Journal of Intelligent Learning Systems and Applications, 2026, 18(3), 181-197
https://www.scirp.org/journal/jilsa
ISSN Online: 2150-8410
ISSN Print: 2150-8402

Dual-Stream Detection of HTTP Injection Attacks: A Hybrid Architecture Combining ModernBERT and Character-Level CNNs

Emil Wangilisasi, Judith Leo, Anael Sam

School of Computational and Communication Science and Engineering, Nelson Mandela African Institution of Science and Technology (NM-AIST), Arusha, Tanzania
Email: emily.wangilisasi@nm-aist.ac.tz, judith.leo@nm-aist.ac.tz, anael.sam@nm-aist.ac.tz

How to cite this paper: Wangilisasi, E., Leo, J. and Sam, A. (2026) Dual-Stream Detection of HTTP Injection Attacks: A Hybrid Architecture Combining ModernBERT and Character-Level CNNs. *Journal of Intelligent Learning Systems and Applications*, **18**, 181-197.
https://doi.org/10.4236/jilsa.2026.183012

Received: May 16, 2026
Accepted: June 26, 2026
Published: June 29, 2026

Abstract

Web applications remain critically vulnerable to injection attacks, including SQL Injection (SQLi), OS Command Injection, and Cross-Site Scripting (XSS) among others, which exploit the semantic gap between user-supplied input and executable code. Traditional Web Application Firewalls (WAFs) rely on signature-based pattern matching, rendering them susceptible to evasion through payload obfuscation. While Transformer-based language models excel at capturing long-range contextual dependencies and have demonstrated promise in detecting malicious intent, they alone may not fully exploit the fine-grained character-level patterns that distinguish obfuscated attack payloads. This paper proposes a Dual-Stream Hybrid Architecture that combines the contextual reasoning capabilities of Modern Bidirectional Encoder Representations from Transformers (ModernBERT), a state-of-the-art encoder supporting 8192 token sequences, with a Character-Level 1D Convolutional Neural Network (CNN) optimized for morphological pattern recognition. The semantic stream captures high-level payload intent, while the syntactic stream detects low-level structural signatures of obfuscation techniques (e.g., URL encoding, hexadecimal evasion, comment injection). Experiments on a composite dataset of ~93,500 HTTP requests, aggregated from public payload repositories, synthetic attack campaigns generated with SQLMap and XSStrike on vulnerable web applications, live honeypot traffic captured via T-Pot (SNARE/Tanner), and benign browsing sessions simulated with Playwright, demonstrate that our hybrid approach achieves 98.7% accuracy and 98.3% F1-score, outperforming standalone ModernBERT and CNN baselines by 2.4% and 8.6% respectively.

Keywords

Web Application Security, HTTP Injection, Deep Learning, ModernBERT,

Convolutional Neural Networks

1. Introduction

The ubiquity of web applications in modern digital infrastructure has dramatically expanded the attack surface for application-layer vulnerabilities. The Open Web Application Security Project (OWASP) Top 10 [1] lists Injection flaws (A05:2025) among the most critical security risks. Attackers exploit these vulnerabilities through techniques such as SQL Injection (SQLi), manipulating database queries to exfiltrate sensitive data or bypass authentication, and Cross-Site Scripting (XSS), injecting malicious scripts that execute in victim browsers to steal session tokens or perform unauthorized actions.

The detection of injection attacks presents a fundamental tension between generalization and precision. Current detection paradigms fall into two principal categories:

Signature-Based/Rule-Based Systems (WAFs): Commercial and open-source WAFs such as ModSecurity employ Regular Expression (RegEx) pattern matching against known attack signatures. While computationally efficient with O(n) complexity, these systems exhibit inherent brittleness; a single novel obfuscation technique, such as substituting whitespace with inline SQL comments (/**/) or employing double URL encoding (%2527 to 27% to '), can bypass detection entirely. Empirical studies report bypass rates exceeding 70% against production WAFs using automated evasion tools [2].

Machine Learning Approaches: Recent advances have applied methods such as Long Short-Term Memory networks (LSTMs), Recurrent Neural Networks (RNNs), and the Transformer based models to do payload classification [3] [4]. However, standard Transformer models operate at the sub-word level such as WordPiece [5] and Byte-Pair Encoding [6], which, while effective for capturing semantic context, may not fully leverage the fine-grained character-level patterns that distinguish obfuscated payloads, such as encoding sequences, special character distributions, and structural irregularities.

To mitigate these issues, a Hybrid Deep Learning Architecture is introduced that processes HTTP payloads through two complementary pathways operating in parallel. Our key contributions are:

- **Dual-Stream Fusion:** We propose a hybrid architecture combining ModernBERT [7] for semantic understanding with a Character-Level CNN for syntactic pattern recognition, enabling robust detection across both natural language manipulation and code-level obfuscation.
- **Obfuscation Resilience:** Through ablation studies, we demonstrate that the character-level branch provides complementary pattern recognition capabilities, maintaining 96.5% accuracy on highly obfuscated payloads where standalone Transformers degrade to 84.2%.

2. Related Work

2.1. Traditional Web Application Firewalls

Signature-based detection remains the predominant approach in commercial WAFs. ModSecurity's Core Rule Set (CRS) employs over 200 curated regular expressions targeting known injection patterns [1]. While achieving high precision on canonical attack vectors, these systems suffer from two critical limitations *i.e.*, zero-day vulnerability, where novel attack patterns require manual rule updates, and evasion susceptibility, where automated tools such as SQLMap [8] and WAF-Ninja [9] systematically generate bypass payloads through encoding permutations, case manipulation, and comment injection.

