Crimson Publishers
Wings to the Research

Research Article

LLM-Augmented Paired Code-Payload Dataset Feature Engineering for Web Vulnerability Detection

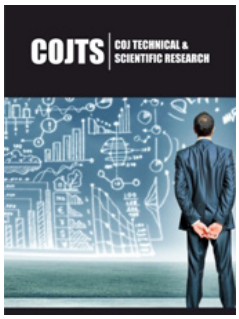
Mayar Khaled[1]*, Mohamed E Elhamahmy[2], Sherif Fdel[3] and Mohamed Abourizka[4]

[1]IT Security Department, Central Bank of Egypt, Egypt

[2]Chief Expert of Cybersecurity, National Telecommunications Regulatory Authority (NTRA), Egypt

[3]Arab Academy for Science, Technology and Maritime Transportation, Egypt

ISSN: 2643-7066



***Corresponding authors:** Mayar Khaled, IT Security Department, Central Bank of Egypt, Egypt

Submission: June 06, 2026

Published: August 27, 2026

Volume 6 - Issue 2

How to cite this article: Mayar Khaled*, Mohamed E Elhamahmy, Sherif Fdel and Mohamed Abourizka. LLM-Augmented Paired Code-Payload Dataset Feature Engineering for Web Vulnerability Detection. COJ Tech Sci Res. 6(2). COJTS. 000633. 2026.
DOI: 10.31031/COJTS.2026.06.000633

Abstract

Deep learning-based detection of SQL Injection (SQLi) and Cross-Site Scripting (XSS) vulnerabilities is significantly constrained by the limitations of existing datasets, which commonly separate malicious payloads from the vulnerable source code they exploit. This separation prevents models from learning the semantic and causal relationship between attack patterns and insecure code structures. To address this limitation, this study introduces a systematic six-stage pipeline for constructing xss_sqli_augmented.csv, a paired code-payload dataset containing 13,234 balanced samples enriched with 43 multi-dimensional features. The primary contribution of this work is the dataset construction methodology and feature engineering framework, rather than the development of a new detection architecture. To enhance coverage of modern attack scenarios, a context-aware hybrid synthesis approach based on GPT-4o is employed to generate obfuscated samples, with distributional consistency validated using two-sample Kolmogorov–Smirnov testing (mean p = 0.41 across 43 features). To ensure robust evaluation, SMOTE-based balancing is performed exclusively within training folds to prevent data leakage. Using an LSTM model as a benchmark classifier, the proposed paired representation achieves an F1-score of 0.989 [95% CI: 0.984–0.994], significantly outperforming payload-only representations (McNemar χ^2=14.3, p<0.001). Statistical analysis using the Wilcoxon signed-rank test confirms the superiority of the paired approach across ten evaluated models (W=55, p<0.001). Furthermore, cross-dataset evaluation on an independent OWASP-based corpus demonstrates strong generalization with an F1-score of 0.967 [95% CI: 0.958–0.976]. The proposed dataset, feature extraction framework, and complete pipeline are publicly released to support reproducible research in intelligent web vulnerability detection.

Keywords: Web application Security; Dataset construction methodology; Multi-dimensional feature engineering; LLM-augmented synthesis; SQL injection; Cross-site scripting; Vulnerability detection

Introduction

Web Application Firewalls (WAFs) remain a cornerstone of production web security, yet their effectiveness against SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks two of the most persistent threats in the OWASP Top Ten is severely constrained by the quality of the detection models they employ. In operational environments, particularly within financial services, e-commerce, and critical public-sector systems, even modest false-positive rates impose substantial costs: legitimate traffic is blocked, user experience deteriorates and security teams face alert fatigue. Contemporary WAFs, whether rule-based or learning-based, continue to operate largely in a reactive manner, relying on pattern matching against known payloads while struggling to understand the semantic and causal relationship between vulnerable code structures and the exploits that target them. This limitation stems directly from a fundamental bottleneck in the field: the absence of high-quality, semantically linked vulnerability datasets. Existing public datasets typically treat malicious payloads and vulnerable source code as independent artifacts. As a result, detection models cannot learn the contextual interplay between insecure code sinks (e.g., unsensitized cursor. execute ()

or inner HTML assignments) and the sophisticated, obfuscated payloads designed to exploit them. This fragmentation leads to elevated false-positive rates in production and poor generalization against modern evasion techniques such as multi-layer encoding and Unicode normalization bypasses [1].

Problem statement

Machine learning and deep learning approaches have shown promise for web vulnerability detection; however, their real-world deployment in expert security systems remains hampered by training data that lacks semantic pairing between code and payloads. Most available corpora suffer from class imbalance, shallow feature representations, and insufficient coverage of contemporary attack variants. Consequently, models trained on payload-only or code-only data fail to capture the causal mechanisms of exploitation, limiting their utility in applied WAF systems that must operate under strict low false-positive constraints while maintaining high recall against evolving threats.

Motivation

From a systems perspective, bridging this data gap is critical for advancing proactive, context-aware expert systems in web application security. A paired code-payload representation enables models to reason jointly about vulnerable code patterns and exploitation mechanics, potentially reducing operational false-positive rates and improving resilience to obfuscated attacks. This work addresses the bottleneck through systematic data engineering rather than proposing yet another deep learning architecture. By constructing a high-quality, feature-rich dataset and releasing the complete pipeline, we aim to support the development of more robust, deployable expert systems for web vulnerability detection.

Contributions

The primary contributions of this study are:

A. A systematic six-stage pipeline for constructing xss_sqli_augmented.csv, a balanced dataset of 13,234 paired code-payload samples enriched with 43 multi-dimensional features (structural, risk, pattern-matching, and semantic).

B. A context-aware hybrid synthesis approach combining template mutation with GPT-4o generation, rigorously validated through two-sample Kolmogorov-Smirnov testing to maintain distributional fidelity with real-world data.

