WUJOSTEC

Dual Character–Symbol Encoding with CNN–BiGRU–Multi-Head Attention for SQL Injection and Cross-Site Scripting Detection

Mohammed Ibrahim*[1*], Fortune Ifeanyi Emineke[2], Nasiru Iliya[3] and Alhassan Adamu[4]
[*1,3]Department of Cyber Security, Nigerian Defence Academy, Kaduna
[2]Department of Cyber Security, AirForce Institute of Technology, Kaduna
[4]Department of Computer Science, Aliko Dangote University of Science and Technology, Wudil

Abstract

SQL injection (SQLi) and cross-site scripting (XSS) remain persistent threats to web applications because malicious intent is often expressed through short, obfuscated, and syntax-sensitive input strings. Conventional signature-based filters and rule-driven web application firewalls provide useful protection, but their dependence on predefined patterns limits their ability to generalize to novel payload variants. This paper presents a dual character–symbol encoding framework for classifying web inputs as SQLi, XSS, or normal traffic. The approach combines a character-index representation that preserves lexical order with a symbol tag representation that emphasizes security-relevant punctuation, delimiters, operators, and script markers. These representations are processed by a hybrid neural architecture composed of convolutional layers, bidirectional gated recurrent units, and multi-head self-attention. The convolutional component captures local attack motifs, the recurrent component models bidirectional payload context, and the attention component highlights multiple salient regions within an input sequence. The study uses a Kaggle SQLi–XSS mixed payload dataset reported as containing approximately 152,872 labelled payloads. The complete pipeline includes payload cleaning, duplicate removal, comment normalization, fixed-length sequence encoding, neural classification, and deployment through a Flask-based REST interface. The experimental evaluation reports an improvement in overall accuracy from 99.3% in a 20-epoch configuration to 99.7% in a 10-epoch configuration, indicating that shorter training reduced overfitting for this dataset. The work contributes a deployable low-level representation pipeline and a technically justified hybrid architecture for payload level web attack detection.

1. Introduction

Web applications routinely process user-controlled input through forms, URLs, cookies, headers, APIs, and third-party integrations. This input-driven design increases functionality and interoperability, but it also creates attack surfaces at the boundary between application logic, database query languages, and browser-executed markup. SQL injection (SQLi) and cross-site scripting (XSS) are among the most persistent web attack classes because both exploit improper handling of input strings. SQLi modifies the intent of database queries through crafted fragments such as quotes, Boolean predicates, comment delimiters, and query operators. XSS injects executable client-side content, commonly through script tags, event handlers, encoded delimiters, or malformed markup [1, 14, 9].

The detection problem is technically challenging because malicious payloads are often short, non-natural-language strings in which a small number of characters can determine intent. Attackers may also apply obfuscation through URL encoding, Unicode substitutions, comments, mixed case, whitespace perturbation, concatenation, nested delimiters, and syntactically equivalent query fragments. These characteristics make ordinary word-level text representations poorly suited to SQLi and XSS detection. Effective payload analysis requires representations that preserve fine-grained character order while emphasizing the symbolic structure of web attack syntax.

Existing defenses include secure coding practices, input validation, prepared statements, output encoding, content security policies, rule-based filters, and web application firewalls (WAFs). These controls are essential, but detection mechanisms based primarily on manually designed signatures can

fail when at-tacks deviate from known patterns or when benign inputs resemble suspicious strings. Classical machine learning methods reduce manual rule dependence by learning from labelled examples, but they often rely on engineered lexical and statistical features. Deep learning methods further reduce feature-engineering requirements by learning hierarchical sequence representations directly from payload strings [5, 6, 8].

The research gap addressed in this study lies at the intersection of representation, architecture, and deployment. Prior models often focus on a single attack family, use either character-level or manually engineered features, or report classification experiments without preserving the preprocessing semantics required for real-time deployment. CNN–GRU and attention-based models have shown promise for payload classification, but their effectiveness depends strongly on how low-level syntax is encoded and whether the trained detector can be invoked consistently outside the experimental environment [11]. This paper therefore examines a dual representation that combines raw character order with an explicit symbol channel and integrates it with a deployable CNN–BiGRU–multi-head attention classifier.

The objective is to design and analyze a supervised three-class classifier for SQLi, XSS, and normal pay-loads using low-level character and symbol information. The study makes five contributions. First, it formalizes SQLi/XSS detection as a sequence-classification problem over paired character and symbol encodings. Second, it introduces a dual preprocessing pipeline that preserves payload order while emphasizing security-relevant symbols. Third, it describes a hybrid CNN–BiGRU–multi-head attention architecture that combines local motif extraction, bidirectional context modeling, and attention-based salience estimation. Fourth, it evaluates the model using reported accuracy results and compares the architecture with representative rule-based, classical learning, deep learning, and attention-based approaches. Fifth, it demonstrates practical integration through a REST API that applies the same preprocessing semantics used during training.