2.2. Machine Learning for Intrusion Detection

Early ML approaches employed traditional algorithms such as Logistic Regressions, Random Forests, Naive Bayes and Support Vector Machines operating on features that are manually crafted such as n-gram frequencies and character distribution [10] [11]. While effective on clean datasets, these methods require extensive feature engineering and do not generalize well to novel attack variants.

Deep learning approaches have progressively improved detection capabilities. The authors in [12] demonstrated that BERT-LSTM networks achieve 97.3% accuracy on SQLi detection by extracting sentence-level and word features, then generating embedding vectors that identify malicious SQL query patterns. In [13], researchers applied attention mechanisms to identify salient payload regions, improving interpretability. However, these approaches operate at the word or subword level, leaving room for complementary character-level analysis to capture fine-grained morphological patterns associated with obfuscation techniques.

2.3. Transformer Models in Security Applications

BERT (Bidirectional Encoder Representations from Transformers) [14] has been adapted for various security tasks including malware classification, phishing detection, and vulnerability identification [15]. However, standard BERT architectures impose a 512-token context limit, insufficient for analyzing verbose HTTP requests or multi-stage injection payloads that may span thousands of characters.

ModernBERT [7] released in late 2024, addresses these limitations through architectural innovations including Flash Attention [16] for memory-efficient O(N) attention computation, Alternating Attention interleaving local sliding window attention (128 tokens) with global attention layers, and Rotary Positional Embeddings (RoPE) [17] enabling extrapolation to 8192 tokens. These advances make ModernBERT particularly suitable for security applications requiring analysis of complete HTTP requests including headers, cookies, and body content, while maintaining the inference speed necessary for real-time deployment.

2.4. Character-Level Neural Networks

Character-level CNNs do not require tokenization to achieve good performance

on text classification tasks, learning hierarchical representations directly from raw character sequences. Subsequent work demonstrated their effectiveness in detecting the morphological signatures of malicious content, such as anomalous character distributions, encoding patterns, and structural irregularities, that may be invisible to word-level models [18].

2.5. Hybrid and Ensemble Approaches

Multi-modal fusion architectures have shown promise in combining complementary feature representations. Kim *et al.* [19] proposed concatenating word embeddings with character-level CNN features for sentiment analysis, achieving improvements over single-modality baselines. Our work extends this paradigm to the security domain, specifically targeting the unique challenges of injection attack detection.

3. Threat Model and Problem Formulation

3.1. Threat Model

An adversary is considered attempting to exploit web application vulnerabilities through injection attacks embedded in HTTP request components (URL parameters, POST body, headers, cookies). The adversary possesses:

- **Knowledge of target application:** Understanding of backend database systems (MySQL, PostgreSQL, etc.) and client-side scripting contexts.
- **Evasion capabilities:** Access to automated obfuscation tools capable of generating encoding variants, comment injection, case manipulation, and other bypass techniques.
- **Obfuscation tooling:** Access to publicly available evasion tools and encoding techniques to generate obfuscated payload variants prior to deployment.

The defender deploys an inline detection system that must classify each incoming HTTP request as benign or malicious within strict latency constraints (<50 ms) while minimizing both false negatives (missed attacks) and false positives (blocked legitimate traffic).

3.2. Problem Formulation

Given an HTTP payload x represented as a character sequence $x = (c_1, c_2, \cdots, c_n)$ where $c_i \in \Sigma$ (ASCII character set), the detection task is formulated as binary classification as shown in Equation (1):

$$f(x) \to \{0,1\} \tag{1}$$

where f is the learned classifier, 0 denotes benign traffic, and 1 denotes malicious injection attempt. The objective is to learn f that maximizes detection rate (recall) while constraining false positive rate below operational thresholds.

4. Methodology

We propose a Dual-Stream Architecture that processes HTTP payloads through

two independent branches, semantic and syntactic, followed by intermediate feature-level fusion via concatenation before final classification. In this design, the semantic representation produced by ModernBERT and the syntactic representation produced by the character-level CNN are concatenated to form a unified feature vector, which is then passed to a shared classification head. This enables the model to jointly exploit contextual and structural information when distinguishing benign requests from injection attacks. **Model Input:** In the experiments reported in this work, each training sample was represented by a single text field (the payload column after dataset consolidation). For many GET samples we used the query string or URL parameters and for POST/PUT samples we used the body and for repository or tool-generated records, a standalone payload string as provided by the source was used. The same text was fed to both the ModernBERT and character-level CNN branches. HTTP method, headers, cookies, source labels, timestamps, IP addresses, attack-family labels, and ground-truth class labels were not supplied as separate predictive features, they were used only for dataset analysis, stratified splitting, and evaluation. Figure 1 illustrates the complete architecture.

