection methods, primarily signature-based, offer basic protection but struggle to identify new signatures embedded within web requests. An alternative involves Machine Learning (ML) and Deep Learning predictive analytics, which provide effective solutions for analyzing large datasets to detect and prevent SQL Injection Attacks (SQLIA). However, these models often face challenges such as high false alarm rates and low accuracy. This study proposes a hybrid Convolutional Neural Network–Long Short-Term Memory (CNN-LSTM) model for SQL injection detection. Five architectures were compared: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), hybrid CNN-LSTM, and hybrid CNN-GRU. Among them, CNN-LSTM achieved the best results, with 99.85% accuracy, 99.86% precision, 99.85% recall, and a 99.85% F1-score, while also recording the lowest false positive rate (0.06%) and misclassification rate (0.15%). CNN-GRU, CNN, and GRU followed closely, though GRU reported the highest false positive rate (0.16%). The standalone LSTM model performed slightly weaker, with 99.80% accuracy, 99.80% precision, 99.79% recall, and 99.80% F1-score, along with false positive and misclassification rates of 0.06% and 0.20%, respectively. Overall, the findings show that all models achieved near-perfect performance, with hybrid architectures outperforming single-architecture models.

INDEX TERMS Deep learning, natural language processing, SQL injection (SQLi), cyberattack, hybrid architectures.

I. INTRODUCTION

In this digital era, online services have replaced multiple traditional services. Data are now managed online and can be accessed from anywhere in the world. The large-scale adoption of digital technologies across organizations has enabled new ways of working, communicating, and managing organizational data [1]. The world has witnessed the widespread adoption of web applications across organizations [2]. With the rise of web application usage, cyber-attacks are becoming a bigger problem for both businesses and users [2]. In recent years, the number and variety of cyber-attacks have increased exponentially [2]. Web applications are confronted with different cyber-attacks, SQL injection being one of the most exploited attacks [3]. The Open Web Application Security Project (OWASP) is a worldwide recognized non-profit organization dedicated to improving software security. Its major purpose is to give developers, businesses, and security professionals information, tools, and resources for identifying, understanding, and mitigating web application vulnerabilities [2]. According to OWASP, SQLi has remained among the top ten security threats for over a decade [4].

Structured Query Language (SQL) is a standardized programming language utilized for accessing and manipulating data in the database [5]. The Structured Query Language injection (SQLi) attack is a prevalent and critical security

The associate editor coordinating the review of this manuscript and approving it for publication was Olarik Surinta.

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

vulnerability in web applications where an attacker manipulates a websites SQL query to gain unauthorized access to the database. SQLi involves injecting harmful SQL commands into the user input field, such as input forms, to gain unauthorized control over the database [6], [7]. SQLi attacks do not involve the development of additional malware, only the construction of structured query statements [8]. This has led SQL injection attacks to become one of the attacks attackers frequently use, with attack methods changing as website technology evolves [8]. With the continuous advancement of web technologies, the structure of the SQL language remains dynamic, accommodating various coding techniques. As a result, conventional detection approaches, such as blocklist filtering and rule-based detection systems, often fail to provide effective defense mechanisms.

The emergence of Artificial intelligence and machine learning has proven effective in identifying indicators of cybersecurity attacks. In recent years, researchers have actively worked on SQL injection detection utilizing both machine learning and deep learning techniques [2], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. Nevertheless, these models frequently exhibit large false alarm rates and lower accuracy. The hybrid architecture has proven efficiency in the detection of other cyberattacks, such as intrusion detection [19], [20]. However, in SQL injection detection, these hybrid models are less explored. It is crucial to explore a deep learning hybrid architecture that can enhance detection performance and reduce the false alarm rate. Using a hybrid architecture leverages the strengths of two or more models, ultimately enhancing the performance of the model.

The motivation for this study comes from the ongoing challenges faced by current SQL injection detection methods. Traditional signature-based and rule-based systems struggle to identify obfuscated and evolving attack patterns, while standalone deep learning models often have high false-positive rates and limited ability to generalize. These issues pose significant risks for organizations that depend on web applications for critical functions. Therefore, there is a pressing need for a more robust and adaptable detection model that can identify both local structural anomalies and long-term query dependencies. This leads to the development of a hybrid CNN–LSTM architecture supported by feature selection to enhance detection accuracy and decrease false alarms.

In Tanzania, threats of SQL injection in web applications are also reported to be high. Over 50% of high-severity SQL injection vulnerabilities were reported in 79 assessed Tanzanian e-government websites [21]. Additionally, in Tanzanian higher learning institutions, SQL injection threats, among others, are reported to be a serious concern [22]. Given these significant challenges, more efforts are necessary to strengthen web application security through advanced and modern detection mechanisms. To address these challenges, this paper proposes a hybrid Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM) model for detecting SQL injection attacks.

The scope of this study is limited to developing and evaluating a hybrid deep learning model capable of improving SQL injection detection accuracy, reducing false alarms, and enhancing generalization across diverse SQLi patterns. This research is important because SQL injection remains one of the most damaging attack vectors affecting web applications, with the potential to expose confidential data, disrupt essential online services, and compromise public-facing systems such as e-government platforms. Improving detection performance, therefore has direct implications for real-world applications, including web application firewalls, intrusion detection systems, and automated security monitoring tools. By achieving higher precision and lower false-positive rates, the proposed model can significantly reduce unnecessary security alerts while ensuring that harmful attacks are accurately identified, resulting in more secure, stable, and trustworthy online systems.

Firstly, significant features were selected using the Chi-square test, and sentences were filtered based on the selected features, creating a cleaner input. Next, the study utilized the CNN's ability to learn spatial data and local patterns from SQL queries, serving as a feature extractor. The extracted features were passed into an LSTM for learning sequential patterns within the queries. A comparative analysis was performed against other developed models (GRU, CNN, LSTM, and CNN-GRU) as well as existing models. This study aims to improve SQLi detection by employing a hybrid architecture to maximize key performance measures like as accuracy, precision, recall, and F1-score to reduce false positives and misclassifications. The research questions guided this study were as follows:

RQ1: Can hybrid deep learning models outperform standalone models in detecting SQL injection attacks?

RQ2: Does Chi-square feature selection improve model accuracy and reduce false positives in SQLi detection?

RQ3: How does the proposed CNN–LSTM model compare to existing state-of-the-art methods in SQLi detection? The remainder of this work is structured as follows: Section II contains information about the related works, Section III discusses the materials and methods utilized, Section IV displays the results and discussion, and Section V describes the conclusion and future works.

II. RELATED WORKS

Earlier methods for preventing SQL injection (SQLi) attacks mainly relied on simple programming practices, such as parameterized queries and input validation, aimed at sanitizing user inputs and lowering attack risks. While these approaches worked against basic SQLi, more sophisticated methods like blind, second-order injections and time-based were able to bypass these defenses and carry out malicious queries later in the process [3]. To counter these concerns, signature-based detection systems were established that use predefined attack patterns to discover malicious queries in real-time [3]. However, these systems struggled to detect new

or disguised assaults that didn't match known signatures, resulting in high false positive and negative rates [23].

Later, rule-based methods were established to analyze SQL query structures more intensely. However, these systems still experienced high false positives and had difficulty detecting subtle attack variations [24]. As SQLi techniques kept advancing, anomaly detection systems were adopted to identify deviations from normal query behavior. While effective at catching irregularities, their use in dynamic environments often caused excessive false alarms [25]. To improve detection accuracy, hybrid techniques that joined static code examination with dynamic execution were introduced, allowing evaluation of both query structure and runtime behavior. Still, their dependence on labeled datasets limited their effectiveness, especially since such datasets are often hard to obtain in real-world situations [26]. Heuristic techniques, like V1p3R, aimed to overcome this limitation by using error feedback to adapt detection processes dynamically. Despite their flexibility, these systems still faced challenges against highly complex and heavily disguised SQLi attacks [27].

Given these challenges, machine learning (ML) and deep learning (DL) approaches have been increasingly employed by researchers for improving SQLi detection. Uwagbole et al [28], utilized SVM models on a known SQLi patterns dataset, which contained SQL tokens and symbols and achieved an accuracy of 98.6% and F1-score of 98.5%. Similarly, Ross et al [29], built three datasets of SQLi attacks: one collected at the web application host, another at a device node between the web host and the MySQL database, and a third combining both sources. Using classifiers like JRip, J48, Random Forest (RF), SVM, and Artificial Neural Networks (ANN), they found RF to be the most effective, with 98.05% accuracy on the combined dataset. Tripathy et al. [11], focused on packet payloads and trained multiple supervised learning models, with RF achieving the best performance at 99.8%, followed by AdaBoost, Boosted Trees, Decision Trees (DT), and Stochastic Gradient Descent (SGD), all over 98.6%. Similarly, Deriba et al. [12] proposed a hybrid framework that combined statistical and dynamic features, achieving up to 99.2% accuracy. In comparison, their SVM and ANN models scored 96.8% and 98.5%, respectively.