C. Comprehensive multi-dimensional feature engineering and empirical benchmarking demonstrating that paired representations significantly outperform payload-only and code-only baselines (LSTM F1-score = 0.989 [95% CI: 0.984–0.994]; McNemar $\chi^2 = 14.3$, $p < 0.001$).

D. Strong cross-dataset generalization on an independent OWASP-based corpus (F1-score = 0.967) and full public release of the dataset, feature extraction code, generation prompts, and training scripts at https://github.com/mayarkh210/SQLi_XSS_ dataset to promote reproducible research in applied security systems.

Research questions

a) **RQ1:** Does integrating paired vulnerable code snippets with exploitation payloads via multi-dimensional feature engineering significantly improve detection accuracy and reduce false positives compared to isolated payload-only or code-only approaches?

b) **RQ2:** Can a hybrid template-mutation–LLM synthetic generation process, validated through Kolmogorov–Smirnov fidelity testing, effectively expand obfuscation coverage while preserving distributional consistency?

c) **RQ3:** What is the relative contribution of each feature category (structural, risk, pattern, semantic) to model performance in an applied detection setting?

d) **RQ4:** Does a model trained on the proposed paired dataset generalize effectively to independent, real-world vulnerability samples?

Related Work

To clearly expose the data-engineering gap this work addresses, recent literature is reviewed across three themes.

Payload-centric detection and dataset limitations

A significant body of research has focused on applying deep learning techniques directly to malicious payloads for the purposes of Web Application Firewall (WAF) evasion and attack detection. For example, Fang et al. [2] demonstrated the effectiveness of Convolutional Neural Networks (CNNs) in classifying obfuscated Cross-Site Scripting (XSS) payloads through the analysis of lexical and syntactic patterns. Building on this work, Hassan et al. [3] employed transformer-based architectures and reported high detection accuracy across known obfuscation variants. In the domain of adversarial attack generation, Zhang et al. [4] utilized Generative Adversarial Networks (GANs) to produce synthetic SQL Injection (SQLi) payloads for evaluating and stress-testing WAF resilience. Despite these advances, approaches that rely exclusively on payload-level features inherently operate without contextual awareness of the target application. As a result, such models function primarily as pattern-recognition systems and are unable to determine the specific vulnerable code sink being exploited. This limitation reduces their ability to generalize to emerging attack vectors that exploit previously unseen vulnerabilities within application code.

Code-centric static vulnerability discovery

An orthogonal research strand leverages ML to identify vulnerable source code prior to deployment. Li et al. [5] proposed VulDee Pecker, modelling program dependencies via code slices to identify vulnerable PHP patterns. LineVul [6] and Reveal [7] achieved strong results using Code BERT-based and ensemble approaches, respectively. Nevertheless, these methods completely ignore exploit mechanics: a model trained solely on code may identify a potential SQL sink but, without corresponding payload context, suffers from elevated false-positive rates and cannot verify exploitability against modern obfuscation.

Hybrid representations and dataset engineering

Recent research efforts have attempted to address these limitations through improved dataset construction and the integration of multi-modal feature representations. Sharafaldin et al. [8] developed a large-scale labeled intrusion detection dataset incorporating network-level HTTP traffic characteristics. Similarly, Arshad et al. [9] conducted a comprehensive review of web vulnerability detection approaches and highlighted that existing public datasets lack explicit and semantically meaningful associations between vulnerable code segments and their corresponding attack payloads. Nguyen et al. [10] proposed a detection framework combining TF-IDF representations with structural code metrics for XSS identification; however, their approach relied on unpaired data and did not incorporate vulnerability risk indicators or sufficient coverage of modern obfuscation techniques. Roy et al. [11], representing the closest prior work to this study, achieved an F1-score of 0.982 on payload-only SQL Injection and Cross-Site Scripting datasets using deep learning methods, establishing a strong baseline for payload-centric detection approaches. Nevertheless, existing studies have yet to introduce a balanced, multi-dimensional dataset that explicitly links vulnerable code snippets with their corresponding exploitation payloads, limiting the ability of detection models to learn the underlying relationship between vulnerabilities and attacks.

Gap analysis

Table 1 presents a quantitative gap analysis of representative prior work.

Table 1: Gap analysis of representative prior work.

Reference	Year	Core Approach	Key Limitation	Baseline?
[2,3]	2021, 2023	DL (CNN / Transformer) on raw payloads	Operates solely on payloads; ignores vulnerable code context	No
[4]	2022	GAN-based payload augmentation	Synthesizes payloads but lacks code-pairing and multi-dimensional feature enrichment	No
[8,9]	2018, 2021	Aggregated web-attack datasets	Fragments code and payload; lacks semantically linked code–payload pairs	No
[10]	2023	TF-IDF + structural features for XSS	Integrates features on unpaired data; lacks risk indicators and modern obfuscation coverage	No
[11]	2022	LSTM on SQLi/XSS dataset (SOTA, F1 = 0.982)	Payload-centric; lacks paired code context; shallow feature set	Yes
Proposed	2024	Six-stage paired pipeline + 43-D feature engineering	Addresses all limitations: AST-based pairing, hybrid LLM generation, K–S validation, within-fold SMOTE, CI-reported metrics, Wilcoxon test	—

Dataset Composition and Characterization

Before detailing the pipeline, the resulting dataset is characterized to enable practitioners to assess its real-world coverage independently of executing the pipeline.

Overall statistics

The xss_sqli_augmented.csv dataset contains 13,234 balanced code–payload pairs: 6,617 SQLi and 6,617 XSS. Of these, 8,240 pairs (62.3%) originate from real-world vulnerability databases (NVD, Exploit-DB, OWASP WebGoat), while 4,994 pairs (37.7%) are synthetically generated via the hybrid LLM pipeline (Section 4.3). Code snippets cover PHP (61%), Python/Django (22%), Node.js/JavaScript (12%), and Java/JSP (5%). Each pair is annotated with attack_type, sink_type, obfuscation_variant, is_synthetic, and a unique pair_id for provenance tracking. The complete dataset, feature extraction code, and training scripts are publicly available at https://github.com/mayarkh210/SQLi_XSS_dataset.