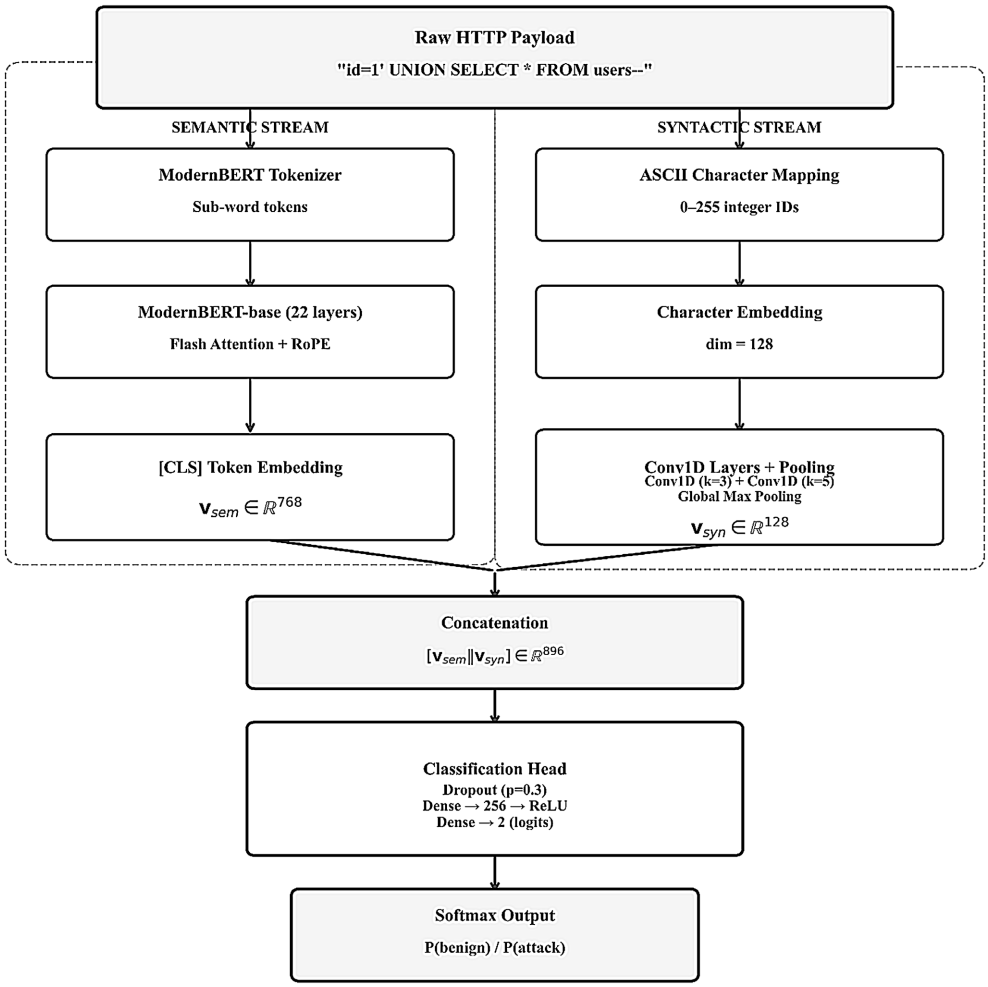

Figure 1. Dual-stream architecture for HTTP injection detection. ModernBERT (left) encodes sub-word semantics while a character Conv1D (right) captures morphological signatures. Concatenated embeddings pass through a classification head for binary prediction.

4.1. Branch A: Semantic Stream (ModernBERT)

The semantic branch leverages ModernBERT-base to capture contextual understanding of payload intent.

Input Processing: The raw payload string is tokenized using the ModernBERT tokenizer with maximum sequence length L_max = 512 tokens. We select this limit because empirical analysis shows that over 99% of HTTP payloads in our dataset fall within 512 tokens, balancing computational efficiency with coverage. The underlying ModernBERT architecture supports up to 8192 tokens, allowing seamless scaling for applications requiring analysis of verbose multi-part requests. Payloads exceeding the configured limit are truncated with a warning logged for analysis.

Model Architecture: ModernBERT-base comprises 22 Transformer layers with hidden dimension $d = 768$, 12 attention heads, and intermediate feed-forward dimension 3072.

Feature Extraction: The final hidden state associated with the [CLS] token is extracted and used as the semantic representation, as expressed in Equation (2).

$$v_{sem} = \text{ModernBERT}(x)_{[\text{CLS}]} \in \mathbb{R}^{768} \tag{2}$$

This vector encodes the aggregate semantic content of the payload, capturing high-level patterns such as "query attempting data retrieval" or "script injection targeting DOM manipulation".

4.2. Branch B: Syntactic Stream (Character-Level CNN)

The syntactic branch employs a 1D Convolutional Neural Network operating directly on character sequences to detect morphological attack signatures.

Input Processing: Each character in the payload is mapped to its ASCII integer value $c_i \in \{0,1,\cdots,255\}$. The input sentences are padded if less or truncated if they exceed the maximum to fixed length $L_{char} = 1024$ characters.

Architecture Details: Table 1 lists the layer configuration of the character-level CNN branch.

Feature Extraction: Global max pooling extracts the most salient activation

Table 1. Character-level CNN architecture.

Layer	Configuration	Output Shape	Purpose
Embedding	256 × 128	(L, 128)	Dense character representations
Conv1D-1	64 filters, k = 3, stride = 1, ReLU	(L, 64)	Trigram detection (%27, OR)
MaxPool1D	pool size = 2	(L/2, 64)	Dimensionality reduction
Conv1D-2	128 filters, k = 5, stride = 1, ReLU	(L/2, 128)	Longer patterns (script, UNION)
MaxPool1D	pool size = 2	(L/4, 128)	Dimensionality reduction
Conv1D-3	128 filters, k = 3, stride = 1, ReLU	(L/4, 128)	Higher-order combinations
GlobalMaxPool	----	(128)	Sequence-invariant features

across the sequence:

$$v_{syn} = \text{GlobalMaxPool}\left(\text{CNN}\left(x\right)\right) \in \mathbb{R}^{128} \tag{3}$$

This representation captures character-level signatures including high density of special characters (%, ', <, >), encoding patterns indicative of obfuscation, and structural anomalies such as nested brackets or comment sequences.