Potinteu and Varga [30] achieved 96% accuracy but only 74% recall and an 81% F1-score, showing limited effectiveness in consistently detecting malicious inputs. This gap hints at issues with sensitivity or dataset imbalance. Jothi et al. [14] presented a more balanced and effective model, reaching 98% in accuracy and recall, with a 97% F1-score. Their findings indicate high level of consistency between predicted outcomes and actual classifications, highlighting the robustness of the approach. However, the performance still lags behind the state of the art. Falor et al. [13] recorded 94% accuracy and a 96% F1-score, but only 85% recall, which may limit its ability to detect all attack instances in real-world use. Chen et al. [10] outperformed others, with 98.57% accuracy, 99.22% precision, 97.95% recall, and a 98.58% F1-score, indicating an optimized detection system that accurately finds SQLi patterns and reduces false positives. Ravishankar et al. [16] demonstrated consistent performance across all metrics, with each at 91%, indicating a well-balanced and finely tuned model. Crespo-Martínez et al. [2] reported 97.23% accuracy, with both recall and F1-score at 97.3%. The lack of precise data limits full comparison, but high recall points to effective threat detection. R. Shakya et al. [15] achieved 95.55% accuracy and 99% recall, indicating strong detection sensitivity, though missing precision and F1-score data leave the overall performance assessment incomplete. Takyi et al. [17] achieved a score of 99.4% across all metrics, including accuracy, precision, recall, and F1-score, highlighting a highly effective and consistent detection system, likely enhanced by advanced feature selection or ensemble methods.

From the reviewed literature, most SQL injection detection models suffer from one or more limitations, including high false-positive and false-negative rates, poor generalization to obfuscated queries, limited feature selection strategies, and inconsistent performance across datasets. It has been observed that hybrid models are also effective in identifying cyber-attacks in web applications, and some have even outperformed single deep learning models [19], [20]. Nonetheless, hybrid approaches have been less explored in detecting SQL injection attacks. These gaps emphasize the need for a more robust, feature-optimized hybrid model, precisely the motivation for the CNN-LSTM framework developed in this study. Additionally, recent research highlights the importance of effective feature extraction and handling data imbalance in improving model performance. Ibnath et al. [31] demonstrated that selecting distinctive and informative features significantly enhances the accuracy of text-based classification models, supporting the use of Chi-square feature selection in this study. Similarly, Hasib et al. [32] provided a comprehensive review of strategies for addressing class imbalance problems, which is relevant to SQL injection datasets that often contain uneven distributions of normal and attack queries. These insights reinforce the methodological choices adopted in our work.

III. MATERIAL AND METHODS

The development workflow for detecting SQL injection attacks, illustrated in Figure 1, follows several key stages. It begins with data preparation and preprocessing, proceeds through the experimental setup and model training, and concludes with the evaluation of the models.

A. OVERVIEW OF THE PROPOSED METHOD

The proposed SQL injection detection framework consists of four main stages: (1) dataset acquisition and cleaning, including the merging of institutional and Kaggle data; (2) feature engineering using TF-IDF combined with Chi-square feature selection to retain significant terms; (3) sentence pruning to remove irrelevant tokens and reduce noise; and (4) training hybrid and standalone deep learning models under identical settings for fair comparison. This structured pipeline

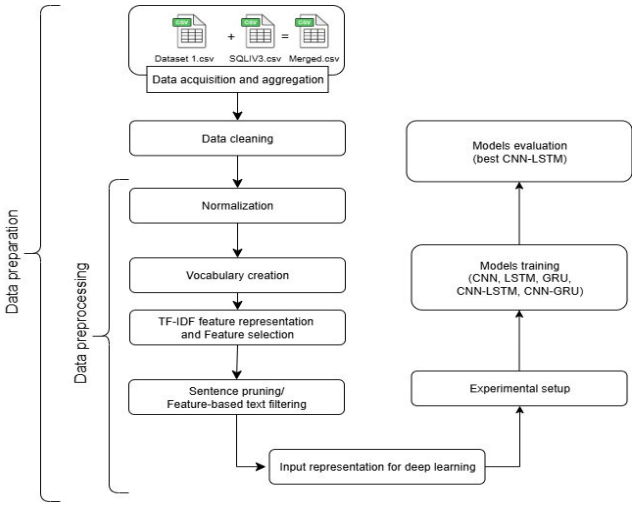

FIGURE 1. Model development flow.

ensures high-quality feature representation and robust model evaluation.

B. DATA PREPARATION

1) DATA ACQUISITION AND AGGREGATION

To train deep learning models, this study utilized two categories of datasets combined to form a merged dataset. The first dataset was created by collecting payloads from the Nelson Mandela Institution of Science and Technology (NM-AIST) Research Data Repository system during its development and testing phase. Malicious queries were generated through controlled SQL injection attacks using SQLMap, configured with comprehensive parameters including –tables, –passwords, –current-db, –roles, –columns, –dbs, –schema, –comments, count, –hostname, –users, –banner, –privileges, –current-user, –is-dba, –dump, –level =5, –risk =3, –random-agent, –batch, and answers = ''follow =Y''. SQLMap's supported techniques (–technique =BEUST), covering Boolean-based, Error-based, Union-based, Stacked, and Time-based blind injections, were executed to ensure comprehensive attack diversity. The second dataset, SQLiV3.csv from Kaggle, was incorporated to introduce additional variability in query structure and lexical distribution. Before merging, the two datasets were kept independent to prevent leakage, and their collection timelines and environments were verified to be distinct. Consistency checks were conducted to harmonize encoding formats, align labeling conventions, and identify structural differences. To quantify potential domain shifts between the datasets, a Jensen–Shannon Divergence (JSD) analysis was carried out on their TF-IDF feature distributions. The overall JSD of 0.5775 indicated moderate divergence, with malicious queries showing relatively low divergence (JSD = 0.1340) and benign queries exhibiting higher variability (JSD = 0.5320) [33]. These findings aligned with expectations, as benign inputs naturally vary across contexts while malicious payloads generated by automated tools tend to follow more consistent structural patterns [33].

Following validation, duplicate payloads were removed and the datasets were merged, resulting in a combined corpus of 41,573 benign and 40,365 malicious queries. The integration of real-application interactions, SQLMap-generated payloads, and community-sourced samples enhanced diversity, reduced the risk of overfitting, and ensured that the dataset reflected a wide range of practical SQL injection behaviors consistent with our previously validated methodology [33] The SQLiV3 dataset used in this study is a publicly available benchmark SQL injection dataset originally published on Kaggle by Hussain [34], and has been widely used in recent research as a reliable source of diverse SQLi payloads. Additionally, the custom NM-AIST dataset used in this study is documented in Casmiry et al. [33], where its collection process and characteristics were formally validated for SQL injection research.

2) DATA CLEANING

One of the key tasks before training the model is data cleaning. In this study, data cleaning was performed on the merged dataset, which involved activities such as handling missing values, removing duplicates, and correcting errors. Next, word segmentation and stop word removal were carried out. After cleaning, the final dataset included 34,367 benign and 30,746 malicious queries, totaling 65,113 samples. Within this cleaned dataset, the SQLiV3 subset contributed 19,061 benign and 11,375 malicious queries, while the custom NM-AIST dataset contributed 15,306 benign and 19,371 malicious queries. This distribution maintained diversity across sources while preserving a balanced dataset suitable for supervised learning. The resulting corpus provided consistent, high-quality representations of both benign and attack-related SQL patterns for model training and evaluation.

3) DATA PREPROCESSING

a: NORMALIZATION

Lexical items were standardized by converting them to lowercase, removing special characters, and managing word inflections. This ensured that words with the same meaning but different forms are treated consistently.

b: VOCABULARY CREATION

A vocabulary was created by collecting all unique lexical items from all documents. The unique index was assigned to each token in the vocabulary to represent a feature in the models' input matrix.

c: FEATURE REPRESENTATION AND SELECTION

Term Frequency–Inverse Document Frequency (TF-IDF) was used to convert SQL queries into numerical feature vectors that emphasize distinctive lexical patterns commonly associated with SQL injection behavior. To further refine the feature space, Chi-square feature selection was applied to identify terms most strongly correlated with each class label.

This methodological choice follows our previously published work [33], in which the Chi-square test demonstrated strong capability to reduce noise, improve classifier robustness, and stabilize feature distributions across SQL injection datasets. Consistent with that earlier study, the top 2,551 most informative features were retained out of a total of 49,607 extracted TF-IDF features, representing approximately 5% of the full vocabulary. This feature count corresponds to the optimal plateau region identified through systematic evaluation in the prior work, balancing dimensionality reduction with retention of discriminative signal.