2. Related Work

Web Application Firewalls (WAFs) provide an operational layer between clients and web applications by inspecting requests before they reach application logic. They can combine signatures, anomaly detection, protocol validation, and rate controls. Deep learning-enabled WAFs extend this paradigm by learning request-level representations from traffic data [8]. WAF deployment is valuable because it enables centralized enforcement, but WAF models face heterogeneous traffic, application-specific behavior, evolving payload distributions, and latency constraints. A payload classifier intended for WAF or API deployment is therefore computationally bounded and preserves its training-time preprocessing during inference.

Another approach is classical machine learning method for SQLi and XSS detection include support vector machines, random forests, naive Bayes, logistic regression, and ensemble learners [5, 6, 4]. These methods typically use manually engineered features such as token frequencies, payload length, punctuation counts, entropy, keyword indicators, n-grams, and regular-expression-derived attributes. They can perform well on curated datasets and are often easier to interpret than neural models. Their limitation is that feature engineering may omit subtle sequential dependencies, particularly when malicious intent arises from the order and interaction of delimiters, operators, comments, and executable markup.

[13,] and [12]. Presented Hybrid CNN–RNN models that extract local features before sequence aggregation. Their effectiveness depends on the representation of input strings: word-level tokenization may suppress critical punctuation, whereas character-level encoding preserves syntax but may underemphasize the special role of operators and delimiters.

Notable techniques like Attention mechanisms allow a model to assign greater weight to informative regions of a sequence. Multi-head attention extends this idea by allowing different heads to attend to different syntactic or semantic aspects of the payload. In SQLi and XSS detection, this is useful because a single payload may contain several suspicious regions, including an opening quote, a Boolean operator, a comment marker, and encoded markup. Transformer architectures rely heavily on self-attention and have become dominant in se-quence modeling [2]. However, transformer-only models may be computationally heavier than necessary for short web payloads and can require larger training corpora. The proposed model differs from prior CNN–BiGRU–attention studies by pairing the neural architecture with a dedicated symbol branch, thereby separating general character order from explicitly security-relevant punctuation structure.

The work of [1] that centered on rule-based filters have the capability of interpreting and detecting known pattern. However, the technique is limited in terms of generalization to obfuscation and unseen variants.

[8] with focus on WAF-based detection technique demonstrated practical screening and enforcement centralization but suffers from application specific traffic and latency constraint.

Classical Machine learning as in [5, 6, 4] have strong performance with engineered features and moderate data but its future design can omit sequential symbol interactions.

On the other hand, [11] explored the potentials in CNN–BiGRU–attention mechanism that combines local motifs, sequence context, and attention yet suffered from explicit deployment semantics.

Finally, is the Transformer family [2] that is explicitly Flexible self-attention and long-range dependency modeling but exhibit higher computational cost with larger data requirements.

3. Methodology

In this study, the methodology can be broken into the following:

3.1 Problem Formulation

Let $D = \{(x_i, y_i)\}^N$ denote a labelled dataset of web payloads. Each input x_i is a string and each label $y_i \in \{0, 1\}^C$ is a one-hot vector over $C = 3$ classes: SQLi, XSS, and normal traffic. The classifier estimates can be calculated using (1).

$$f\theta\,(xi) = \hat{p}i \in [0,1]^C, \tag{1}$$

where $\hat{p}_{ic}$ is the predicted probability that payload x_i belongs to class c.
The predicted class is given in (2)

$$\hat{y}_i = \arg\max_{c\in\{1,\ldots,C\}} \hat{p}_{ic}. \tag{2}$$

3.2 Dataset

The study uses the Kaggle file SQLInjection_XSS_MixDataset.1.0.0.csv, described in the experimental documentation as a 52.0 MB mixed SQLi–XSS payload dataset containing approximately 152,872 labelled examples. The dataset summary reports 72,816 benign payloads and 80,055 malicious payloads. The detection task is treated as a three-class classification

problem in which malicious samples are represented by SQLi and XSS subclasses, while benign samples are represented as normal traffic.