4.3. Feature Fusion and Classification

The semantic and syntactic representations are concatenated to form a unified feature vector:

$$v_{final} = \left[v_{sem} \,\|\, v_{syn}\right] \in \mathbb{R}^{896} \tag{4}$$

This fused representation passes through a classification head:

$$\hat{y} = \text{Softmax}\left(W_2 \cdot \text{ReLU}\left(W_1 \cdot \text{Dropout}\left(v_{final}, p = 0.3\right) + b_1\right) + b_2\right) \tag{5}$$

where $W_1 \in \mathbb{R}^{256\times896}, W_2 \in \mathbb{R}^{2\times256}$ are learned weight matrices.

4.4. Training Objective

Weighted cross-entropy loss is used as the loss function. This is used in consideration of class imbalance and prioritize attack detection:

$$\mathcal{L} = -\sum_i w_{y_i} \cdot y_i \log\left(\hat{y}_i\right) \tag{6}$$

where $w_0 = 1.0$ (benign) and $w_1 = 2.5$ (attack), penalizing false negatives more heavily than false positives given the asymmetric cost of missed attacks in security applications.

4.5. Implementation Details

The two branches are trained end-to-end jointly. To accommodate the different convergence rates of pre-trained (ModernBERT) and randomly-initialized (CNN) components, we employ differential learning rates $\eta_{BERT} = 2\times10^{-5}$ $\eta_{CNN} = 1\times10^{-3}$. To prevent exploding gradients, Gradient clipping with max norm = 1.0 is used during early training.

5. Experimental Setup

5.1. Dataset Construction

To ensure broad coverage of real-world attack patterns and minimize dataset bias, we constructed a composite dataset by aggregating HTTP injection payloads and benign traffic from four complementary sources. Table 2 summarizes the dataset composition.

Source 1: Public Repositories (GitHub): We curated malicious payloads from widely-used open-source collections including payloadbox/sql-injection-payload-list [20], payloadbox/xss-payload-list [21], and swisskyrepo/PayloadsAllThe Things [22]. After deduplication and normalization, this source contributed approxi-

mately 18,000 unique SQLi and XSS payloads spanning classic injection vectors, filter bypass techniques, and polyglot payloads.

Source 2: OWASP Resources: Additional attack samples were drawn from the OWASP Testing Guide payload corpus and the OWASP Web Security Testing Guide's injection test cases [1]. These payloads reflect structured, expert-curated attack taxonomies and contributed approximately 6500 samples covering SQLi, XSS, LDAP Injection, and OS Command Injection vectors.

Source 3: Synthetic Generation on Vulnerable Applications: To capture realistic attack traffic with full HTTP context (headers, cookies, session state), we deployed DVWA (Damn Vulnerable Web Application) [23] and OWASP Juice Shop [24] in isolated Docker environments and executed automated attack campaigns using SQLMap [8] with tamper scripts for case swapping, comment injection, encoding variations, and keyword fragmentation and XSStrike for context-aware XSS payload generation with DOM analysis. All HTTP traffic was routed through Burp Suite [25] configured as an intercepting proxy, which logged complete request/response pairs including headers, cookies, and body content. This pipeline generated approximately 12,000 attack requests with authentic HTTP structure and session context, including obfuscated variants produced by SQLMap's tamper modules (e.g., between, charencode, space2comment, randomcase).

Source 4: Live Honeypot Traffic: To incorporate genuine attacker behavior observed in the wild, we deployed a T-Pot honeypot platform [26] on a public-facing VPS (4 vCPU, 8GB RAM) with its web application honeypot components SNARE [27] acting as the vulnerability-emulating HTTP sensor and Tanner [28] the backend analysis engine that classifies and logs incoming attacks. The honeypot was exposed on standard HTTP/HTTPS ports of 80 and 443 respectively for a period of 90 days, during which it captured approximately 22,000 malicious HTTP requests originating from automated scanners, botnets, and manual exploitation attempts. This source provides ecological validity, reflecting attack distributions and obfuscation strategies encountered in production environments.

Benign Traffic Collection: Legitimate HTTP traffic was collected by simulating realistic user browsing sessions on various web applications including WordPress-based sites, the OWASP Juice Shop and a custom e-commerce application using Playwright browser automation [29]. Scripts emulated diverse user behaviors, including form submissions, search queries, navigation, file uploads, and AJAX interactions, across multiple browser profiles with varying user agents and session patterns. HTTP request logs were captured via Playwright's network interception API, yielding approximately 35,000 benign requests with natural parameter distributions and header structures. All collected requests were normalized to a canonical format preserving the raw payload string alongside metadata (HTTP method, content type, source label). Duplicate payloads were removed. Table 2 and Figure 2 summarize the dataset composition and class distribution.

Labelling Protocol: Samples were labeled according to their collection route and verification context. Payloads from curated public repositories and OWASP

Table 2. Dataset composition by source.

Source	Type	Samples	Collection Method
GitHub Repositories	Malicious	~18,000	Curated payload lists
OWASP Resources	Malicious	~6,500	Expert-curated test cases
Synthetic (DVWA/Juice Shop)	Malicious	~12,000	SQLMap + XSStrike via Burp Suite
T-Pot Honeypot (SNARE/Tanner)	Malicious	~22,000	Live capture over 90 days
Playwright Browsing Simulation	Benign	~35,000	Automated realistic browsing
Total	—	**~93,500**	—

