



Risk-Calibrated Mitigation of XSS and SQL Injection in JSON API Traffic Using Deep Learning and Class-Specific Thresholds

Ali Kamal Taqi^{*}, Hassan Jaleel Hassan^{}, Ghaidaa Muttasher Abdulsahab^{}

Computer Network and Cyber Security Department, Computer Engineering College, University of Technology, Baghdad 10001, Iraq

Corresponding Author Email: ali.k.taqi@uotechnology.edu.iq

Copyright: ©2026 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ijss.160811>

ABSTRACT

Received: 16 June 2026

Revised: 31 July 2026

Accepted: 6 August 2026

Available online: 31 August 2026

Keywords:

API security, deep learning, risk-calibrated mitigation, SQL Injection, threshold calibration, Cross-Site Scripting detection

Deep learning detects Cross-Site Scripting (XSS) and SQL Injection (SQLi) accurately, but accuracy alone does not tell an Application Programming Interface (API) gateway or Web Application Firewall (WAF) what to do with a request. This paper presents a risk-calibrated mitigation framework that converts detector outputs into auditable Allow, Block and Review actions using temperature scaling, class-specific thresholds and a margin-based Review band. The detector is a Bidirectional Long Short-Term Memory network (BiLSTM)-Attention model with JavaScript Object Notation (JSON)-aware canonicalization and few-shot adversarial fine-tuning. Evaluation uses a three-class corpus of 248,389 HTTP payloads, deduplicated at exact, near-duplicate and template level before training to 180,730 records: a conventional random split shares 16.7% exact, 29.4% near-duplicate and 54.9% template-level payloads between training and test, so the corpus is re-split with template groups kept whole until cross-split overlap at all three levels is zero. The temperature ($T = 0.489$) and the threshold vector ($0.40 / 0.40 / 0.60$) are fitted on one half of the validation split and reported on the other, giving a fit-to-report macro-F1 gap of 0.0023. On the deduplicated test split, the calibrated system reaches macro-F1 0.9862 and accuracy 0.9851, and temperature scaling reduces the expected calibration error from 0.038 to 0.011. Against a bounded black-box adversary re-optimized against the deployed pipeline rather than against an earlier model, 4.00% of 2,000 malicious seed payloads are Allowed (95% upper bound 4.95%), compared with 80.30% for the same architecture without canonicalization; canonicalization, not fine-tuning, is the dominant robustness contributor. Median end-to-end latency is 4.0 ms. Calibration and thresholding therefore provide a practical, auditable bridge between deep-learning classification and deployable web-security decisions.

1. INTRODUCTION

Modern web applications rely on Application Programming Interface (API)-based communication, in which user-controlled data arrives through query parameters, headers, cookies, form fields and structured request bodies. JavaScript Object Notation (JSON) dominates this traffic because it supports lightweight, machine-readable exchange between clients, mobile applications, backend services and cloud platforms. The design improves interoperability but widens the input-validation attack surface, because malicious payloads can hide inside nested objects, arrays, serialized values or encoded strings rather than appearing as simple parameters [1, 2]. Cross-Site Scripting (XSS) and SQL Injection (SQLi) remain the most persistent input-validation attacks in this setting. They target different execution contexts, a browser and a database respectively, but share one weakness: untrusted input reaches a sensitive sink without adequate validation, sanitization, canonicalization, or contextual encoding [2, 3]. In JSON API traffic, the same malicious intent appears under many surface forms, including URL encoding,

HTML entities, Unicode confusables, whitespace variation, comment padding, case variation and JSON wrapping, and that variability is what makes reliable detection difficult.