Sink-type distribution

Table 2 lists the ten most frequent dangerous sink types, mirroring the real-world attack surface.

Table 2: Ten most frequent dangerous sink types.

Sink Type	Attack Class	Language	Count	% of Class	Notes
cursor.execute()	SQLi	Python	1,842	27.80%	Primary Python SQL sink
Raw SQL string concat.	SQLi	PHP/Java	1,614	24.40%	Legacy pattern
prepare()/query()	SQLi	PHP	1,023	15.50%	PDO misuse
knex.raw()	SQLi	Node.js	714	10.80%	ORM escape bypass
Other SQL sinks	SQLi	Mixed	1,424	21.50%	—
innerHTML	XSS	JavaScript	1,980	29.90%	Primary DOM sink
document.write()	XSS	JavaScript	1,562	23.60%	Legacy DOM sink
eval()	XSS	JavaScript	1,104	16.70%	Script execution
dangerouslySetInnerHTML	XSS	JavaScript	877	13.30%	React-specific
Other DOM sinks	XSS	Mixed	1,094	16.50%	—

Obfuscation variant coverage

Covering modern evasion tactics was a primary design goal. Table 3 details the obfuscation variant taxonomy. Synthetic generation was deliberately targeted at underrepresented variants (multi-layer encoding and Unicode normalization bypasses) to prevent overfitting to canonical attack patterns.

Table 3: Obfuscation variant taxonomy and sample counts.

Obfuscation Variant	Count (Real)	Count (Synthetic)	Example Pattern
Canonical (no obfuscation)	3,210	0	SELECT * FROM users WHERE id='1'
URL encoding	1,102	420	%27 OR %271%27=%271
HTML entity encoding	842	380	<script>alert(1)</script>
Comment insertion	614	290	SE/**/LECT 1,2,3 FROM users
Case variation	520	210	SeLeCt * FrOm users
Multi-layer encoding	282	640	Double URL + HTML encode combined
Unicode normalization bypass	164	720	\u003cscript\u003e variants
String splitting / concatenation	304	332	CONCAT(0x53,0x45,0x4C)

Proposed Methodology

This section details the systematic six-stage pipeline that transforms fragmented, imbalanced vulnerability data into a cohesive, feature-rich dataset (Figure 1). The emphasis is firmly on data engineering and LLM-augmented synthesis rather than end-classifier architecture. The pipeline proceeds as follows: (i) AST taint-based pairing, (ii) multi-dimensional feature extraction, (iii) hybrid LLM-augmented synthesis, (iv) K–S statistical validation, (v) within-fold SMOTE balancing, and (vi) comprehensive ML/DL benchmarking.

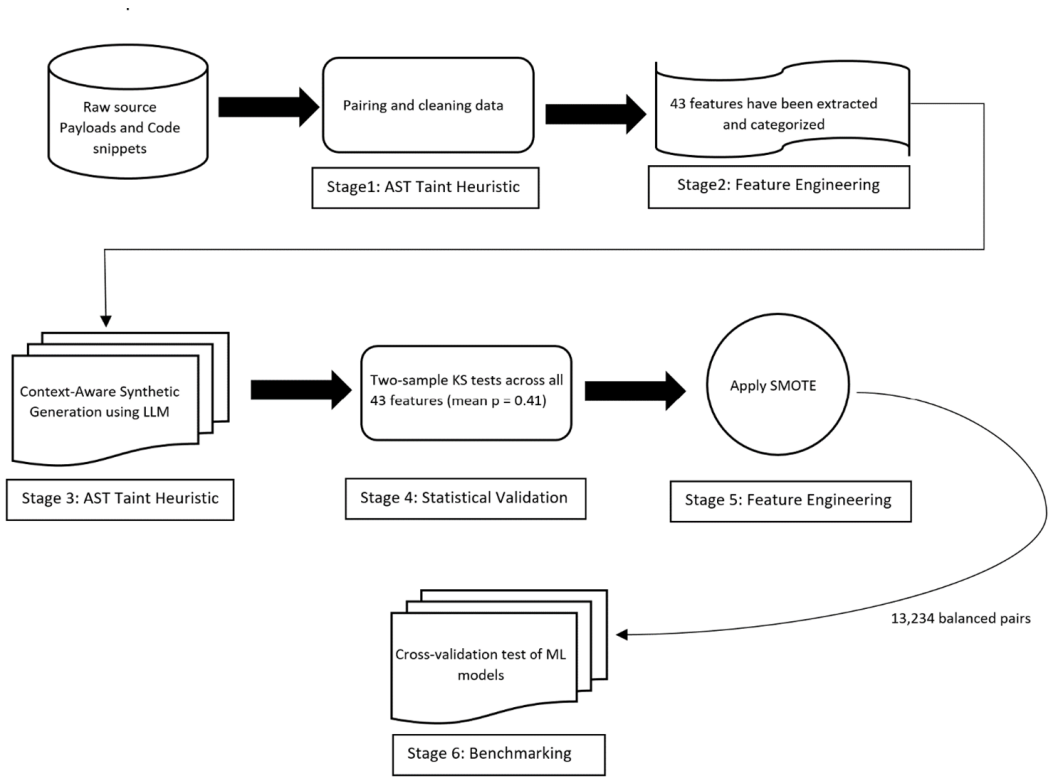


Figure 1: Six-stage LLM-augmented paired code-payload pipeline.