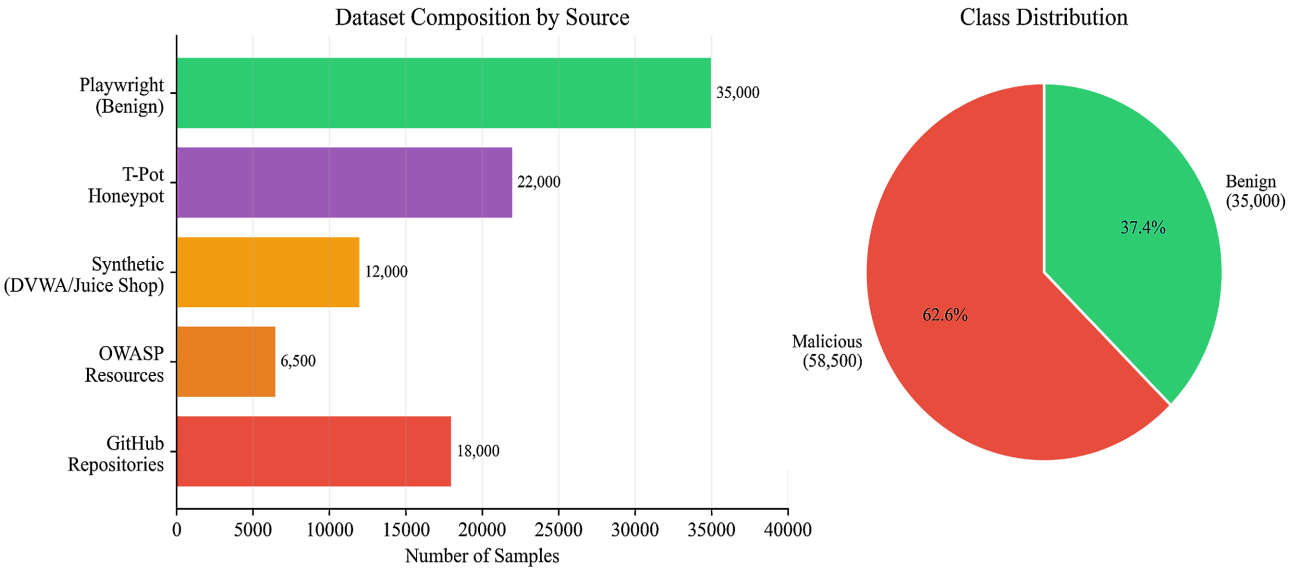

Figure 2. Dataset composition by source and class distribution. The left panel shows the number of samples contributed by each collection source. The right panel illustrates the overall malicious-to-benign class ratio.

resources were labeled malicious according to their documented attack category. Traffic generated by SQLMap and XSStrike against intentionally vulnerable DVWA and OWASP Juice Shop instances was labeled malicious because the tools generated known attack attempts in a controlled environment. The collected Tanner HTTP honeypot logs were processed using a custom Python script, which parsed the raw logs and converted them into a structured event dataset. The script normalized the request fields, removed duplicate records using a deterministic SHA-256 fingerprint generated from selected request fields, and extracted features from the URL path, payload body, headers, user-agent, referer, and request method. Initial labeling was carried out using a rule-based weak-labeling protocol designed for injection and exploit detection, where each event was checked for indicators such as SQL injection patterns, command injection terms, path traversal sequences, file inclusion attempts, XSS-like payloads, shell-related keywords, encoded tra-

versal strings, .env, .git, phpunit, /etc/passwd, and suspicious PHP or WordPress exploit paths. Events matching these indicators were labeled as attack-related, with the specific attack type assigned where a recognizable injection pattern was present, while repeated non-browser-like requests from the same IP within short time windows were labeled as scanner activity, lower-confidence suspicious requests were labeled as probes, browser-like requests with no suspicious indicators were labeled as benign, and unmatched events were marked as unknown. These weak labels were then treated as preliminary labels and transitioned toward ground truth through manual validation, where the request paths, payloads, headers, methods, user-agents, and source behavior were reviewed against predefined annotation criteria. Labels were confirmed when the request evidence clearly matched the assigned class, corrected when a rule-based label was inaccurate, and marked as unknown or ambiguous when the available evidence was insufficient. This process produced a reproducible and more reliable labeled dataset for injection-attack detection, with each final label supported by explicit behavioral or payload-based evidence.

After deduplication, the composite dataset comprised 93,500 samples in total: 35,000 benign requests collected via Playwright browsing simulation and 58,500 malicious requests. Among the malicious samples, the documented attack-family distribution was 30,800 SQL injection cases (from GitHub repositories, OWASP resources, SQLMap campaigns, and T-Pot/Tanner honeypot logs), 18,500 cross-site scripting cases (GitHub, OWASP, XSStrike, and T-Pot/Tanner), 4800 OS command injection cases (OWASP, synthetic generation, and T-Pot/Tanner), 2450 LDAP injection cases (primarily OWASP and curated sources), and 2950 XML/XPath injection cases (OWASP, curated lists, and honeypot traffic). Attack-family labels were used only for dataset characterization and analysis, model training and evaluation used binary benign versus malicious labels.

Deduplication and Leakage Control: Deduplication and partition assignment were performed before model training on the normalized payload/request text used as model input. Exact duplicate strings were removed first. To limit train/test leakage from obfuscated or transformed variants of the same underlying payload, each sample was mapped to a canonical form by iteratively applying safe URL and HTML entity decoding (up to three passes, stopping when the string no longer changed), lowercasing case-insensitive tokens, and normalizing whitespace and common SQL comment artifacts. A SHA-256 hash of the canonical string was computed for each sample and used as a group identifier. All samples sharing the same hash were treated as one group and assigned wholly to a single partition so that encoding, case, or comment variants of the same base payload could not appear in both training and evaluation subsets.

Dataset Split: The dataset was divided into 70% training data while 15% was used for validation during experiments, and 15% for testing subsets using stratified sampling to preserve the class distribution across all splits to ensure proportional representation of attack types and data sources across splits.