Adopting the same optimized feature subset in this study ensures methodological continuity and provides the deep learning models with a compact, information-rich representation. This significantly mitigates feature-space inflation, enhances computational efficiency, and improves generalization by allowing the models to focus only on the most relevant SQL injection indicators. Chi-square feature selection was applied to reduce redundancy and mitigate the high noise levels commonly reported in SQLi datasets, which contribute to false alarms in prior studies.

d: SENTENCE PRUNING

To reduce noise and focus on relevant terms, the sentence was pruned by filtering the text based on selected features. This allowed the dataset to retain only the significant tokens selected by the chi-square test. The set of important terms (vocabulary set) was created from selected features. Each sentence was processed to keep only the words present in the selected set. This was done using a filter function that removed irrelevant words, leaving only the statistically significant terms. This step ensured that the input to the model was limited to the most informative features, thereby improving the efficiency and performance of the classification process. This step directly addresses a known limitation in existing work, insufficient removal of irrelevant tokens, which often leads to inflated feature spaces and reduced generalization. The mathematical procedures are presented in Equations (1)–(7).

Let

$$\boldsymbol{D} = \left\{\left(\boldsymbol{s}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{i=1}^{\boldsymbol{M}}, \tag{1}$$

be the corpus of $\boldsymbol{M}$ labeled SQL queries, where $\boldsymbol{s}^{(i)}$ is the i^{th} sentence (query) and $\boldsymbol{y}^{(i)} \in \{\mathbf{0}, \mathbf{1}\}$ and its label (0 = benign, 1 = SQLi).

Let

$$\boldsymbol{T} = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N\}, \tag{2}$$

be the full token vocabulary extracted TF-IDF feature representation (size N).

Apply Chi-square feature selection to obtain the reduced feature set of top k tokens:

$$\mathcal{V}_k = \boldsymbol{top}_k\left(\boldsymbol{\chi}^{\mathbf{2}}\left(\boldsymbol{t}_j, \boldsymbol{Y}\right)\right), \boldsymbol{t}_j \in \boldsymbol{T} \tag{3}$$

(In this study k=2551).

Represent a sentence **s** as an ordered token sequence:

$$\boldsymbol{s} = \left[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{\boldsymbol{n}_s}\right], \boldsymbol{w}_l \in \boldsymbol{T}, \tag{4}$$

Pruning function (token-level).

Define the indicator function for membership in the selected vocabulary:

$$\mathbb{I}_{\mathbf{V}_k}(\boldsymbol{w}) = \begin{cases} \mathbf{1}, & \boldsymbol{w} \in \mathcal{V}_{\mathbf{k}}, \\ \mathbf{0}, & \boldsymbol{w} \notin \mathcal{V}_k. \end{cases} \tag{5}$$

The pruned sentence $\boldsymbol{prune}_k(\boldsymbol{s})$ is obtained by retaining only tokens in $\mathcal{V}_k$, preserving the original order:

$$\begin{aligned} \boldsymbol{prune}_k(\boldsymbol{s}) &= \left[\boldsymbol{w}_l | \boldsymbol{w}_l \in \boldsymbol{s}, \boldsymbol{I}_{\mathcal{V}_k}(\boldsymbol{w}_l) = 1\right] \\ &= \left[\boldsymbol{w}_l : \boldsymbol{l} \in \{\mathbf{1}, \ldots, \boldsymbol{n}_s\}, \mathbb{I}_{\mathcal{V}_k}(\boldsymbol{w}_l) = \mathbf{1}\right] \end{aligned} \tag{6}$$

If all tokens are removed, the pruned query becomes the empty set.

$$\boldsymbol{prune}_k(\boldsymbol{s}) = \emptyset. \tag{7}$$

e: INPUT REPRESENTATION FOR DEEP LEARNING

After sentence pruning, the next step was to prepare the data suitable for training deep learning models. To support supervised learning, class labels were extracted from the dataset. Tokenization is the process of converting texts into small units called tokens. Deep learning models can then process these tokens. In this study, a word-level tokenizer was implemented. Deep learning models accept uniform input lengths across all samples, which is achieved by applying padding to tokenized text. The mathematical procedures are presented in Equations (8)–(13).

Let each pruned SQL query be represented as a sequence of tokens:

$$\boldsymbol{s}' = [\boldsymbol{w}_1, \boldsymbol{w}_2, .., \boldsymbol{w}_n], \tag{8}$$

where each token $\boldsymbol{w}_i$ is retained after pruning based on the Chi-square-selected vocabulary $\boldsymbol{V}_k$

A tokenizer function $\boldsymbol{\phi}(.)$ Maps each token to a unique integer index:

$$\boldsymbol{\phi} : \boldsymbol{V}_k \longrightarrow \{\mathbf{1}, \mathbf{2}, \ldots, |\mathcal{V}_k|\}. \tag{9}$$

Thus, each pruned sequence is numerically encoded as:

$$\boldsymbol{z} = [\boldsymbol{\phi}(\boldsymbol{w}_1), \boldsymbol{\phi}(\boldsymbol{w}_2), \ldots, \boldsymbol{\phi}(\boldsymbol{w}_n)]. \tag{10}$$

Because deep learning models require fixed-length input, each encoded sequence z is padded or truncated to length L:

$$\boldsymbol{x} = \boldsymbol{pad}_{\boldsymbol{L}}(\boldsymbol{z}), \tag{11}$$

where padding uses the value 0 to denote empty positions.

The final model input with corresponding class labels then becomes:

$$\boldsymbol{X} = \left\{\boldsymbol{x}^{\mathbf{1}}, \boldsymbol{x}^{\mathbf{2}}, \ldots, \boldsymbol{x}^{\boldsymbol{M}}\right\}, \tag{12}$$

$$\boldsymbol{y} = \left\{\boldsymbol{y}^{\mathbf{1}}, \boldsymbol{y}^{\mathbf{2}}, \ldots, \boldsymbol{y}^{\boldsymbol{M}}\right\}, \boldsymbol{y}^{(i)} \in \{\mathbf{0}, \mathbf{1}\}. \tag{13}$$

These numerical representations are then passed to the embedding layer of each deep learning model for training.

The dataset was partitioned into training (72%), testing (20%), and validation (8%) subsets using stratified splitting, ensuring that the class distribution (benign vs. SQLi) remained consistent across all partitions.

C. EXPERIMENTAL SETUP

Five deep learning architectures, Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), hybrid CNN-LSTM, and hybrid CNN-GRU, were chosen and set up for training in this study. Microsoft Windows 11 Home version 24H2, an x64-based PC, an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (8CPUs), about 2.40 GHz, 16GB of installed RAM, and a Colab Pro with a Tesla T4 GPU and 25 GB of RAM were the device characteristics used in these studies.

D. MODEL TRAINING

In this study, all models were trained under identical settings to ensure that performance differences reflect architectural strengths rather than inconsistencies in preprocessing or hyperparameter selection, a limitation frequently noted in earlier SQLi detection research. All models share a common foundational structure that begins with an embedding layer to represent input tokens as dense vectors. This is followed by feature extraction layers, which vary across models and include convolutional layers (Conv1D), recurrent layers such as LSTM and GRU, or hybrid combinations of both. Dropout layers are applied throughout the architectures for regularization, and fully connected dense layers with ReLU activation are used to learn higher-level feature representations before the final sigmoid output layer for binary classification. The consistent architecture backbone ensures that performance differences can be attributed primarily to the choice and arrangement of convolutional and recurrent layers rather than other components.

E. CONVOLUTIONAL NEURAL NETWORK-LONG SHORT-TERM MEMORY (CNN-LSTM)

The CNN-LSTM model is a deep learning-based system that chains two different architectures: CNN and LSTM. The CNN can detect local or spatial patterns in data, making it useful for spotting suspicious patterns in SQL queries. The LSTM is effective at learning sequences, which is important for understanding sentence context, especially when attack indicators are spread out in the query. A hybrid architecture was developed to integrate the complementary strengths of Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. An input layer, a one-dimensional convolutional layer, a pooling layer, an LSTM layer, and a fully connected layer make up the model, as shown in Figure 2. The ideal setup makes use of 128 convolutional filters with a kernel size of 7 and a pooling size of 2, with an embedding dimension of 100. A 64-unit LSTM layer captures sequential dependencies, and then a 64-unit dense layer follows. A dropout rate of 0.3 is used to reduce overfitting. The Adam optimizer is used in the training procedure, with a batch size of 32, a learning rate of 0.0001, and binary cross-entropy as the loss function. The model incorporates early halting with a patience of five and is trained for a maximum of thirty epochs to preserve the best-performing weights. This design was chosen specifically to address shortcomings in existing models by combining CNN's ability to detect structural irregularities with LSTM's strength in modeling long-term dependencies, both of which are crucial for detecting obfuscated SQL injection patterns. The CNN component extracts spatial patterns such as token sequences, operators, and structural irregularities within SQL queries, making it effective for identifying localized attack indicators. The LSTM layer complements this by capturing long-term dependencies and sequential relationships that appear in obfuscated attacks where malicious patterns may be spread across the query. By combining these strengths, the hybrid architecture provides enhanced robustness compared to using CNN or LSTM alone.