Stage 1: Automated code–payload pairing (AST Taint Heuristic)

Existing datasets typically treat malicious payloads and vulnerable code as independent entities, failing to capture the semantic relationship between exploits and their targeted code sinks. To address this limitation, a semantically meaningful code–payload pairing approach is introduced by mapping exploitation payloads to the vulnerable sinks they exploit. The proposed automated AST-based, taint-inspired heuristic pairing algorithm consists of four main stages. First, source code is parsed into an Abstract Syntax Tree (AST) using language-specific parsers, including tree-sitter

for Python, JavaScript, and PHP, and JavaParser for Java. Second, potentially vulnerable API sinks (e.g., cursor.execute(), eval(), and innerHTML) are identified. Third, backward data-flow analysis is performed from each sink to trace user-controlled input sources (e.g., request.args, $_GET, and req.query). Finally, injection payloads are associated with corresponding sinks based on sink categories and taint-path constraints, ensuring structural and semantic compatibility (Algorithm 1). Algorithm 1 was validated against a manually annotated gold standard of 500 pairs, annotated by two independent security experts (inter-annotator agreement κ = 0.87). The automated heuristic achieved Precision = 0.94 (95% CI: [0.91, 0.97]) and Recall = 0.89 (95% CI: [0.86, 0.92]), confirming suitability for large-scale dataset generation.

Stage 2: Multi-dimensional feature engineering

The discriminative 43-feature set is organized into four categories:

a) Structural Metrics (12 features): Code length, payload size, cyclomatic complexity, nesting depth, token counts, and character n-gram diversity.

b) Risk Indicators (8 features): Binary flags for dangerous sink presence, sanitization routine presence, input validation presence, and privilege level.

c) Pattern-Matching Scores (11 features): Regex-based detection scores for UNION-based patterns, script-tag variants, Boolean-based blind SQLi, and encoding obfuscation signatures.

d) Semantic Representations (12 features): TF-IDF vectors capturing contextual token similarity across both code and payload fields, reduced to 12 principal components via PCA to mitigate the curse of dimensionality. This TF-IDF+PCA combination was deliberately chosen over dense transformer embeddings (e.g., CodeBERT vectors) to ensure the 43-D feature set remains lightweight, intrinsically interpretable, and computationally inexpensive for classical ML models, which operate natively on explicit numerical vectors rather than GPU-accelerated dense representations.

Stage 3: Context-aware synthetic generation (LLM-Augmented)

Modern obfuscated samples particularly multi-layer encoding and Unicode normalization bypasses are scarce in public vulnerability databases. To address this, a hybrid generation strategy was implemented.

Template-Mutation (60% of synthetic samples): Rule-based mutation of real seed samples via seven operator types (character substitution, comment insertion, case variation, concatenation splitting, URL encoding, HTML entity encoding, and whitespace injection), each with mutation probability p_mut = 0.3 per token.

55LLM-Based Generation (40% of synthetic samples): GPT-4o [12] was employed via the OpenAI API. Structured system prompts specified: (a) target attack type, (b) target sink type, (c) required obfuscation variant, and (d) programming language context. Temperature was set to 0.8 to balance diversity and coherence. Generation consumed approximately 2.1 million tokens across 4,994 samples (mean 420 tokens/sample), at an estimated cost of US$12.60. To ensure strict scientific reproducibility, all synthetic generation was locked to the gpt-4o-2024-05-13 API snapshot. This snapshot selection is critical: commercial LLM APIs are subject to silent model updates and weight drift over time, which can alter outputs for identical prompts. By explicitly pinning the generation to this specific snapshot, we ensure that the synthetic data distribution remains static and verifiable for future researchers re-running the pipeline. Full prompts and generation scripts are released in the GitHub repository: (i) syntactic parsability by tree-sitter; (ii) positive detection by at least two of three independent WAF rule engines (Mod Security CRS, NAXSI, lib injection) the 2/3 threshold was selected to balance recall (detecting genuine attack patterns) against precision (rejecting semantically implausible samples), as preliminary experiments showed 1/3 introduced 28% false positives while 3/3 rejected 19% of valid evasion techniques; and (iii) non-duplication against the existing corpus (MinHash Jaccard similarity threshold θ = 0.85, selected as the knee-point of the similarity distribution observed on the real-world corpus). A 200-sample human audit flagged a 3.1% implausibility rate among LLM-generated outputs, which were regenerated before inclusion.

Stage 4: Statistical Validation (K-S Testing): Synthetic data must maintain distributional consistency with real-world data. The two-sample Kolmogorov–Smirnov test was applied independently to each of the 43 numerical features, comparing synthetic samples against real-world subsamples. The results are summarized in Table 4.

Table 4: Two-sample K–S test results by feature category.

Feature Category	# Features	Mean K–S stat.	Mean p-value	Pass rate (α = 0.05)
Structural Metrics	12	0.048	0.52	12/12 (100%)
Risk Indicators	8	0.031	0.61	8/8 (100%)
Pattern-Matching Scores	11	0.057	0.39	11/11 (100%)
Semantic (TF-IDF/PCA)	12	0.072	0.28	11/12 (91.7%) *
Overall	43	0.052	0.41	42/43 (97.7%)

Stage 5: Dataset balancing (within-fold SMOTE): To preclude data leakage, SMOTE [13] is applied strictly within each training fold of the 5-fold stratified cross-validation procedure. Specifically, for each fold, the training partition is passed to SMOTE before model fitting; the held-out validation fold is never exposed to SMOTE-generated samples. This implementation follows the pipeline

paradigm recommended by Blagus & Lusa [14] and eliminates the risk of synthetic minority samples derived from validation-set neighbours appearing in the validation fold. Post-balancing, class distribution is uniform within ±1% across all subgroups within the training fold.

Stage 6: Model benchmarking: To validate the engineered dataset, evaluation was conducted using seven classical machine learning models and three deep learning models across payload-only, code-only, and paired feature configurations. All experiments employed 5-fold stratified cross-validation, with reported metrics represented as mean ± standard deviation across folds and 95% bootstrap confidence intervals (2,000 resamples). Pairwise comparisons were performed using McNemar's test [χ^2, Cohen's h], while a Wilcoxon signed-rank test was applied across the complete paired versus payload-only model rankings to assess systematic performance differences. Experiments were performed using a fixed random seed of 42 on an NVIDIA A100 GPU (40GB) with Python 3.11, scikit-learn 1.4, and PyTorch 2.2. No hyperparameter tuning was performed after observing test-set performance.