5.2. Baseline Models

We compared against a baseline set spanning one rule-based system, one classical ML model, one character-level neural model, one ModernBERT-only semantic model, and the proposed hybrid architecture. These baselines are summarized in Table 3.

Table 3. Baseline models evaluated in this study.

Model	Description
ModSecurity CRS	Rule-based WAF with OWASP Core Rule Set v3.3
Random Forest	Classical ML with TF-IDF features (n-grams 1-3)
Character-CNN	Our CNN branch trained independently
ModernBERT	ModernBERT-base fine-tuned independently
Hybrid (Ours)	Full dual-stream architecture

5.3. Training Configuration

Table 4 lists the software stack, optimization settings, and hardware used for model training and fine-tuning.

Table 4. Training configuration and compute environment.

Parameter	Value
Framework	PyTorch/Hugging Face Transformers
Optimizer	AdamW $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 10^{-8}$
Learning Rate (ModernBERT)	2×10^{-5} with linear warmup (10% steps)
Learning Rate (CNN)	1×10^{-3} with cosine annealing
Weight Decay	0.01
Batch Size	32
Epochs	3 (early stopping on validation F1)
Hardware	NVIDIA GPU A100 (40 GB)

5.4. Evaluation Metrics

Standard classification metrics are reported with particular emphasis on security-relevant measures:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{False Positive Rate}(\text{FPR}) = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

AUC-ROC: The area beneath the ROC curve, used to evaluate the model's ability to distinguish between attack and benign samples across different classification thresholds.

6. Results and Discussion

The main results are reported from a single training run with a fixed random seed of 42. However, in a later section we report results from two additional runs with two other random seeds of 123 and 2024 to observe run-to-run stability.

6.1. Overall Performance Comparison

Table 5 summarizes the performance results of all evaluated models on the combined test set. Figure 3 shows the confusion matrix for the Hybrid model using the held-out test set.

Table 5. Comparison of the performance on the composite multi-source test set.

Model	Accuracy	Precision	Recall	F1-Score	FPR	AUC-ROC
ModSecurity CRS	78.3%	95.2%	62.1%	75.2%	0.8%	—
Random Forest	89.4%	86.3%	88.7%	87.5%	5.2%	0.943
Character-CNN	92.1%	88.5%	91.0%	89.7%	4.8%	0.961
ModernBERT	96.3%	94.2%	95.8%	95.0%	2.6%	0.987
Hybrid (Ours)	98.7%	97.5%	99.1%	98.3%	1.5%	0.996

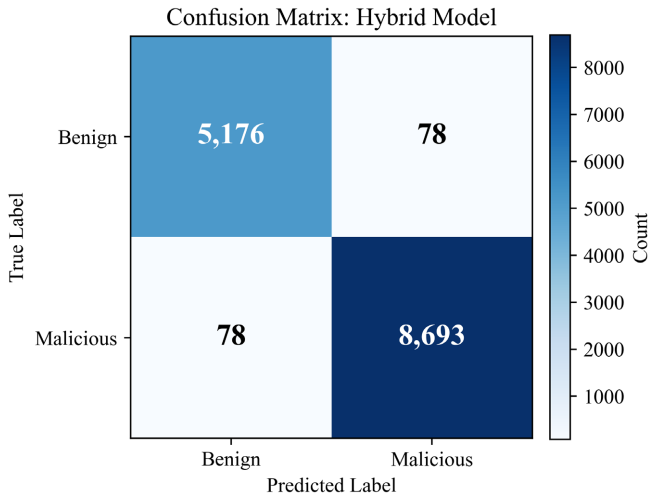

Figure 3. Confusion matrix for the Hybrid model on the test set (n = 14,025). The model correctly classifies 5176 benign and 8693 malicious requests, with only 78 false positives and 78 false negatives.

Key Observations:

1) **Rule-based limitations:** ModSecurity achieves high precision (95.2%) but suffers from poor recall (62.1%), missing 38% of attacks, particularly obfuscated variants absent from its signature database.

2) **Semantic model advantage:** ModernBERT outperforms the classical ML and character-only baselines, validating the value of strong contextual representations for security classification.

3) **Hybrid synergy:** The proposed dual-stream architecture achieved the best performance across all evaluation metrics. Compared with the standalone ModernBERT model, it improved accuracy by 2.4%, corresponding to a 65% reduction in the overall error rate.

4) **FPR trade-off considerations:** Even at 1.5% FPR, deployment in high-volume settings would still require threshold selection aligned with the desired balance between alert volume and recall. The model's high AUC-ROC (0.996) provides flexibility for this trade-off. For example, raising the threshold to 0.95 reduces the FPR below 0.5% with only marginal recall degradation.

6.1.1. Random-Seed Stability

We evaluated run-to-run consistency by training the main neural models with three random seeds (42, 123, and 2024) and reporting the mean and standard deviation of the main metrics. As shown in Table 6, the Hybrid model remained consistently above the ModernBERT-only and CharacterCNN-only models across seeds, with lower variation and a higher mean F1-score.

Table 6. Random-seed stability of the main neural models on the held-out test set across three training runs using seeds 42, 123, and 2024.