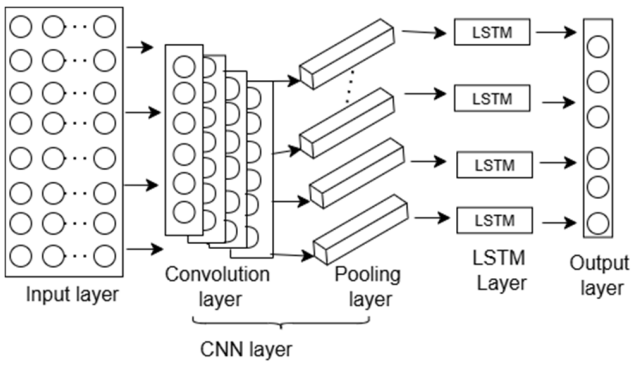

FIGURE 2. CNN-LSTM model architecture.

F. CONVOLUTIONAL NEURAL NETWORK (CNN)

CNN is a deep learning-based model introduced by Lecun et al. in 1998 [35]. It utilizes convolutional filters to identify local patterns or spatial dependencies within the input data and demonstrates good performance in image processing and natural language processing [36]. CNN has two key parts: the convolution layer and the pooling layer. Convolutional kernels are contained in each convolution layer, and their mathematical formula is given as formula (14). The features extracted by these convolutions are in very high dimensions. To reduce the feature dimension, a pooling layer is applied immediately after the convolution layer. This helps to resolve the challenge and minimize the expense of training the network.

$$\boldsymbol{l_t} = \boldsymbol{tanh}\left(\boldsymbol{x_t} * \boldsymbol{k_t} + \boldsymbol{b_t}\right), \tag{14}$$

Here, $\boldsymbol{l_t}$ symbolizes the output obtained after convolution, $\boldsymbol{x_t}$ refers to the input vector, $\boldsymbol{tanh}$ represents the activation function, whereas $\boldsymbol{k_t}$ and $\boldsymbol{b_t}$ correspond to the convolution kernel's weight and bias, respectively.

G. LONG SHORT-TERM MEMORY (LSTM)

LSTM is a network model introduced in 1997 by Schmidhuber [37]. This form of the Recurrent Neural Network (RNN) is specifically developed to overcome the vanishing gradient challenge, which traditional RNN struggles with. It is powerful in learning and remembering sequential patterns from input data [38], [39]. It has proven effective in a variety of natural language processing applications. LSTM provides a memory cell capable of carrying information across several time steps and controlling when to forget, update, or output information using a structure known as gates. Figure 3 depicts the three primary components of the LSTM memory cell: the forget gate, input gate, and output gate.

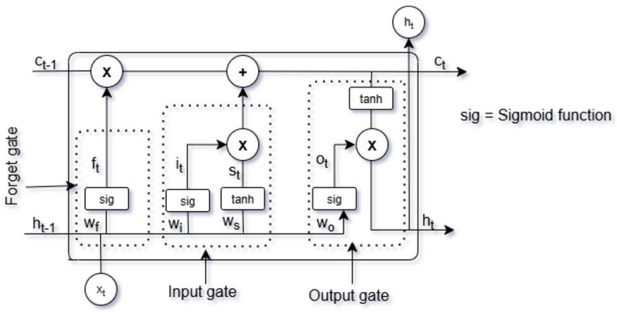

FIGURE 3. LSTM cell.

Cell state (c_t) represents the memory that flows through the network and is selectively modified by forget and input gates over time steps.

Forget gate (f_t) chooses what information to throw away from the previous cell state c_{t-1}. It is given by formula (15), where a value close to 0 indicates forget and close to 1 indicates retain.

$$f_t = \sigma\left(w_f . \left[h_{t-1}, x_t\right] + b_f\right) \tag{15}$$

The input gate (i_t) and candidate values (s_t) control what information to add. It is given by formulas (16) and (17).

$$i_t = \sigma\left(w_i . \left[h_{t-1}, x_t\right] + b_i\right) \tag{16}$$

$$s_t = tanh\left(w_s . \left[h_{t-1}, x_t\right] + b_s\right) \tag{17}$$

The updated cell state (ct) is given by the formula (18).

$$c_t = f_t * c_{t-1} + i_t * s_t \tag{18}$$

Output gate (o_t) determines the output of the LSTM cell (h_t) as given in equations (19) and (20)

$$o_t = \sigma\left(w_o . \left[h_{t-1}, x_t\right] + b_o\right) \tag{19}$$

$$h_t = o_t * \mathbf{tanh}\left(c_t\right) \tag{20}$$

In these equations, x_t denotes the current input, h_{t-1} is the previous hidden state, c_{t-1} corresponds to the previous cell state, f_t denotes the forget gate, it indicates input gate, s_t signifies the candidate cell state, o_t denotes the output gate, and h_t denotes the new hidden state.

H. HYPERPARAMETER TUNING PROCEDURE

A systematic hyperparameter tuning process was conducted to avoid ad-hoc parameter selection and to ensure optimal model performance. Using the validation set, a combination of grid search and iterative refinement was applied to evaluate different configurations of key parameters. Multiple convolution kernel sizes, including {2, 3, 4, 5, 7, and 9}, were tested to determine the most effective pattern width for capturing SQL token interactions, with a kernel size of 7 yielding the best validation accuracy. Embedding dimensions of {50, 100, 150, and 200} were explored, and an embedding size of 100 provided the best trade-off between expressive power and generalization. To mitigate overfitting, dropout rates ranging from 0.2 to 0.5 were evaluated, and a rate of 0.3 was selected based on its consistent reduction of validation loss. The learning rate was tuned within the range {1e-4, 5e-4, 1e-3} using the Adam optimizer, with 1e-4 demonstrating the most stable convergence; additionally, a learning rate scheduler was applied to reduce the rate by a factor of 0.5 when the validation loss plateaued for several epochs. All selected hyperparameters were further validated through five-fold cross-validation to ensure their stability across different data splits and to confirm that the model showed no signs of overfitting. This rigorous tuning procedure contributed to the robustness and reproducibility of the final hybrid CNN–LSTM architecture.

I. MODEL EVALUATION

Five evaluation metrics were employed during the evaluation phases to evaluate the effectiveness of the developed models. These measurements included precision, recall, accuracy, confusion matrix, and F1 score. These metrics provided a base for comparisons of these deep neural network models.

The model's accuracy is measured by how effectively it classifies the input query as normal or SQL injection. It can be calculated using the formula indicated in equation (21).

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{TN} + \boldsymbol{FP} + \boldsymbol{FN}} \tag{21}$$

Precision is used to measure the number of classified positive cases that were positive. High precision means fewer false positives. It is given by the formula shown in equation (22).

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}} \tag{22}$$

Recall is the evaluation metric that measures the sensitivity of a model in correctly identifying positive cases. Measures how many positive cases were correctly classified. It is given by the formula shown in equation (23).

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} \tag{23}$$

The F1-score serves as an evaluation metric that integrates both precision and recall, offering a more balanced evaluation of the model. It penalizes the extreme value more than the arithmetic mean to ensure that both precision and

recall contribute equally. It is given by the formula shown in equation (24).

$$F\,1-score = \frac{2*recall*precision}{recall + precision} \tag{24}$$

The confusion matrix is a specific layout table that is used to visualize the performance of the model. The confusion matrix appears as indicated in Table 1.

TABLE 1. Confusion matrix.

	Actually Positive	Actually Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

Where TP (True Positive), FP (False Positive), TN (True Negative), and FN (False Negative), True means the values were accurately classified, and false means the values were wrong classified.

Additionally, although these metrics provide a wide-ranging view of model performance, False Positive Rate (FPR) and Misclassification Rate were derived from the confusion matrix to provide additional interpretability.

The False Positive Rate (FPR) is the proportion of negatives falsely labeled as positive by a model. This can help understand the model's tendency to generate false alarms. It is obtained by the formula (25).

$$FPR = \frac{FP}{TN + FP} \tag{25}$$

Misclassification rate refers to the proportion of total predictions that were incorrectly classified which gives a straightforward measure of overall error. It is given by the formula (26).

$$Misclassification\ rate = \frac{FP + FN}{TP + TN + FP + FN} \tag{26}$$

Additional evaluation and validation methodology

To strengthen the reliability, robustness, and generalization assessment of the proposed approach, additional evaluation procedures were conducted beyond the standard train–validation–test split. These procedures include cross-validation analysis, comparison with a baseline machine learning model, independent test set evaluation, and qualitative error analysis. All evaluations were designed to ensure methodological consistency and fair comparison across models.

First, a five-fold stratified cross-validation strategy was employed to assess model stability and reduce dependency on a specific data split. The cleaned dataset was divided into five mutually exclusive folds while preserving the original class distribution between benign and SQL injection queries in each fold. In each iteration, four folds were used for training and one fold for validation, such that every sample was evaluated exactly once. All models were trained using identical preprocessing steps, feature representations, and hyperparameter settings across folds. Performance metrics including accuracy, precision, recall, F1-score, false positive rate, and misclassification rate were computed for each fold, and final results were reported as the mean and standard deviation across folds.