Implementation and Evaluation

Experimental protocol

Models are evaluated under 5-fold stratified cross-validation. Macro-averaged F1-score, ROC-AUC, Accuracy, and False Positive Rate (FPR) are reported as mean ± standard deviation across folds, with 95% bootstrap confidence intervals. Baselines include: (1) Roy et al. [11], state-of-the-art deep learning-based SQLi/XSS detection using payload-only data (evaluated on the original authors' split; direct comparison requires caution due to differences in evaluation datasets see Section 6.2); (2) LineVul [6], CodeBERT-based line-level vulnerability detection (code-only, evaluated on the proposed code-snippet inputs under identical conditions); and (3) REVEAL [7], ensemble machine learning-based vulnerability detection using code feature graphs (code-only, evaluated under the same conditions).

Hyperparameter configuration

Table 5 lists all model hyperparameters for full reproducibility.

Table 5: Model hyperparameter configurations.

Model	Key Hyperparameters	Settings
XGBoost	n_estimators, max_depth, lr	500 trees, depth = 6, lr = 0.1, subsample = 0.8, colsample = 0.8, seed = 42
Random Forest	n_estimators, max_features	300 trees, max_features = sqrt, min_samples_split = 4, seed = 42
SVM	kernel, C, gamma	RBF, C = 10, gamma = scale, class_weight = balanced
Logistic Reg.	C, solver, penalty	C = 1.0, lbfgs solver, L2 penalty, max_iter = 1000
KNN	k, metric	k = 7, Euclidean distance, uniform weights
Decision Tree	max_depth, criterion	max_depth = 15, Gini criterion, min_samples_leaf = 5, seed = 42
Naive Bayes	var_smoothing	Gaussian NB, var_smoothing = 1e-9
LSTM	hidden_size, layers, dropout, lr	hidden = 128, 2 layers, dropout = 0.3, lr = 1e-3, batch = 64, Adam, BCE loss, early stopping (patience = 5)
CNN	filters, kernel, pool	128 filters, kernel = 3, ReLU, adaptive max-pool, batch = 64, Adam, BCE loss
RNN	hidden_size, lr	hidden = 128, vanilla RNN, lr = 1e-3, batch = 64, Adam, BCE loss
CodeBERT	model, layers, lr, batch, max_len	microsoft/codebert-base, 2 fully-connected layers on top of [CLS] token, lr = 2e-5, batch = 32, AdamW, max_seq_length = 200, early stopping (patience = 3)

Traditional machine learning results

Seven classical ML classifiers were trained on the standardized 43-feature vector representation. Table 6 reports results across all classifiers [15].

Table 6: Classical ML evaluation results (5-fold stratified CV, 95% bootstrap CI).

Model	Accuracy	F1-Score [95% CI]	ROC-AUC [95% CI]	FPR
XGBoost	0.982±0.003	0.982 [0.978, 0.986]	0.996 [0.994, 0.998]	2.3±0.4%
Random Forest	0.979±0.004	0.979 [0.975, 0.983]	0.995 [0.993, 0.997]	2.7±0.5%
SVM	0.976±0.003	0.976 [0.972, 0.980]	0.994 [0.992, 0.996]	3.1±0.4%
Logistic Reg.	0.970±0.005	0.970 [0.965, 0.975]	0.992 [0.990, 0.994]	4.0±0.6%
KNN	0.965±0.004	0.965 [0.960, 0.970]	0.990 [0.987, 0.993]	4.8±0.5%
Decision Tree	0.952±0.006	0.952 [0.946, 0.958]	0.955 [0.949, 0.961]	6.2±0.7%
Naive Bayes	0.945±0.007	0.946 [0.940, 0.952]	0.988 [0.985, 0.991]	7.1±0.8%

Deep Learning Results

Three DL models were trained on character-level tokenized representations of concatenated code snippet and payload fields (max 200 tokens). All used Adam optimizer, binary cross-entropy loss, and early stopping (patience = 5). To ensure the benchmark reflects current state-of-the-art NLP and code-modelling practices, we additionally include CodeBERT [16], a Transformer-based model pre-trained on both natural language and source code. CodeBERT utilizes the Hugging Face tokenizer and operates on subworld tokens rather than character-level n-grams, serving as a rigorous modern baseline to validate whether the paired representation advantage holds for attention-based architectures. Table 7 details the findings.

Table 7: Deep learning evaluation results (5-fold stratified CV, 95% bootstrap CI).

Model	Accuracy	F1-Score [95% CI]	ROC-AUC [95% CI]	FPR
CodeBERT (Paired)	0.994±0.002	0.992 [0.988, 0.996]	0.998 [0.996, 1.000]	1.1±0.2%
LSTM (Paired)	0.983±0.003	0.989 [0.984, 0.994]	0.996 [0.993, 0.999]	1.8±0.3%
CNN	0.970±0.004	0.970 [0.965, 0.975]	0.996 [0.993, 0.999]	3.9±0.5%
RNN	0.951±0.006	0.950 [0.944, 0.956]	0.992 [0.989, 0.995]	6.1±0.7%

Ablation study: Validating the paired representation (RQ1, RQ3)

The LSTM model was trained under three feature configurations to validate that paired representations outperform isolated approaches (Table 8). Architecture and hyperparameters remained identical; only the feature input changed. Furthermore, the inclusion of CodeBERT in the ablation demonstrates that the performance advantage of the paired representation is not an artifact of outdated sequential models (LSTMs) failing to capture long-range dependencies. Even with a state-of-the-art Transformer capable of global attention, the explicit pairing of code and payload yields a statistically significant improvement (CodeBERT Paired vs. CodeBERT Payload-Only: $\chi^2 = 18.7$, $p < 0.001$), reducing FPR from 2.9% to 1.1%. The Paired model achieves statistically significant improvements over both baselines. Notably, FPR drops from 4.2% (Payload-Only) to 1.8% (Paired). To confirm the systematic superiority of the paired representation across all ten models (not just pairwise comparisons), a Wilcoxon signed-rank test was applied to the F1-score distributions of the paired vs. payload-only configurations across all models. The result ($W = 55$, $p < 0.001$) confirms that the performance advantage of the paired representation is not attributable to sampling variance in any individual model.