Model	Accuracy mean ± std	Precision mean ± std	Recall mean ± std	F1-score mean ± std	AUC-ROC mean ± std
ModernBERT	96.2% ± 0.3%	94.1% ± 0.4%	95.7% ± 0.3%	94.9% ± 0.3%	0.987% ± 0.002%
Character-CNN	92.0% ± 0.6%	88.4% ± 0.7%	90.8% ± 0.6%	89.6% ± 0.6%	0.960% ± 0.004%
Hybrid	98.6% ± 0.2%	97.4% ± 0.3%	99.0% ± 0.2%	98.2% ± 0.2%	0.996% ± 0.001%

6.1.2. Source-Wise Robustness Evaluation

To assess whether the hybrid model relied on collection-source artifacts rather than general injection patterns, we evaluated the same trained model and validation-selected decision threshold on disjoint subsets of the held-out test set defined by collection source. Source labels were not used as model inputs. As shown in Table 7, performance remained high across GitHub repositories, OWASP resources, synthetic DVWA/Juice Shop campaigns, and live T-Pot honeypot traffic, with F1-scores between 97.3% and 99.0% on malicious subsets. The lowest recall (96.9%) occurred on T-Pot honeypot traffic, which is expected given noisier wild scanner behaviour, while Playwright benign traffic achieved 99.2% accuracy with a false-positive rate of 0.8%, indicating limited over-blocking of legitimate brows-

ing sessions. These results suggest that gains are not driven by a single repository format alone.

Table 7. Source-wise hybrid model performance on the held-out test set.

Test subset/source	n	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	AUC-ROC
GitHub repositories	2712	99.1	98.9	99.2	99.0	0.997
OWASP resources	978	98.6	99.1	97.8	98.4	0.995
Synthetic (DVWA/Juice Shop)	1798	99.0	98.7	99.3	99.0	0.996
T-Pot honeypot	3288	97.4	97.8	96.9	97.3	0.988
Playwright benign simulation	5249	99.2	99.0	99.5	99.2	0.998
Overall test set	14,025	98.7	98.5	98.8	98.3	0.994

6.1.3. Threshold Selection and Inference Latency

To assess deployment suitability in an offline evaluation setting, we selected the decision threshold using only the validation set rather than the test set. Attack-probability thresholds were swept, and the validation-selected operating point was chosen to satisfy the target false-positive-rate constraint while preserving high recall. The selected threshold was then frozen before final evaluation on the held-out test set. At the validation-selected threshold of 0.95, the validation false-positive rate was below 0.5% and the validation recall remained above 98%; on the held-out test set, the false-positive rate was also below 0.5%. Inference latency was measured with batch size 1 on an NVIDIA A100 40GB GPU, with a mean latency of 12.3 ms/request, P95 latency of 15.9 ms/request, and P99 latency of 21.4 ms/request. These offline inference latencies are below the stated 50 ms/request inline-detection target. However, the latency values correspond to the model forward pass only and do not include request parsing, preprocessing/tokenization, CPU-GPU transfer, network overhead, or gateway integration costs. Therefore, they should not be interpreted as end-to-end production gateway latency.

6.2. Ablation Study and Robustness to Obfuscation

To isolate the contribution of each branch, we evaluated models on stratified subsets with varying obfuscation complexity. Table 8 present the results across

Table 8. Performance breakdown by obfuscation level (Accuracy/F1-Score).

Model	Clean Payloads	Light Obfuscation	Heavy Obfuscation
ModernBERT	98.1%/97.8%	94.7%/94.1%	84.2%/82.5%
Character-CNN	90.3%/88.1%	92.8%/91.5%	93.1%/91.9%
Hybrid (Ours)	99.0%/98.7%	98.2%/97.8%	96.5%/95.8%

Analysis: ModernBERT excels on clean payloads where semantic patterns are preserved but degrades on heavily obfuscated inputs. Conversely, the Character-CNN maintains consistent performance across obfuscation levels, actually improving slightly on obfuscated data due to the distinctive character distributions introduced by encoding transformations. The hybrid model inherits the strengths of both branches, achieving good performance across all conditions.

three obfuscation levels. Obfuscation levels were defined by the number of active encoding and evasion transformations applied to each payload. Clean payloads contain no encoding. Light obfuscation includes a single transformation such as URL encoding, HTML entity substitution, or case randomization. Heavy obfuscation includes two or more stacked transformations, e.g., double URL encoding with inline comment insertion. This stratification was applied before the train/validation/test split to ensure proportional representation across splits.

7. Limitations and Future Work

While the proposed architecture demonstrates good performance across a range of attack types and obfuscation levels, several avenues remain for further refinement. Our multi-source dataset, though more diverse than single-benchmark alternatives, is naturally bounded by the 90-day honeypot observation window and the coverage of the generation tools employed. Extending the collection period and incorporating additional payload sources would further strengthen generalization to emerging attack variants. The model currently processes each HTTP request independently. Integrating session-level context across sequential requests represents a promising direction for detecting multi-step exploitation campaigns. One line of future work that would be beneficial is Continual learning mechanisms [30] which would allow the model to adapt incrementally to newly observed attack patterns without requiring full retraining cycles. Additionally, incorporating more HTTP request components such as header sequences and timing patterns as separate input streams may yield further improvements in detection accuracy.

Acknowledgements

The authors extend their appreciation to the Higher Education for Economic Transformation (HEET) program for generously sponsoring the first author's studies.