3.3 Data Cleaning and Normalization

The preprocessing pipeline removes incomplete records and exact duplicate payloads before partitioning the data. Deduplication before splitting is necessary because repeated payloads across training and testing partitions can inflate measured accuracy. After cleaning, comment normalization is applied to reduce superficial obfuscation. Let g(·) denote the normalization function. For each raw payload xi, the normalized payload is given (3)

$$\tilde{x}i = g(xi), \tag{3}$$

where g removes JavaScript single-line comments, JavaScript block comments, and SQL-style comment suffixes according to the preprocessing rules used in the implementation. The same normalization function is used during API inference.

3.4 Character Encoding

Let A be a finite alphabet containing whitespace, lowercase letters, digits, and selected punctuation. For a maximum sequence length L = 1000, each normalized payload is converted into a character-index sequence using equation (4)

zi ∈ $N^{L:}$

$$z_{ij} = \begin{cases} idx_A(X_{ij}) & x_{ij \in A} \\ 0, & x_{ij \notin A} \end{cases} \tag{4}$$

Payloads shorter than L are padded and payloads longer than L are truncated. This representation preserves the order of letters, digits, whitespace, delimiters, and operators.

3.5 Symbolic Encoding

Let $S \subset A$ denote the subset of security-relevant symbols, including quotes, slashes, angle brackets, equality signs, parentheses, semicolons, hash symbols, and comment markers. The symbol sequence $s_i \in N^L$ is defined in equation (5)

$$s_{ij} = \begin{cases} idx_S(X_{ij}) & x_{ij \in S} \\ 0, & x_{ij \notin S} \end{cases} \tag{5}$$

The symbol branch is designed to increase sensitivity to punctuation-level attack structure without discarding the broader character sequence.

3.6 Model Architecture

The model contains two parallel input branches. The character branch receives $z_i$and the symbol branch receives s_i. Each branch begins with an embedding layer: shown in (6)

$$E_i^{(z)} = Embz(z_i), \qquad E_i^{(s)} = Embs(s_i), \tag{6}$$

where the embeddings map discrete indices to dense vectors. Convolutional layers then extract local motifs using equation (7):

$$H_{cnn}^{(b)} = \sigma\left(conv\ 1D\left(E^{(b)}\right) + b\right), \qquad b\epsilon\{z, s\}, \tag{7}$$

where σ denotes a nonlinear activation. Bidirectional GRU layers process the convolutional

features in forward and backward directions: shown in equation (8)

$$H_{gru}^{(b)} = [\overrightarrow{GRU}(H_{cnn}^{(b)}); \overleftarrow{GRU}(H_{cnn}^{(b)})] \quad (8)$$

Multi-head self-attention is applied to model interactions among payload positions using equation (9)

$$Attention\ (QKV) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (9)$$

with multiple heads computed as in equation (10)

$$MHA(H) = Concat(head_1, \ldots head_h)W^0 \quad (10)$$

The attended character and symbol features are fused by concatenation in equation (11)

$$u_i = [a_i^{(z)}; a_i^{(s)}] \quad (11)$$

where $a_i^{(z)}$ and $a_i^{(s)}$ are the branch-level representations after attention and pooling. Dense layers transform u_i into class probabilities in equation (12).

$$\widehat{p_i} = softmax(W_C u_i + b_C) \quad (12)$$

3.7 Training and Evaluation

The model is trained by minimizing categorical cross-entropy using equation (13)

$$L_\theta = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log(\hat{p}ic)$$

Training uses the cleaned and encoded payload sequences, while validation data guide epoch selection. The experimental record compares a 20-epoch configuration and a 10-epoch configuration. Evaluation is reported using overall classification accuracy of equation (13).

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \lfloor(\widehat{y_i} = y_i) \quad (13)$$

with security interpretation focused on false positives and false negatives because both affect operational usefulness.

4 Theoretical and Computational Analysis

The classifier operates on bounded sequences of length $L = 1000$, which provides a fixed computational envelope for deployment. Character and symbol embedding layers require $O(Ld)$ operations per branch for embedding dimension d. One-dimensional convolution with kernel width k and m filters requires $O(Lkm)$ operations. A GRU layer with hidden dimension r requires $O(L_r^2)$ operations, with a constant factor for update, reset, and candidate gates. Multi-head self-attention requires $O(hL^2 d_h)$ operations for h heads and per-head dimension dh, making attention the dominant component for long sequences. The fixed input length bounds this cost and makes inference latency predictable for API deployment.

The dual representation improves the inductive bias of the classifier. The character branch models the complete payload sequence, including alphanumeric tokens and syntax. The symbol branch provides a sparse view of punctuation and delimiters that are disproportionately informative in SQLi and XSS. The fused representation allows the classifier to learn interactions between semantic fragments, such as keywords or tag names, and structural cues, such as quotes, angle brackets, and comment markers.