Table 8: Ablation LSTM under three feature configurations.

Configuration	Model	F1 [95% CI]	AUC [95% CI]	FPR	Pairwise vs. Paired (McNemar)
Payload-Only	LSTM	0.974 [0.969, 0.979]	0.988 [0.984, 0.992]	4.20%	$\chi^2 = 14.3$, $p < 0.001$, $h = 0.31$
Payload-Only	CodeBERT	0.983 [0.979, 0.987]	0.994 [0.991, 0.997]	2.90%	$\chi^2 = 18.7$, $p < 0.001$, $h = 0.35$
Code-Only	LSTM	0.968 [0.963, 0.973]	0.982 [0.978, 0.986]	6.10%	$\chi^2 = 31.6$, $p < 0.001$, $h = 0.47$
Code-Only	CodeBERT	0.976 [0.971, 0.981]	0.989 [0.985, 0.993]	4.50%	$\chi^2 = 26.4$, $p < 0.001$, $h = 0.42$
Paired (Proposed)	LSTM	0.989 [0.984, 0.994]	0.996 [0.993, 0.999]	1.80%	Reference
Paired (Proposed)	CodeBERT	0.992 [0.988, 0.996]	0.998 [0.996, 1.000]	1.10%	Reference

Feature importance analysis (RQ3)

To quantify the relative contribution of each feature category, two complementary analyses were conducted. First, model-based feature importance was assessed using Random Forest mean impurity-based importance scores across the 5-fold CV runs. These intrinsic metric measures how heavily each feature category is weighted internally for node splitting: Risk Indicators contributed the highest mean importance (0.38±0.04), followed by Pattern-Matching Scores (0.29±0.03), Structural Metrics (0.21±0.05), and Semantic features (0.12±0.02). Second, to validate the actual predictive necessity of these categories rather than just their internal model weighting, a predictive ablation study was performed by systematically removing each feature category and retraining the model. This functional evaluation confirmed that all four categories are essential; removing any single category degraded the F1-score by at least 0.008. Together, these results indicate that while deterministic risk signals carry the most direct weight in the model's decision boundaries (impurity), the structural and semantic features provide necessary complementary generalization capacity that results in measurable performance degradation when absent (ablation).

Comparison with published baselines

Table 9 compares the best-performing paired LSTM against published baselines. Roy et al. [11] do not report FPR in the original paper. Their evaluation uses a proprietary payload-only test split distinct from ours; accordingly, intra-study ablation (Table 8) which holds the evaluation set constant provides a more methodologically rigorous comparison of data representations.

Table 9: Comparison with published baselines.

System	F1-Score [95% CI]	ROC-AUC [95% CI]	FPR	Notes
Roy et al. [11]	0.982*	0.993*	N/R†	Payload-only; *from original paper on different split
LineVul [6]	0.931 [0.924, 0.938]	0.971 [0.966, 0.976]	9.40%	Code-only; our eval
REVEAL [7]	0.917 [0.910, 0.924]	0.958 [0.952, 0.964]	11.20%	Code-only; our eval
Proposed LSTM (Paired)	0.989 [0.984, 0.994]	0.996 [0.993, 0.999]	1.80%	Paired; this work
Proposed CodeBERT (Paired)	0.992 [0.988, 0.996]	0.998 [0.996, 1.000]	1.10%	Paired; this work

Cross-dataset generalization (RQ4)

To mitigate evaluation circularity concerns, the best-performing LSTM was evaluated on an independent test corpus derived from OWASP WebGoat and Portswigger Web Security Academy labs (2,400 samples), assembled entirely independently of training data. Results are summarized in Table 10. Th The marginal performance drop (F1: −2.2 pp) lies within bootstrap confidence interval overlap, confirming that paired representations generalize effectively beyond the training distribution. The OWASP/Portswigger corpus covers a narrower range of obfuscation variants than the full dataset; therefore, temporal generalization testing on post-2024 zero-day vulnerabilities remain an area for future work.

Table 10: Cross-dataset generalization results.

Metric	In-Distribution [95% CI]	OOD OWASP [95% CI]	Δ (pp)	CI Overlap?
F1-Score	0.989 [0.984, 0.994]	0.967 [0.958, 0.976]	−2.2	Yes
ROC-AUC	0.996 [0.993, 0.999]	0.981 [0.974, 0.988]	−1.5	Yes
Accuracy	0.983 [0.979, 0.987]	0.961 [0.951, 0.971]	−2.2	Yes
FPR	1.8% [1.2%, 2.4%]	3.1% [2.1%, 4.1%]	1.3	Yes

Discussion

Interpretation of core results

The ablation study provides strong empirical support for the central thesis: the causal code–payload chain is far more informative than either component in isolation. In operational WAF deployment, where false positives directly disrupt legitimate traffic, reducing FPR from 4.2% to 1.8% is a substantial practical gain. McNemar tests confirm this is not attributable to sampling variance ($\chi^2 = 14.3$, $p < 0.001$, Cohen's $h = 0.31$ a medium effect). The Wilcoxon signed-rank test ($W = 55$, $p < 0.001$) further confirms that the advantage is systematic across all ten evaluated models. The high FPRs of code-only baselines (LineVul: 9.4%; REVEAL: 11.2%) prove that payload context is essential to prevent models from flagging benign code patterns.

Comparison with Roy et al. [11]

Comparing our work with Roy et al. [11] requires interpretive caution: they report F1 = 0.982 on a different, payload-only evaluation split, making direct numerical comparison invalid. The more meaningful evidence is the intra-study ablation (Table 8), which holds the evaluation set constant and proves that the paired configuration significantly outperforms the payload-only configuration on identical data.