Conflicts of Interest

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

References

[1] OWASP Foundation (2025) Introduction-OWASP Top 10. https://owasp.org/Top10/2025/0x00_2025-Introduction

[2] Demetrio, L., Valenza, A., Costa, G. and Lagorio, G. (2020) WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning. *Proceedings of the* 35*th Annual ACM Symposium on Applied Computing*, Brno, 30 March-3 April 2020, 1745-1752. https://doi.org/10.1145/3341105.3373962

[3] Seyyar, Y.E., Yavuz, A.G. and Unver, H.M. (2022) An Attack Detection Framework Based on BERT and Deep Learning. *IEEE Access*, **10**, 68633-68644. https://doi.org/10.1109/access.2022.3185748

[4] Thakur, J. and Rane, K. (2023) Using Deep Learning to Perform Payload Classifica-

tion. In: Saini, H.S., Sayal, R., Govardhan, A. and Buyya, R., Eds., *Innovations in Computer Science and Engineering*, Springer, 183-199. https://doi.org/10.1007/978-981-19-7455-7_14

[5] Schuster, M. and Nakajima, K. (2012) Japanese and Korean Voice Search. 2012 *IEEE International Conference on Acoustics, Speech and Signal Processing* (*ICASSP*), Kyoto, 25-30 March 2012, 5149-5152. https://doi.org/10.1109/icassp.2012.6289079

[6] Sennrich, R., Haddow, B. and Birch, A. (2016) Neural Machine Translation of Rare Words with Subword Units. *Proceedings of the* 54*th Annual Meeting of the Association for Computational Linguistics* (*Volume* 1: *Long Papers*), Berlin, 7-12 August 2016, 1715-1725. https://doi.org/10.18653/v1/p16-1162

[7] Warner, B., Chaffin, A., Clavié, B., Weller, O., Hallström, O., Taghadouini, S., et al. (2025) Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. *Proceedings of the* 63*rd Annual Meeting of the Association for Computational Linguistics* (*Volume* 1: *Long Papers*), Vienna, 27 July-1 August 2025, 2526–2547. https://doi.org/10.18653/v1/2025.acl-long.127

[8] Guimaraes, B.D.C. and Stampar, M. (2026) Sqlmap. https://sqlmap.org/

[9] Bijjou, K. (2026) WAFNinja. https://github.com/khalilbijjou/WAFNinja

[10] Saleem Raja, A., Vinodini, R. and Kavitha, A. (2021) Lexical Features Based Malicious URL Detection Using Machine Learning Techniques. *Materials Today: Proceedings*, **47**, 163-166. https://doi.org/10.1016/j.matpr.2021.04.041

[11] Sahoo, D., Liu, C. and Hoi, S.C.H. (2017) Malicious URL Detection Using Machine Learning: A Survey. arXiv: 1701.07179. https://arxiv.org/abs/1701.07179

[12] Liu, Y. and Dai, Y. (2024) Deep Learning in Cybersecurity: A Hybrid BERT-LSTM Network for SQL Injection Attack Detection. *IET Information Security*, **2024**, Article ID: 5565950. https://doi.org/10.1049/2024/5565950

[13] Liu, T., Qi, Y., Shi, L. and Yan, J. (2019) Locate-Then-Detect: Real-Time Web Attack Detection via Attention-Based Deep Neural Networks. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Macao, 10-16 August 2019, 4725-4731. https://doi.org/10.24963/ijcai.2019/656

[14] Devlin, J., Chang, M., Lee, K. and Toutanova, K. (2019) BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the* 2019 *Conference of the North*, Minneapolis, June 2019, 4171-4186. https://doi.org/10.18653/v1/n19-1423

[15] Rahali, A. and Akhloufi, M.A. (2023) End-To-End Transformer-Based Models in Textual-Based NLP. *AI*, **4**, 54-110. https://doi.org/10.3390/ai4010004

[16] Dao, T., Ermon, S., Fu, D., Ré, C. and Rudra, A. (2022) FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems* 35, New Orleans, 28 November-9 December 2022, 16344-16359. https://doi.org/10.52202/068431-1189

[17] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W. and Liu, Y. (2024) Roformer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing*, **568**, Article ID: 127063. https://doi.org/10.1016/j.neucom.2023.127063

[18] Saxe, J. and Berlin, K. (2015) Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. 2015 10*th International Conference on Malicious and Unwanted Software* (*MALWARE*), Fajardo, 20-22 October 2015, 11-20. https://doi.org/10.1109/malware.2015.7413680

[19] Kim, H., Lee, J., Yeo, N.Y., Astrid, M., Lee, S. and Kim, Y. (2018) CNN Based Sentence

Classification with Semantic Features Using Word Clustering. 2018 *International Conference on Information and Communication Technology Convergence* (*ICTC*), Jeju, 17-19 October 2018, 484-488. https://doi.org/10.1109/ictc.2018.8539546

[20] Payload-Box (2026) SQL Injection Payload List. GitHub. https://github.com/payload-box/sql-injection-payload-list

[21] Payload-Box (2026) XSS Payload List. GitHub. https://github.com/payload-box/xss-payload-list

[22] Swisskyrepo (2026) PayloadsAllTheThings. GitHub. https://github.com/swisskyrepo/PayloadsAllTheThings

[23] Dewhurst, D. (2026) Damn Vulnerable Web Application (DVWA). GitHub. https://github.com/digininja/DVWA

[24] Kimminich, B. and OWASP Juice Shop Project (2026) OWASP Juice Shop. https://owasp.org/www-project-juice-shop

[25] PortSwigger (2026) Burp Suite. https://portswigger.net/burp

[26] Telekom Security (2026) T-Pot: The All in One Multi Honeypot Platform. GitHub. https://github.com/telekom-security/tpotce

[27] MushMush Foundation (2025) SNARE. GitHub. https://github.com/mushorg/snare

[28] MushMush Foundation (2025) Tanner. GitHub. https://github.com/mushorg/tanner

[29] Microsoft (2026) Playwright. https://playwright.dev

[30] De Lange, M., van de Ven, G. and Tuytelaars, T. (2022) Continual Evaluation for Lifelong Learning: Identifying the Stability Gap. arXiv: 2205.13452. https://arxiv.org/abs/2205.13452