Second, to provide a strong non-neural reference point, an Extreme Gradient Boosting (XGBoost) classifier was implemented as an additional baseline model. The baseline was trained and evaluated on the same processed dataset and under the same stratified splitting strategy used for the deep learning models. This ensured that differences in performance could be attributed to model capability rather than data or preprocessing variations. The same evaluation metrics were applied to enable direct comparison between traditional machine learning and deep learning-based approaches.

Third, an independent test set evaluation was conducted to further examine the generalization capability of the proposed CNN–LSTM model. An external dataset (sqli.csv), not used during training, validation, or cross-validation, was employed for this purpose. The trained model was applied directly to the independent dataset without additional retraining. The same preprocessing and feature selection pipeline was used to ensure compatibility. This evaluation provides an additional safeguard against overfitting and optimistic performance estimates arising from reuse of training data.

Finally, a qualitative error analysis was performed to better understand model limitations and misclassification behavior. False positives and false negatives from the test set were extracted and examined at multiple preprocessing stages, including the original query, lemmatized form, and pruned representation. This analysis aimed to identify recurring error patterns and to assess the impact of preprocessing steps such as Chi-square feature selection and sentence pruning on classification decisions.

IV. RESULTS AND DISCUSSION

A. TRAINING AND VALIDATION ACCURACY AND LOSS

The models were evaluated on training and validation accuracy and loss. Hybrid models (CNN-LSTM and CNN-GRU) consistently outperformed standalone CNN, LSTM, and GRU models, achieving the highest validation accuracy of 99.94 and demonstrating the most stable generalization as shown in Figures 4, 5, 6, 7, and 8. The CNN-LSTM model also achieved the lowest validation loss (0.0029), indicating very precise predictions. The LSTM model had the lowest training accuracy (99.87%), while the CNN model converged the fastest due to its parallel feature extraction, but was slightly weaker in capturing long-term dependencies. Overall, hybrid architectures provided the best balance between feature extraction and sequential learning, leading to superior performance across metrics.

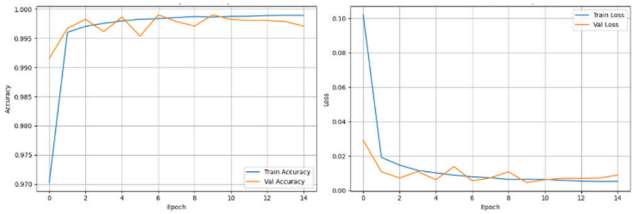

FIGURE 4. CNN model training and validation accuracy and loss.

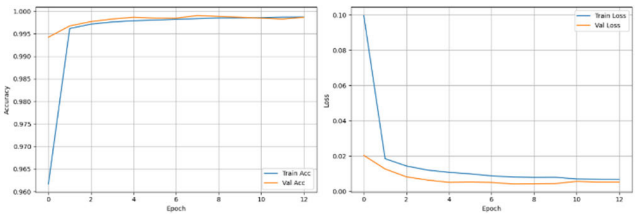

FIGURE 5. LSTM model training and validation accuracy and loss.

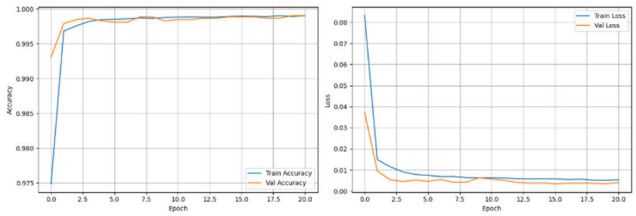

FIGURE 6. GRU model training and validation accuracy and loss.

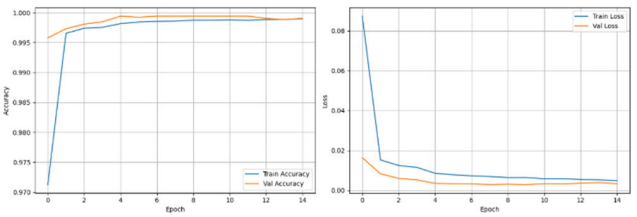

FIGURE 7. CNN-LSTM model training and validation accuracy and loss.

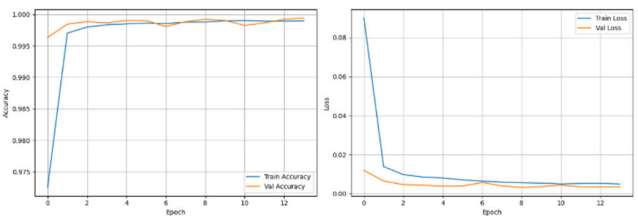

FIGURE 8. CNN-GRU model training and validation accuracy and loss.

B. CONVERGENCE BEHAVIOR AND STABILITY

To further distinguish model behavior beyond overlapping accuracy curves, we analyze convergence behavior and training stability using epoch-level validation statistics as shown in Table 2.

Across all evaluated models, validation accuracy exceeded 0.995 after the initial training epochs, indicating rapid convergence for both standalone and hybrid architectures. However,

TABLE 2. Comparative summary of training convergence and validation performance across models.

Model	Best Validation Accuracy	Epoch (Best Val Acc)	Lowest Validation Loss	Epoch (Best Val Loss)	Restored Epoch	Training Accuracy at Restored Epoch
CNN	0.9990	7 / 10	0.0046	10	10	0.9988
LSTM	0.9990	8	0.0041	8	8	0.9981
GRU	0.9990	20	0.0035	16	16	0.9988
CNN-LSTM	0.9994	5–15	0.0029	8 / 10	10	0.9989
CNN-GRU	0.9994	14	0.0032	9	9	0.9989

differences in convergence stability and loss minimization are evident from the epoch-level statistics summarized in Table 2. The standalone CNN and LSTM models reached their peak validation accuracy earlier (epochs 7–10 and epoch 8, respectively) but exhibited relatively higher minimum validation loss values compared to the hybrid architectures. The GRU model showed a more gradual convergence pattern, achieving its highest validation accuracy at a later epoch (epoch 20) while minimizing validation loss at epoch 16, suggesting slower but steadier training dynamics. In contrast, the hybrid CNN-GRU and CNN-LSTM models achieved both higher validation accuracy (0.9994) and lower minimum validation loss (0.0032 and 0.0029, respectively), with earlier stabilization and reduced oscillation across epochs. Notably, the CNN-LSTM model maintained high validation accuracy over a wider range of epochs (epochs 5–15) while preserving the lowest observed validation loss, indicating more stable convergence. These results demonstrate that hybrid architectures provide improved training stability and more consistent optimization behavior compared to standalone CNN, LSTM, and GRU models.

C. MODELS CLASSIFICATION RESULTS

The CNN-LSTM model outperformed other models across most metrics, achieving the highest accuracy (99.85%), precision (99.86%), recall (99.85%), and F1-Score (99.85%), while also recording the lowest FPR (0.06%) and misclassification rate (0.15%). CNN, LSTM, and GRU models performed nearly equally with small marginal differences, although the LSTM model had the lowest accuracy (99.80%), recall (99.79%), and misclassification rate (0.20%), and the GRU recorded the lowest precision (99.81%) and highest false positive rate (0.16%), as shown in Table 3 and Figure 9. Additionally, CNN-LSTM and LSTM models recorded high true negatives (6930) and low false positives (4), whereas the GRU had fewer true negatives (6923) and more false positives (11). Meanwhile, the GRU recorded high true positives (6073) and the lowest false negatives (14), closely followed by the CNN-LSTM with true positives (6072) and false negatives (15). The LSTM model, however, had lower true positives (6025) and higher false negatives (22), as shown in Figure 10. The experimental results demonstrate

that hybrid architectures consistently outperform standalone models, confirming the hypothesis raised in the introduction that combining spatial and sequential feature learning enhances robustness against complex SQLi payloads. This performance boost directly addresses limitations identified in previous studies, such as high false-positive rates and limited ability to generalize across diverse attack patterns.

TABLE 3. Models classification results.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	FPR (%)	Misclassification rate (%)
CNN	99.81	99.81	99.80	99.81	0.12	0.19
LSTM	99.80	99.80	99.79	99.80	0.06	0.20
GRU	99.81	99.81	99.81	99.81	0.16	0.19
CNN-LSTM	99.85	99.86	99.85	99.85	0.06	0.15
CNN-GRU	99.84	99.84	99.83	99.84	0.07	0.16

A clear contrast emerges when comparing the proposed CNN–LSTM model to existing SQL injection detection approaches. While previous studies typically rely on either convolutional networks or recurrent architectures alone, the hybrid CNN–LSTM model integrates both local pattern extraction and long-term dependency modeling, enabling superior detection of obfuscated and structurally complex SQLi payloads. This explains the consistent improvement across all metrics, as shown in Table 3 and Figure 9. Unlike prior methods that struggle with high false-positive rates or limited generalization, the proposed model significantly reduces misclassification errors and achieves higher stability across validation metrics. This contrast highlights the architectural advantage of combining convolutional and sequential learning for SQLi detection.