Deep-learning detectors perform well on this task. Sequence models, including Convolutional Neural Network (CNN), Long Short-Term Memory network (LSTM), and Bidirectional Long Short-Term Memory network (BiLSTM), attention architectures and multi-feature fusion models, learn attack-related patterns such as SQL operators, delimiters, HTML tags, JavaScript fragments and encoded characters [4, 5]. High clean-test accuracy is not sufficient for deployment, however. A detector may classify benchmark samples correctly yet fail once an attacker applies semantics-preserving transformations, and black-box and reinforcement-learning mutation methods are known to produce evasive XSS and SQLi variants against learned detectors and WAF-like systems [6, 7]. Deployment also demands more than a predicted class: an API gateway must allow, block, challenge, or escalate every request, and a maximum-probability classifier cannot express that policy. What is needed is risk-calibrated mitigation, in which confidence is interpreted through class-

specific thresholds and translated into auditable actions. Calibration is the prerequisite, because neural networks produce confidence scores that need not reflect the true likelihood of correctness; temperature scaling corrects this without retraining [8]. It must be tied to the decision thresholds, because the cost of a missed SQLi is not the cost of a blocked benign request [9, 10].

This paper presents a risk-calibrated mitigation framework for XSS and SQLi detection in JSON-oriented API traffic. The framework uses a trained deep-learning detector as a scoring component, converts its outputs into calibrated probabilities by temperature scaling, and applies class-specific thresholds with an explicit review margin to produce Allow, Block and Review actions. The mitigation layer applies the threshold vector 0.40/0.40/0.60 for Normal, SQLi and XSS respectively, together with a review margin of 0.30 that withholds near-miss requests from the Allow action. The selected constants are fitted on one half of the validation split and every reported figure is computed on data disjoint from that fit.

The contributions are as follows. The paper formulates XSS and SQLi defense as a mitigation decision problem rather than a three-class classification problem; applies class-specific thresholds with an explicit review margin so that the policy reflects different operational risks; defines an auditable Allow/Block/Review layer; evaluates the calibrated system on a corpus deduplicated at template level, against an adversary re-optimized against the deployed pipeline, and under a latency budget; and reports how much the evaluation protocol itself, rather than the model, determines the headline numbers.

2. RELATED WORK

2.1 Web attack detection in Application Programming Interface-oriented traffic and adversarial robustness

XSS and SQLi persist because both exploit weak processing of untrusted input, and in API-oriented systems that input arrives in query parameters, headers, cookies, form values or structured JSON bodies, so payloads can hide inside nested fields, serialized values, arrays or encoded strings [1, 2]. Detection has moved from handcrafted features such as keyword counts, character frequencies, n-grams and entropy toward deep sequence models that learn directly from character or token representations: CNNs capture local fragments such as `<script>`, `onerror`, `UNION` or encoded delimiters, while LSTM and BiLSTM models capture longer-range dependencies and attention focuses on high-risk regions of long payloads [3-5]. Unified models handle XSS and SQLi in a single multi-class pipeline with strong clean-test performance [4, 5]. Clean performance, however, does not describe operational security: a detector in an API gateway must also process encoded, obfuscated and adversarially modified requests, which motivates a shift from classification evaluation toward mitigation evaluation.

Security classifiers operate in adversarial environments. Attackers can deliberately modify payload representation while preserving malicious intent. Examples include URL encoding, HTML entity encoding, Unicode homoglyph substitution, whitespace insertion, comment padding, keyword splitting, case variation, and JSON wrapping. These transformations can significantly change the surface form observed by a detector without necessarily changing the attack objective.

A parallel line of work concerns how such defenses should be evaluated rather than built. Tramèr et al. [11] show that defenses evaluated only against attacks generated for an earlier or different model routinely appear far more robust than they are, and that a defense is properly tested only by an attacker re-optimized against the deployed system. Decision-based attacks [12] make this feasible in the black-box setting relevant to a deployed WAF, since they require only the system's output decision rather than gradients or class probabilities. The evaluation in Section 4.5 follows both principles.