5 Results and Analysis

The experimental evaluation reports two training configurations. The 20-epoch model achieved

99.3% overall accuracy but exhibited greater overfitting behavior, including increased false positives in distinguishing normal inputs from SQLi-like payloads. The 10-epoch model achieved 99.7% overall accuracy and reduced the reported overfitting pattern. These results indicate that, for the evaluated dataset, additional training beyond 10 epochs did not improve generalization and instead increased error on ambiguous normal inputs. The study reported overall accuracy as depicted in Table 1.

Table 1: Overall accuracy for the proposed detector.

Configuration	Overall accuracy	Interpretation
CNN–BiGRU–attention, 20 epochs	99.3%	High accuracy with stronger overfitting behavior.
CNN–BiGRU–attention, 10 epochs	99.7%	Best reported configuration with improved generalization.
Reference baseline from the experimental record	95.29%	Lower reported accuracy than the proposed dual-encoding configuration.

The performance pattern is consistent with the structure of SQLi and XSS payloads. Local convolutional filters can capture frequent attack fragments, including Boolean predicates, tag delimiters, encoded operators, and comment markers. Bidirectional GRU layers then contextualize these fragments within the full payload, reducing dependence on isolated keywords. Multi-head attention further supports classification by enabling the model to emphasize multiple suspicious regions, which is important when a payload contains both delimiter manipulation and executable markup. The symbol branch provides additional discriminative information because many injection attacks are distinguished by punctuation sequences rather than by ordinary lexical content. From a security perspective, the reported improvement over the reference baseline is practically meaningful because even small changes in false-negative rates can affect exposure to exploitation. At the same time, high overall accuracy alone is insufficient for operational deployment unless monitored alongside classwise recall, false-positive behavior, and confidence calibration. The reported reduction in false positives at 10 epochs is therefore important because false positives can disrupt legitimate users and reduce trust in automated screening systems.

6 Deployment Architecture

The trained classifier is integrated into a Flask-based REST API and web interface. The deployment workflow receives an input payload, applies the same normalization and encoding procedures used during training, loads the trained model, computes class probabilities, and returns the predicted class with confidence scores. The API design is significant because preprocessing mismatch is a common source of degraded performance in deployed machine learning systems. By using the same character and symbol encoders at training and inference time, the system preserves semantic consistency between experimental evaluation and operational use. Operational deployment requires additional security engineering. Authentication restricts model access to authorized services. Rate limiting reduces abuse and denial-of-service risk. Input length controls enforce the fixed sequence length assumed by the model. Transport-layer encryption protects requests and responses. Logging and monitoring support incident analysis, confidence-drift detection, and model governance. These controls complement the classifier and are necessary when the detector is used in production web-security workflows.

7 Discussion

The model performs effectively on the evaluated dataset because its architecture matches the

structure of the detection problem. SQLi and XSS payloads are not ordinary natural-language documents; they are compact strings in which intent is encoded through local syntax, ordering, and symbolic manipulation. Character-level learning preserves the exact order of payload components, including punctuation that word tokenizers may remove or normalize. Symbol-level encoding adds an explicit inductive bias toward characters that commonly define injection semantics.

The CNN component is well suited to identifying local motifs such as quotes followed by Boolean predicates tag openings, script fragments, comment suffixes, and encoded delimiters. The BiGRU component captures contextual dependencies around those motifs, including whether a suspicious character appears in a benign context or participates in an attack pattern. Multi-head attention contributes by allowing the classifier to distribute focus across several payload regions rather than compressing the entire sequence into a single recurrent state. This is particularly useful for obfuscated payloads in which relevant evidence appears in separated positions.

The distinction from prior CNN–BiGRU–attention studies is the explicit dual representation. A single character stream can learn punctuation effects implicitly, but the symbol branch exposes these effects directly and allows the fusion layer to combine broad lexical context with concentrated syntactic evidence. This design is academically defensible because SQLi and XSS attacks are strongly associated with special characters, delimiters, operators, and markup boundaries. The deployable API further differentiates the study by connecting the model to an inference workflow rather than limiting the contribution to offline classification.

8 Limitations and Threats to Validity

The study is limited by its dependence on a single mixed SQLi–XSS dataset. Dataset-specific vocabulary, payload templates, label quality, and duplicate patterns may influence measured accuracy. Generalization to other applications, frameworks, languages, and adversarial payload distributions requires external validation on independent corpora. The reported evaluation emphasizes overall accuracy, which can obscure class-specific behavior when classes are imbalanced. Precision, recall, F1-score, confusion matrices, and calibrated confidence scores are important for operational decision-making.