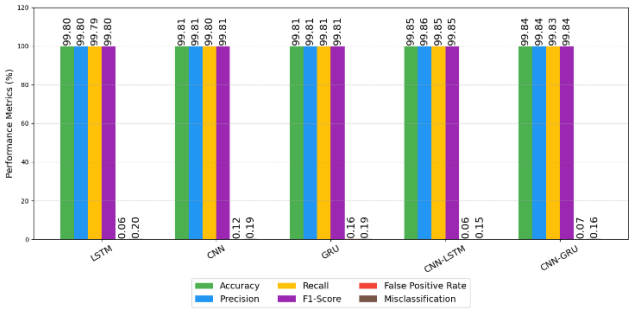

FIGURE 9. Models performance chart.

D. CROSS-VALIDATION RESULTS

To further assess the robustness and generalization capability of the models, a five-fold cross-validation experiment was conducted. The averaged results are presented in Table 4. As shown, all models maintained consistently high performance across folds, with only minimal variance

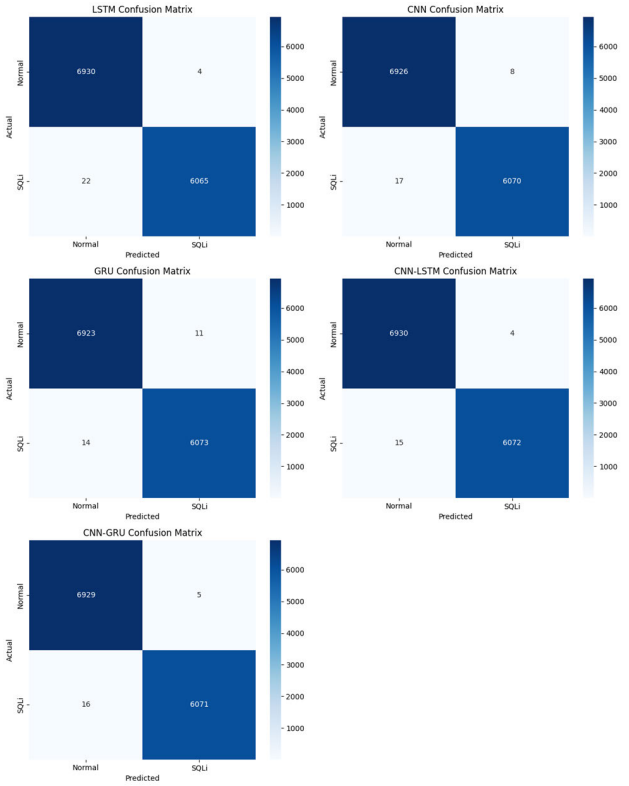

FIGURE 10. Model's confusion matrix. (A) LSTM. (B) CNN. (C) GRU. (D) CNN-LSTM. (E) CNN-GRU.

(±0.02–0.06), indicating stable learning behavior. The hybrid models continued to demonstrate the strongest performance, with CNN–LSTM achieving the highest averaged accuracy (99.83%) and F1-score (99.82%), followed closely by CNN–GRU. The standalone models (CNN, LSTM, GRU) showed slightly lower but still strong performance, aligning with the expectation that hybrid architectures better capture both local token patterns and long-range dependencies in SQL queries. The low variation observed across folds confirms that the reported results are not dependent on a specific train–test split and reflect genuine generalization rather than overfitting. Overall, the cross-validation outcomes reinforce the reliability and stability of the proposed CNN–LSTM approach for SQL injection detection.

E. ABLATION EXPERIMENTS

To assess the effect of chi-square feature selection on deep learning models, CNN, LSTM, and GRU were tested without applying feature selection. These models were evaluated using similar metrics used in other experiments; the models' classification results are shown in Table 5. The lower performance across all metrics was observed compared to the selected features in Table 3. This indicates that the Chi-square feature selection method boosts the models' performance. The ablation analysis was expanded to more clearly distinguish the effects of feature selection on

TABLE 4. Models cross-validation results.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	FPR (%)	Misclassification rate (%)
CNN	99.78 ± 0.03	99.84 ± 0.04	99.69 ± 0.05	99.76 ± 0.03	0.14 ± 0.02	0.22 ± 0.03
LSTM	99.77 ± 0.04	99.91 ± 0.02	99.60 ± 0.06	99.75 ± 0.04	0.08 ± 0.01	0.23 ± 0.04
GRU	99.79 ± 0.03	99.80 ± 0.03	99.73 ± 0.04	99.76 ± 0.03	0.15 ± 0.02	0.21 ± 0.03
CNN-LSTM	99.83 ± 0.02	99.91 ± 0.02	99.72 ± 0.04	99.82 ± 0.02	0.07 ± 0.01	0.17 ± 0.02
CNN-GRU	99.82 ± 0.03	99.90 ± 0.02	99.70 ± 0.05	99.80 ± 0.03	0.08 ± 0.02	0.18 ± 0.03

TABLE 5. Ablation experiments.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	FPR (%)	Misclassification rate (%)
CNN	99.69	99.69	99.68	99.68	0.25	0.31
LSTM	99.73	99.74	99.72	99.73	0.10	0.27
GRU	99.72	99.73	99.71	99.72	0.13	0.28

each model. By comparing performance with and without Chi-square–selected features, the study demonstrates how feature reduction improves generalization, reduces noise, and stabilizes model performance across architectures.

F. COMPARISON WITH ADDITIONAL BASELINE MODEL

To broaden the evaluation and include a strong machine learning baseline, the XGBoost (XGB) classifier was also tested on the same processed dataset. The XGB model achieved an accuracy of 99.28%, precision of 99.04%, recall of 99.13%, and an F1-score of 99.08%, with a false positive rate of 0.63% and a misclassification rate of 0.72%. Although XGBoost performed well, its results remained lower than those of the proposed CNN–LSTM model across all major performance metrics. The higher false positive and misclassification rates observed in the XGB model indicate challenges in capturing the structural and sequential complexity inherent in SQL queries. In contrast, the CNN–LSTM architecture demonstrated a stronger ability to generalize across diverse SQL injection patterns, resulting in more accurate and reliable detection outcomes.

G. INDEPENDENT TEST SET EVALUATION

To further validate the robustness and generalization ability the proposed CNN–LSTM model, an independent external dataset (sqli.csv) was used for testing. This dataset was not included in training, validation, or cross-validation processes. The model achieved slightly higher performance than the original train–test evaluation, recording an accuracy of 99.87%, precision of 99.95%, recall of 99.79%, and an F1-score of 99.87%. The false positive rate decreased to 0.05%, and the overall misclassification rate dropped to 0.13%. These improvements indicate that the CNN–LSTM model is capable of maintaining strong detection performance when exposed to previously unseen inputs. Although the independent test set (sqli.csv) was not part of the training data, it was also sourced from the Kaggle SQL injection repository, similar to the SQLiV3 dataset used during dataset merging. While the two datasets are independent, they may still share structural or linguistic characteristics associated with common SQL injection patterns generated in similar environments. As such, the strong performance on this dataset demonstrates good robustness but should not be interpreted as a complete measure of generalization across all domains. For broader validation, future work will evaluate the model using more heterogeneous sources, including real-world web server logs and enterprise-level security datasets, to better examine performance under diverse operational conditions.

H. ERROR ANALYSIS FOR PROPOSED MODEL

To better understand the limitations of the proposed model, an error analysis was conducted on misclassified samples from the test set. The analysis revealed two major categories of errors: false negatives (malicious queries incorrectly classified as benign) and false positives (benign queries incorrectly classified as malicious). Clear error patterns emerged, shedding light on how preprocessing, feature selection, and token reduction influenced model behavior.

1) FALSE NEGATIVES (MISSED SQL INJECTION ATTACKS)

Most false negatives corresponded to queries that became extremely short or empty after preprocessing. Examples include single-character inputs such as 6,), a, asc, and procedure. After lemmatization and pruning, these queries contained insufficient semantic information for the model to infer malicious intent.

Additionally, several obfuscated attacks such as hexadecimal payloads ($0 \times 776169\ldots$) and escape-sequence based injections (\x27UNION SELECT) were pruned to empty strings or incomplete SQL fragments. This occurred because Chi-square feature selection removed low-frequency but meaningful tokens, causing the model to lose critical attack indicators. Such errors highlight a known challenge in SQL injection detection: rare or obfuscated payloads often lack distinctive structural features after normalization.