Surveys of adversarial machine learning in cybersecurity show that learned security systems are vulnerable to evasion when robustness is not explicitly evaluated [13, 14]. In the web-attack domain, black-box methods generate XSS payloads that bypass detection models [6], and reinforcement-learning methods learn transformation sequences that raise the bypass probability for both XSS and SQLi [7, 15]. High clean accuracy can therefore create a false sense of security, and robust systems increasingly add canonicalization, adversarial testing and adversarial fine-tuning. Even robust classification is not sufficient by itself: a deployable system must still decide whether to allow, block or review each request, and recent work in this journal points the same way, applying deep learning to malicious network-traffic detection [16] and hybrid machine learning to zero-day intrusion detection [17]. Pallikonda et al. [18] combine autoencoders with LSTM networks for real-time anomaly detection in IoT network traffic, and Ibrahim and Ghanim [19] show that convolutional and support-vector models detect blackhole routing attacks with near-perfect accuracy, reinforcing the case for deep learning as the basis of operational attack detection.

2.2 Web Application Firewall-oriented defense, operational decision-making and calibration

Web Application Firewalls (WAFs) sit between clients and protected applications. Traditional ones rely on signatures, regular expressions, anomaly scoring and rule-based filtering, which are effective against known attack forms but brittle against encoding, obfuscation and adaptive mutation; the OWASP Core Rule Set is the widely used rule-based foundation [20]. Recent work combines rule-based control with machine learning, grammar-aware filtering or adversarial hardening: PROGESI strengthens SQLi prevention through structure-aware enforcement [21], and ModSec-AdvLearn explores adversarially robust machine-learning support for adaptive SQLi payloads in ModSecurity-like settings [22]. Both support the view that practical web defense should be layered rather than dependent on a single classifier.

WAF-like deployment requires explicit operating decisions. A model's class prediction must be translated into security actions such as Allow, Block, Review, Challenge, or Log, which requires the system to weigh model confidence, class-specific risk, false-negative cost, false-positive cost, and the role of uncertainty. A request that is not confidently benign should not be treated as safe, especially when the competing class is SQLi or XSS.

Neural networks can produce probability scores that are poorly calibrated, meaning that the predicted confidence does not reliably match the empirical likelihood of correctness. This is especially problematic in security systems because the probability output may directly influence whether a request is allowed or blocked. Temperature scaling has been proposed as a simple and effective post-processing method for calibrating

neural-network confidence scores without changing the underlying model parameters [8]. Earlier calibration approaches, such as Platt scaling and multiclass probability calibration, also show the importance of transforming raw classifier scores into more reliable probability estimates [23, 24].

Threshold selection is closely related to calibration. In ordinary classification, the predicted label is the maximum-probability class, which is unsuited to risk-sensitive deployment: a request scored 0.42 Normal, 0.39 SQLi and 0.19 XSS is classified as Normal even though the SQLi confidence is operationally concerning. Class-specific blocking thresholds alone do not resolve it either, since 0.39 clears no blocking threshold while 0.42 clears the Allow threshold. Section 3.5 therefore adds a margin band that routes this pattern to Review.

Class-specific thresholds provide a more flexible decision mechanism. Instead of using one global threshold, the system can require different confidence levels for Normal, SQLi, and XSS. This is useful because different errors have different operational consequences. A SQLi false negative may allow database compromise, while a benign false positive may block a legitimate user. Classification metrics such as precision, recall, F1-score, macro-F1, and confusion matrices are therefore necessary for evaluating threshold behavior [9]. ROC analysis and threshold-based evaluation further support the need to study operating points rather than relying only on accuracy [10].

2.3 Research gap and paper positioning

Across these four strands, the same gap recurs. Detection accuracy, adversarial robustness, probability calibration and operational decision-making are studied separately, and the step that converts a calibrated probability into an auditable security action is left implicit. Evaluation practice compounds the gap: benchmarks are rarely checked for near-duplicate or template-level contamination, and adversarial claims are often supported by attacks generated against a different model from the one deployed. This paper addresses both, treating threshold calibration as the mitigation mechanism and reporting the evaluation protocol as a result in its own right.