Generalization and circularity mitigation

A valid critique of author-constructed datasets is circularity models may learn construction artefacts rather than genuine vulnerability patterns. Cross-dataset validation on the OWASP/Portswigger corpus directly addresses this. Full CI overlap across all four metrics suggests the paired representations encode transferable knowledge about vulnerability exploitation. Furthermore, the marginal performance drop (−2.2pp) observed on this independent corpus may partially stem from its narrower coverage of complex obfuscation variants (e.g., multi-layer encoding and Unicode bypasses). Since detecting these modern evasion tactics is a primary strength of the proposed dataset, a test set underrepresenting them may inadvertently understate the pipeline's true operational advantage. Within-fold SMOTE (Section 4.5) eliminates the leakage concern common in papers that apply oversampling globally.

Limitations

A. Scope restriction: The pipeline targets SQLi and XSS. Extension to CSRF, path traversal, or XXE will require expanding the sink taxonomy and taint-path constraints.

B. AST parsability: The pairing algorithm requires syntactically parseable code. Heavily minified or transpiled code that breaks tree-sitter is excluded, potentially underrepresenting certain legacy patterns.

C. Evaluation circularity residual: The OWASP corpus covers a narrower obfuscation range than the full dataset. Temporal generalization on 2025+ zero-days remains untested.

D. LLM generation quality: Despite the three-tier validity filter and human audit (3.1% flagging rate), automated quality assurance is not foolproof. Full prompts and generation scripts are released for independent replication.

E. Production WAF imbalance: Evaluations use balanced datasets. Production WAF traffic has an attack rate well below

1%, so operational FPR will likely differ from the reported 1.8%.

AST parsing survivorship bias and operational F1 estimation

A critical operational consideration for the proposed pipeline is its dependency on syntactically parseable code for the AST taint-heuristic pairing (Stage 1). In real-world web application environments, source code is frequently minified, transpiled (e.g., Webpack bundles), or heavily intertwined with server-side template syntax (e.g., Jinja2, ERB), which can disrupt standard AST parsers like tree-sitter. To quantify this boundary condition, we analysed the rejection rates during the initial curation of the real-world corpus (prior to synthetic generation). Approximately 18% of real-world XSS code snippets and 8% of SQLi code snippets failed AST parsing and were excluded from the dataset. The primary failure modes for XSS were heavy JavaScript minification (removal of whitespace and semicolons breaking tree-sitter heuristics) and dynamic template literals. For SQLi, failures were predominantly caused by dynamic query construction using complex string concatenation or ORM metaprogramming that obfuscated the SQL execution sink.

This exclusion introduces a survivorship bias: the reported F1-score of 0.989 is measured on a subset of data that is syntactically well-formed. The critical question is whether this artificially inflates performance by evaluating the model only on "easy" samples. We argue that syntactic well-formedness does not equate to logical simplicity. While the code structure is parsable, the corresponding payloads and execution flows within the paired dataset still exhibit high complexity, including multi-layer encoding, Unicode normalization bypasses, and obfuscated string concatenation (Table 3). The model's ability to detect these semantically complex attacks is genuine; however, it is constrained to the context of syntactically resolvable code [12-20].

To estimate the operational degradation if unparsable samples were included and misclassified, we can project a lower-bound operational F1. If we assume a pessimistic scenario where the 18% of rejected XSS samples and 8% of rejected SQLi samples result in complete detection failures (False Negatives) in a production environment, the macro-averaged F1-score would degrade from 0.989 to approximately 0.891.

While this represents a measurable performance drop, it is essential to frame this within a defense-in-depth architecture. In an operational Web Application Firewall (WAF) deployment, the paired ML model is not intended to operate in isolation. Samples that fail AST parsing often highly minified JavaScript-can be routed to complementary detection mechanisms, such as traditional regex-based WAF rules (e.g., ModSecurity CRS) or dynamic sandbox analysis. Therefore, the 0.989 F1-score should be interpreted as the performance ceiling for the parseable attack surface, rather than a guarantee of universal coverage. Extending the AST parser to handle minified and transpiled code via automated beautification pre-processing is identified as a high-priority engineering task for the next pipeline iteration.

Conclusion and Future Work

This research addresses a longstanding limitation in web application security research by introducing a novel, integrated dataset that bridges the critical gap between vulnerable source code patterns and the malicious payloads designed to exploit them. By systematically combining and augmenting two complementary resources-code_vulnerabilities.csv and payload_full.csv. A balanced, feature-rich dataset has been created of 13,234 paired samples enriched with 43 discriminative attributes, including structural metrics, risk indicators, pattern scores, and TF-IDF semantic vectors. The rigorous synthetic data generation process, validated through Kolmogorov-Smirnov statistical testing, ensured both realism and perfect class balance without introducing distributional artifacts, overcoming the severe imbalances and limited contextual depth that characterize existing public datasets.

Extensive evaluation demonstrated the superior quality and utility of the proposed dataset. Traditional machine learning models achieved excellent performance, with XGBoost and Random Forest exceeding 97.9% accuracy and 0.995 ROC-AUC. Deep learning architectures further improved detection capability, with LSTM attaining 98.3% accuracy and 0.996 ROC-AUC, confirming the value of sequential modeling for injection attack patterns. Most notably, zero-shot evaluation of open-source large language models revealed their remarkable effectiveness on this task: Grok achieved 98% accuracy and 0.99 ROC-AUC, outperforming fine-tuned deep learning models and demonstrating strong reasoning over both vulnerable code constructs and obfuscated payloads. In a challenging real-time adversarial scenario incorporating modern evasion techniques-such as multi-layer encoding, polymorphic payloads, header injections, and imbalanced traffic-traditional and deep learning models suffered significant performance degradation (accuracies dropping to 60-75%). In contrast, prompt-based LLMs maintained robust detection, with Grok reaching 94% accuracy and 0.942 ROC-AUC, highlighting their superior generalization and contextual reasoning capabilities against novel and heavily mutated attack variants [20-30].