2) FALSE POSITIVES (BENIGN QUERIES FLAGGED AS MALICIOUS)

False positives commonly involved benign SQL queries or non-SQL text that, after pruning, resembled known SQL injection structures. For instance, SELECT CONVERT(150, CHAR); was pruned to select convert (char), a pattern frequently observed in type-casting-based injections. Similarly, benign text corrupted by noise such as com/]Buy

Ciddds[/url]...was reduced to buy =, a structure resembling assignment-based attack payloads.

Another source of false positives was lemmatization, which occasionally mapped benign terms to high-risk SQL operators. For example, the legitimate token ors was reduced to or, a known indicator of Boolean-based SQL injection (OR 1=1). Furthermore, removal of table names in benign SQL (e.g., wp_users) produced truncated queries that resembled enumeration or exploitation probes.

3) INSIGHTS AND IMPLICATIONS

The error patterns indicate that (i) over-aggressive preprocessing may remove meaningful context, (ii) obfuscated payloads remain inherently difficult to classify, and (iii) certain benign queries become structurally similar to malicious ones after pruning. These findings suggest opportunities for improvement, such as retaining rare tokens, leveraging character-level embedding for obfuscated patterns, and refining pruning rules to preserve essential structural information.

I. COMPARISONS WITH EXISTING STUDIES

A comparative analysis with existing studies was conducted as shown in Table 6.

TABLE 6. Comparative results with existing studies.

Authors	Accuracy	Precision	Recall	F1-Score
Potinteu & Varga [30]	96%	90%	74%	81%
Jothi et al. [14]	98%	98%	97%	-
Chen et al. [10]	98.57%	97.95%	99.22%	98.58%
Falor et al. [13]	94%	85%	96%	-
Ravishankar et al [16]	91%	91%	91%	91%
Crespo-Martínez et al. [2]	97.23%	-	97.3%	97.3%
R. Shakya et al [15]	95.55%	99%	-	-
Takyi et al.[17]	99.4%	99.4%	99.4%	99.4%
Proposed model (CNN-LSTM)	99.85%	99.86%	99.85%	99.85%

The proposed model CNN-LSTM outperformed the compared existing studies in all metrics, recording the accuracy of 99.85%, precision of 99.86%, recall of 99.85, and F1-score of 99.85%.

J. DISCUSSION

The results address the shortcomings identified in existing research, particularly the need for models that generalize well to diverse SQLi patterns while minimizing false alarms. The hybrid CNN-LSTM architecture's ability to combine local pattern extraction with long-term dependency modeling directly contributes to reducing false positives and false negatives, a recurrent challenge in previous SQLi detection approaches. The experimental results show that all models performed very well in distinguishing malicious from normal SQL queries, with hybrid models outperforming single-architecture ones. This supports previous research that highlighted the advantages of combining different deep learning architectures for intrusion detection [19], [40]. However, unlike earlier studies such as Potinteu & Varga [30] (96% accuracy) and Ravishankar et al. [16] (91% accuracy), the proposed CNN-LSTM model achieved a significantly higher accuracy of 99.85%. This indicates that hybrid models, especially those blending convolutional and recurrent layers, can better capture both spatial and sequential patterns in SQL injection data. Interestingly, the single deep learning models (CNN, LSTM, GRU) performed at nearly the same levels (99.80%-99.81%), suggesting that both convolutional filters and recurrent structures contribute equally to feature learning in this dataset. Nonetheless, the hybrid CNN-LSTM model maintained a clear edge by combining these strengths, reducing the false positive rate to 0.06%. Compared to CNN-GRU, which achieved 99.84% accuracy, the CNN-LSTM demonstrated slightly better generalization, likely because LSTM units are more effective at modeling long-term dependencies than GRUs. The confusion matrices further confirm these performance differences. All models showed strong classification ability, but subtle variations existed. For example, the LSTM model misclassified more malicious queries as benign (22 FN) compared to CNN-LSTM (15 FN) and CNN-GRU (16 FN), indicating that hybrid models are more effective at lowering false negatives. This is especially crucial in security contexts, since undetected attacks pose a higher risk than false alarms. Similarly, CNN alone produced slightly more false positives (8 FP) than the hybrid models (4 FP for CNN-LSTM and 5 FP for CNN-GRU), reflecting its weaker ability to distinguish benign queries. These findings align with the lower misclassification and false positive rates seen in the quantitative results for the CNN-LSTM model. The role of feature selection was also clear. In ablation tests without feature selection, accuracy ranged from 99.69% to 99.73%, but with feature selection, the hybrid models exceeded 99.84%. This emphasizes the importance of choosing relevant input features to reduce noise and improve learning. Overall, these findings highlight two key points. First, hybrid deep learning architectures are very effective for SQL injection detection, achieving top results that outperform recent methods like Chen et al. [10] (98.57%) and Takyi et al. [17] (99.4%). Second, with very low misclassification (0.15%) and false positive rates (0.06%), the CNN-LSTM model is highly practical for real-world use, where minimizing false alarms is essential for system reliability. The confusion matrix analysis further supports this, showing that CNN-LSTM not only maximizes accuracy but also reduces critical security errors (false negatives), offering a robust and dependable solution for SQL injection detection. Overall, the findings confirm that the limitations highlighted in the introduction, high error rates, inconsistent performance, and limited use of hybrid models can be effectively mitigated by integrating convolutional and recurrent components with feature selection. This positions the proposed approach as a meaningful improvement over existing SQLi

detection techniques. The proposed CNN–LSTM model can be integrated into real-world security infrastructures such as web application firewalls (WAFs), intrusion detection systems (IDS), and API gateways. Its low false-positive rate reduces unnecessary alerts, making it suitable for use in automated monitoring environments. The model can operate as part of a backend classification service, scanning incoming SQL queries in real time and flagging suspicious payloads for further inspection. Beyond runtime deployment, the payload-oriented nature of the proposed approach also enables its application in pre-deployment security settings, where SQL queries generated during testing phases such as fuzz testing, symbolic execution, or automated test-case execution can be analyzed to identify potentially exploitable query patterns before system release. In this setting, the model complements static analysis techniques by providing behavioral validation of SQL query execution, thereby contributing to a more comprehensive SQL injection defense strategy that spans both pre-deployment and runtime environments.

V. CONCLUSION AND FUTURE WORKS

This paper addresses the problem of a high false alarm rate and lower accuracy in machine learning and deep learning models for SQL injection detection. The primary goal is to enhance SQL injection detection performance by using hybrid deep learning architectures. Key findings show that learning both local patterns and sequential patterns improves good model generalization, where hybrid models outperformed single-architecture models. Also, adding chi-square feature selection improved the performance by reducing the irrelevant and noise features. These findings indicate that for better generalization of deep learning models, utilizing a heterogeneous architecture can improve the performance and enhance injection detection. Also, although deep learning models have a high capacity for extracting features from queries, adopting a feature selection method such as chi-square helps reduce noise and redundant features, providing the model with cleaner input. This study is limited to the need for testing hybrid models in other SQL injection multiclass data and non-SQL injection datasets. Future direction will focus on the deployment of the best-performing model in a web application for real-time SQL injection detection and prevention. Thus, as science and technology continue to advance rapidly, cyberattack techniques are also evolving. Researchers must remain vigilant, exploring diverse approaches, including hybrid techniques, to combat these threats, thereby ensuring that information systems remain aligned with the principles of Confidentiality, Integrity, and Availability (CIA).

REFERENCES

[1] V. Sheth, U. Tripathi, and A. Sharma, "A comparative analysis of machine learning algorithms for classification purpose," *Proc. Comput. Sci.*, vol. 215, pp. 422–431, Jan. 2022, doi: 10.1016/j.procs.2022.12.044.

[2] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego-Delcastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, "SQL injection attack detection in network flow data," *Comput. Secur.*, vol. 127, Apr. 2023, Art. no. 103093.

[3] N. S. Dasari, A. Badii, A. Moin, and A. Ashlam, "Enhancing SQL injection detection and prevention using generative models," 2025, *arXiv:2502.04786*.

[4] OWASP Foundation. *OWASP Top 10—The Ten Most Critical Web Application Security Risks*. Accessed: Feb. 5, 2025. [Online]. Available: https://owasp.org

[5] D. Mitropoulos, P. Louridas, M. Polychronakis, and A. D. Keromytis, "Defending against web application attacks: Approaches, challenges and implications," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 2, pp. 188–203, Mar. 2019.

[6] F. Kausar, M. Rehman, and S. Hussain, "A comprehensive systematic review of SQL injection attack detection techniques," *Int. J. Tech. Phys. Problems Eng.*, vol. 15, no. 4, pp. 256–261, Mar. 2023.

[7] J. Xu, M. Ni, Z. Danjiang, and X. Yu, "Overview of SQL injection attack detection techniques," Tech. Rep., 2024, p. 225, doi: 10.1145/3640912.3640956.

[8] Q. Li, W. Li, J. Wang, and M. Cheng, "A SQL injection detection method based on adaptive deep forest," *IEEE Access*, vol. 7, pp. 145385–145394, 2019, doi: 10.1109/ACCESS.2019.2944951.