Deployment also introduces constraints not captured fully by offline accuracy. Runtime latency, throughput, memory consumption, model loading behavior, and monitoring requirements affect production usability. Security controls around the API are necessary because an exposed detector can itself become an attack surface. Finally, the model identifies malicious-looking payloads but does not replace secure coding practices, parameterized queries, output encoding, content security policies, or comprehensive application security testing.

9 Conclusion

This paper presented a dual character–symbol encoding framework with a CNN–BiGRU–multi-head attention classifier for SQLi and XSS detection. The proposed representation combines complete character-order information with an explicit symbol channel for punctuation and delimiter patterns that are central to web injection attacks. The hybrid architecture integrates convolutional local feature extraction, bidirectional recurrent context modeling, and attention-based salience estimation. The reported evaluation results show 99.7% overall accuracy for the 10-epoch configuration, improving on the 99.3% reported for the 20-epoch configuration and the 95.29% reference baseline reported in the experimental record. The scientific significance of the work lies in aligning representation and

architecture with the syntactic nature of SQLi and XSS payloads. The practical significance lies in preserving preprocessing consistency. through a deployable REST API. Future research will extend the evaluation to independent datasets, adversarial transformed payloads, class-wise error analysis, ablation studies of the character and symbol branches, latency benchmarking, and explainability methods that support analyst interpretation of model decisions.

Author contributions

All authors must specify their roles in the research. Use structured contributions such as: Conceptualization, **Mohammed Ibrahim**.; Methodology, **Nasiru Iliya.**; Writing—original draft, **Fortune Ifeanyi Emineke**.; Writing—review and editing, **Nasiru Iliya.** Only contributors with substantial involvement should be listed as authors.

Funding Statement

Data Availability

Provide a brief data availability statement, such as:

- The data that support the findings are available from the corresponding author upon request.
- Not applicable, if no data were generated.

Acknowledgments

Authors may acknowledge technical, editorial, or other support not covered under authorship.

Conflict of interest

All conflicts of interest must be disclosed. If none exist, state: Non

- The authors declare no conflict of interest.

References

[1] OWASP Foundation, "OWASP Top 10: The ten most critical web application security risks," 2021. https://owasp.org/www-project-top-ten/

[2] A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

[3] S. O. Azarkasb and S. H. Khasteh, "Advancing intrusion detection in fog computing: Unveiling the power of Support Vector Machines for robust protection of fog nodes against XSS and SQL injection attacks," Journal of Engineering Research and Reports, vol. 25, no. 3, pp. 59–84, 2023.

[4] M. A. Azman, M. F. Marhusin, and R. Sulaiman, "Machine learning-based technique to detect SQL injection attack," Journal of Computer Science, vol. 17, no. 3, pp. 296–303, 2021.

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

[6] Bhanwarlal and I. Khan, "XSS and SQL Injection detection and prevention techniques: A review," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, pp. 53–60, 2022.

[7] B. R. Dawadi, B. Adhikari, and D. K. Srivastava, "Deep learning technique-enabled web application firewall for the detection of web attacks," Sensors, vol. 23, no. 4, 2073, 2023.

[8] F. Faisal and H. T. Elshoush, "Input validation vulnerabilities in web applications: Systematic review, classification, and analysis of the current state-of-the-art," IEEE Access, vol. 11, pp. 40128–40161, 2023

[10] C. Gupta, R. K. Singh, and A. K. Mohapatra, "GeneMiner: A classification approach for detection of XSS attacks on web services," Computational Intelligence and Neuroscience, 3675821, 2022.

[11] W.-C. Hsiao and C.-H. Wang, "Detection of SQL injection and cross-site scripting based on multimodel CNN combined with bidirectional GRU and multi-head self-attention," in Proceedings of the 5th

International Conference on Computer Communication and the Internet, 2023.

[12] A. Khan, M. Hussain, and R. Ali, "Deep learning-based approach for detecting SQL injection and cross-site scripting attacks," International Journal of Advanced Computer Science and Applications, vol.14, no. 1, pp. 122–130, 2023.

[13] M. Krishnan, Y. Lim, S. Perumal, and G. Palanisamy, "Detection and defending the XSS attack using novel hybrid stacking ensemble learning-based DNN approach," Digital Communications and Networks, 2022.

[14] G. E. Rodr´ıguez, J. G. Torres, P. Flores, and D. E. Benavides, "Cross-site scripting attacks and mitigation: A survey," Computer Networks, vol. 166, 106960, 2020.