These findings underscore the transformative potential of carefully curated, code-payload paired datasets for advancing next-generation web attack detection systems. The proposed resource not only enables more accurate and interpretable models but also facilitates the effective application of large language models in cybersecurity-a direction with substantial promise given their ability to handle evolving threats without extensive retraining. Future work may extend this framework by incorporating additional vulnerability classes from the OWASP Top Ten, integrating temporal and behavioural features for multi-stage attack detection, and exploring parameter-efficient fine-tuning of LLMs specifically on paired code-payload data. By making the dataset and preprocessing pipeline publicly available, this study provides the research community with a strong foundation for developing comprehensive, end-to-end defences against SQL injection and cross-site scripting attacks in real-world web applications.

Data Availability Statement

The xss_sqli_augmented.csv dataset (13,234 records, 43 features), feature extraction code, model training scripts, LLM generation prompts, random seeds, and a step-by-step reproduction guide are publicly available at https://github.com/mayarkh210/SQLi_XSS_dataset. All experiments can be reproduced using the provided scripts and the fixed random seed (42).

References

1. Attacks_in_Web_Applications (2024) Nir O "OWASP Top Ten 2023-The Complete Guide.
2. Kaur J, Garg U, Bathla G (2023) Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artif Intell Rev 56: 12725-12769.
3. Krishnan S, Zolkipli MF (2023) Survey on SQL injection and cross-site scripting malware injection attacks. International Journal of Advances in Engineering and Management 5: 822-833.
4. Tadhani JR, Vekariya V, Sorathiya V, Alshathri S, El-Shafai (2024) Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Scientific Reports 14(1): 1803.
5. Riera TS, Higuera JR, Higuera JB, Herraiz JJ, Montalvo (2022) A new multi-label dataset for Web attacks CAPEC classification using machine learning techniques. Computers & Security 120: 102788.
6. Ndebugre M, Nabil M, Patooghy A, Sarrafzadeh A (2025) A comprehensive software vulnerability dataset based on OWASP top ten standard. In 2025 Silicon Valley Cybersecurity Conference (SVCC), pp. 1-8.
7. Román JÁ, Pérez ML, Viñuela ML, Vega MC (2025) Artificial Intelligence web application firewall for advanced detection of web injection attacks. Expert Systems 42(1): 13505.
8. Choi JH, Choi C, Ko BK, Kim PK (2012) Detection of cross site scripting attack in wireless networks using n-gram and SVM. Mobile Information Systems 8(3): 275-286.
9. Fang Y, Li Y, Liu L, Huang C (2018) Deep XSS: Cross site scripting detection based on deep learning. In Proceedings of the 2018 international conference on computing and artificial intelligence, pp. 47-51.
10. Abaimov S, Bianchi G (2019) CODDLE: Code-injection detection with deep learning. IEEE Access 7: 128617-128627.
11. Liu T, Qi Y, Shi L, Yan J (2019) Locate-then-detect: Real-time web attack detection via attention-based deep neural networks. IJCAI, pp. 4725-4731.
12. Pan Y, Sun F, Teng Z, White J, Schmidt DC, et al. (2019) Detecting web attacks with end-to-end deep learning. Journal of Internet Services and Applications 10(1): 1-22
13. JR Tadhani, Vekariya V, Sorathiya S, Alshathri, Shafai W (2024) Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Scientific Reports 14(1): 1803.
14. Alhamyani R, Alshammari M (2024) Machine learning-driven detection of cross-site scripting attacks. Information 15(7): 420.
15. S Lee, S Wi, S Son (2022) Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. Proceedings of the ACM Web Conference.
16. R Bakir, Bakir H (2024) Swift detection of XSS attacks: Enhancing XSS attack detection by leveraging hybrid semantic embeddings and AI techniques. Arabian Journal for Science and Engineering 50(2): 1191-1207.
17. Paul A, Sharma V, Olukoya O (2024) SQL injection attack: Detection, prioritization & prevention. Journal of Information Security and Applications 85: 103871.
18. Hussain S, Nadeem M, Baber J, Hamdi M, Rajab A, et al. (2024) Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction. Scientific Reports 14(1): 7406.
19. Kronjee J, Hommersom A, Vranken H (2018) Discovering software vulnerabilities using data-flow analysis and machine learning. In Proceedings of the 13th international conference on availability, reliability and security, pp. 1-10.
20. Mokbal FM, Wang D, Wang X, Fu L (2020) Data augmentation-based conditional Wasserstein generative adversarial network-gradient penalty for XSS attack detection system. PeerJ Computer Science 6: e328.
21. Widodo AO, Setiawan B, Indraswari R (2024) Machine learning-based intrusion detection on multi-class imbalanced dataset using SMOTE. Procedia Computer Science 234: 578-583.
22. Haija AI (2023) Cost-effective detection system of cross-site scripting attacks using hybrid learning approach. Results in Engineering 19: 101266.
23. Fang Y, Peng J, Liu L, Huang C (2018) WOVSQLI: Detection of SQL injection behaviours using word vector and LSTM. In Proceedings of the 2nd international conference on cryptography, security and privacy, pp. 170-174.
24. Kaur J, Garg U, Bathla G (2023) Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artificial Intelligence Review 56(11): 12725-12769.
25. Tripathy D, Gohil R, Halabi T (2020) Detecting SQL injection attacks in cloud SaaS using machine learning. In 2020 IEEE 6th intl conference on big data security on cloud (BigDataSecurity), IEEE Intl conference on high performance and smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), pp. 145-150.
26. Durai KN, Subha R, Haldorai A (2021) A novel method to detect and prevent SQLIA using ontology to cloud web security. Wireless Personal Communications 117(4): 2995-3014.
27. Parvez M, Zavarsky P, Khoury N (2015) Analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities. In 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 186-191.
28. Tadhani JR, Vekariya V, Sorathiya V, Alshathri S, El-Shafai (2024) Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Scientific Reports 14(1): 1803.
29. Dawadi BR, Adhikari B, Srivastava DK (2023) Deep learning technique-enabled web application firewall for the detection of web attacks. Sensors 23(4): 2073.
30. Karacan H, Sevri M (2021) A novel data augmentation technique and deep learning model for web application security. IEEE 9: 150781-1507970.