[9] J. M. Alkhathami and S. M. Alzahrani, "Detection of SQL injection attacks using machine learning in cloud computing platform," *J. Theor. Appl. Inf. Technol.*, vol. 100, no. 15, pp. 1–14, 2022.

[10] D. Chen, Q. Yan, C. Wu, and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *J. Phys., Conf. Ser.*, vol. 1757, no. 1, Jan. 2021, Art. no. 012055.

[11] D. Tripathy, R. Gohil, and T. Halabi, "Detecting SQL injection attacks in cloud SaaS using machine learning," in *Proc. IEEE IEEE 6th Int. Conf. Big Data Secur. Cloud (BigDataSecurity) Int. Conf. High Perform. Smart Comput., (HPSC) IEEE Int. Conf. Intell. Data Secur. (IDS)*, May 2020, pp. 145–150, doi: 10.1109/BIGDATASECURITY-HPSC-IDS49724.2020.00035.

[12] F. G. Deriba, A. O. Salau, T. M. Kassa, and W. B. Demilie, "Development of a compressive framework using machine learning approaches for SQL injection attacks," *Przeglad Elektrotechniczny*, vol. 2022, p. 98, Mar. 2022.

[13] A. Falor, M. Hirani, H. Vedant, P. Mehta, and D. Krishnan, "A deep learning approach for detection of SQL injection attacks using convolutional neural networks," in *Proc. Data Analytics Manag. ICDAM*, 2021, pp. 293–304.

[14] K. R. Jothi, S. Balaji B, N. Pandey, P. Beriwal, and A. Amarajan, "An efficient SQL injection detection system using deep learning," in *Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE)*, Mar. 2021, pp. 442–445.

[15] R. Shakya, D. N. S. Dharmaratne, and M. Sandirigama, "Detection of SQL injection attacks using machine learning techniques," in *Proc. Int. Conf. Electr., Commun. Comput. Eng. (ICECCE)*, Oct. 2024, pp. 1–6, doi: 10.1109/icecce63537.2024.10823462.

[16] N. Ravishankar, M. Raju, and N. C. Ravi, "Surgical striking SQL injection attacks using LSTM," Tech. Rep., 2022.

[17] K. Takyi, R.-M. O. M. Gyening, M. Kobinnah, M. A. Boateng, and S. Boadu-Acheampong, "Enhancing SQL injection detection with long short term memory networks in deep learning," *Int. J. Open Inf. Technol.*, vol. 13, no. 1, pp. 7–13, 2025.

[18] 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, Aug. 2022.

[19] M. Ali, N. Hussain, and A. Rahman, "Hybrid deep learning techniques for intrusion detection in modern web applications," Tech. Rep., 2023.

[20] T. Bakhshi and B. Ghita, "Anomaly detection in encrypted internet traffic using hybrid deep learning," *Secur. Commun. Netw.*, vol. 2021, Sep. 2021, Art. no. 5363750.

[21] N. Elisa, "Usability, accessibility and web security assessment of e-government websites in Tanzania," 2020, *arXiv:2006.14245*.

[22] F. N. Ntembo and R. Casmir, "The driving forces for the involvement of higher learning institution's students in cybercrime Acts. A case of selected higher learning institutions in Tanzania," *Eur. J. Theor. Appl. Sci.*, vol. 1, no. 4, pp. 911–922, Jul. 2023.

[23] A. Sadeghian, M. Zamani, and S. Ibrahim, "SQL injection is still alive: A study on SQL injection signature evasion techniques," in *Proc. Int. Conf. Informat. Creative Multimedia*, Sep. 2013, pp. 265–268, doi: 10.1109/ICICM.2013.52.
[24] W. G. J. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proc. Int. Symp. Secure Softw. Eng.*, 2006.
[25] M. Kiani, A. Clark, and G. Mohay, "Evaluation of anomaly based character distribution models in the detection of SQL injection attacks," Tech. Rep., 2008, p. 55, doi: 10.1109/ARES.2008.123.
[26] L. K. Shar, H. Beng Kuan Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 642–651, doi: 10.1109/ICSE.2013.6606610.
[27] A. Ciampa, A. Corrado, C. A. Visaggio, and M. Di Penta, "A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications," Tech. Rep., May 2010, doi: 10.1145/1809100.1809107.
[28] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1087–1090, doi: 10.23919/INM.2017.7987433.
[29] K. Ross, M. Moh, T.-S. Moh, and J. Yao, "Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection," in *Proc. ACMSE Conf.* New York, NY, USA: Association for Computing Machinery, Mar. 2018, pp. 1–8, doi: 10.1145/3190645.3190670.
[30] I.-C. Potinteu and R. Varga, "Detecting injection attacks using long short term memory," in *Proc. 16th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2020, pp. 163–169.
[31] M. H. Ibnath, K. M. Hasib, M. R. Parvez, and M. F. Mridha, "Enhancing multi-emotion detection in text: A comparative study of feature extraction techniques and machine learning models," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2025, pp. 1–6.
[32] K. M. Hasib, M. S. Iqbal, F. M. Shah, J. A. Mahmud, M. H. Popel, M. I. H. Showrov, S. Ahmed, and O. Rahman, "A survey of methods for managing the classification and solution of data imbalance problem," 2020, *arXiv:2012.11870*.
[33] E. Casmiry, N. Mduma, and R. Sinde, "Enhanced SQL injection detection using chi-square feature selection and machine learning classifiers," *Frontiers Big Data*, vol. 8, pp. 8–2025, Nov. 2025, doi: 10.3389/fdata.2025.1686479.
[34] S. S. Hussain. *SQL Injection Dataset [Data Set]*. Accessed: Mar. 31, 2025. [Online]. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset
[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.
[36] B. S. Kim and T. G. Kim, "Cooperation of simulation and data model for performance analysis of complex systems," *Int. J. Simul. Model.*, vol. 18, no. 4, pp. 608–619, Dec. 2019.
[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[38] B. Ghojogh and A. Ghodsi, "Recurrent neural networks and long short-term memory networks: Tutorial and survey," 2023, *arXiv:2304.11461*.
[39] C. Qin, L. Chen, Z. Cai, M. Liu, and L. Jin, "Long short-term memory with activation on gradient," *Neural Netw.*, vol. 164, pp. 135–145, Jul. 2023.
[40] I. S. Mambina, J. D. Ndibwile, D. Uwimpuhwe, and K. F. Michael, "Uncovering SMS spam in swahili text using deep learning approaches," *IEEE Access*, vol. 12, pp. 25164–25175, 2024, doi: 10.1109/ACCESS.2024.3365193.

EMANUEL N. CASMIRY received the Bachelor of Science with Education (B.Sc. Ed.) degree in informatics and mathematics from the University of Dar es Salaam, Tanzania, in 2020. He is currently pursuing the master's in information systems and network security with The Nelson Mandela African Institution of Science and Technology, Tanzania. He is also a Tutorial Assistant with the Department of Mathematics, Physics, and Informatics, Mkwawa University College of Education (MUCE), a constituent college of the University of Dar es Salaam (UDSM), Tanzania. His research interests include artificial intelligence, machine learning, and information systems security.

RAMADHANI S. SINDE (Member, IEEE) received the B.Sc. degree in engineering and technologies in telecommunication and the M.Sc. degree in engineering and technologies in telecommunication specialized in radio engineering, communication systems, and equipment from Moscow Technical University of Communication and Informatics, Russia, in 2009 and 2011, respectively, and the Ph.D. degree in information communication science and engineering with electronics and telecommunication engineering. Since 2014, he has been a Visiting Researcher at the University of Ghana, Carleton University, Oldenburg University, the Pedagogical University of Mozambique, and the Dhirubhai Ambani Institute of Information and Communication Technology. He is currently a Senior Lecturer with the School of Computation and Communication Science and Engineering, The Nelson Mandela African Institution of Science and Technology (NM-AIST). His research interests include the modeling and performance evaluation of wireless communication, embedded systems, and artificial intelligence of things enabler embedded systems. He is working on energy-efficient IoT-based systems using deep reinforcement learning for energy-optimized systems.

NEEMA M. MDUMA (Member, IEEE) is currently a Computer Scientist and a Senior Lecturer at The Nelson Mandela African Institution of Science and Technology (NM-AIST), specializing in artificial intelligence (AI) and machine learning (ML). She is also the Founder of an initiative called BakiShule, which promotes science, technology, engineering, and mathematics (STEM) to girls in secondary school in Tanzania. She also serves as a Principal Investigator in various projects that use AI and ML to address challenges in agriculture, health, education, and other sectors. Additionally, she serves as a reviewer for scientific journals and conferences and has contributed as a technical team member in drafting key national and international frameworks, including Tanzania Development Vision 2050, the Long-Term Perspective Plan (LTPP), the Five-Year Development Plan (FYDP), the National Guidelines for Artificial Intelligence in Education, the UNESCO's Artificial Intelligence Readiness Assessment Methodology (RAM), and the National Data Strategy.