3. PROPOSED RISK-CALIBRATED MITIGATION FRAMEWORK

The framework has four stages: a request is canonicalized, scored by a trained BiLSTM-Attention detector, converted into calibrated probabilities by temperature scaling, and mapped to an action by class-specific thresholds and a review margin. The first three estimate how likely each class is; only the fourth decides what to do, and keeping them separate is what makes the policy auditable and adjustable without retraining.

3.1 Input and canonicalization stage

Let x represent a raw web payload or API request content. The input may be a raw string, URL parameter, header value, form field, or JSON request body. Since the target deployment setting is JSON-oriented API traffic, the input may contain nested objects, arrays, escaped strings, encoded characters, or attack-bearing values embedded inside structured fields.

Before inference, the input is processed by a canonicalization function:

$$x_c = C(x) \quad (1)$$

where, $C(\cdot)$ denotes the JSON-aware canonicalization layer and x_c is the canonicalized input. The purpose of this stage is to reduce representation-level variance before the detector sees the payload. The canonicalization layer handles supported forms of URL encoding, HTML entity encoding, selected Unicode/confusable forms, whitespace and comment variation, and deterministic JSON serialization.

3.2 Deep learning scoring layer

After canonicalization, the input is encoded into a model-ready sequence representation and passed to the trained detector. Let f_θ denote the selected BiLSTM-Attention model with parameters θ . The detector produces a vector of raw logits:

$$z = f_\theta(x_c) \quad (2)$$

$$z = (z_{Normal}, z_{SQLi}, z_{XSS}) \quad (3)$$

The three target classes are:

$$K = \{Normal, SQLi, XSS\} \quad (4)$$

The raw logits are not used directly as deployment decisions; they are passed to the calibration stage, because neural-network scores may be poorly calibrated and a high raw score need not correspond to reliable operational confidence.

3.3 Temperature scaling

The framework applies temperature scaling to transform raw logits into calibrated probabilities. Temperature scaling is a post-processing method that adjusts the sharpness of the predicted probability distribution without retraining the model [8].

Given logits z and temperature parameter $T > 0$, the calibrated probability for class k is:

$$p_k = \exp(z_k/T) / \sum_{j \in K} \exp(z_j/T) \quad (5)$$

If $T > 1$, the probability distribution becomes softer. If $T < 1$, the probability distribution becomes sharper. The temperature value is fitted using the validation set and then frozen before the final test and adversarial evaluation. The detector parameters are not changed during this step.

For the selected robust configuration, the temperature is fitted on the `val_fit` half of the validation split by minimising the negative log-likelihood with the L-BFGS optimizer, following Guo et al. [8], and is then frozen. The fitted value is $T = 0.4889$. Because $T < 1$, the softmax distribution is sharpened, and because the transformation is strictly monotone, it preserves the ranking of classes within a request, so the predicted label is unchanged and only the confidence that the thresholds act on is recalibrated.

The calibrated probability vector is:

$$p = (p_{Normal}, p_{SQLi}, p_{XSS}) \quad (6)$$

3.4 Class-specific thresholds

Instead of applying a single global threshold or relying on the maximum-probability class, the framework uses separate thresholds for each class:

$$\tau = (\tau_{Normal}, \tau_{SQLi}, \tau_{XSS}) \quad (7)$$

Instead of one global threshold, each class carries its own confidence requirement, because the operational cost of an error differs by class: a missed SQLi may expose the database, while a blocked benign request denies service to a legitimate user. A fourth constant, the review margin, is a floor on the malicious score above which a request is treated as uncertain rather than allowed; because it applies to whichever malicious class scores higher, the resulting band runs from the margin up to that class's own blocking threshold. For the selected robust configuration, the constants fitted in Section 5.2 are:

$$\tau_{Normal} = 0.40, \tau_{SQLi} = 0.40, \tau_{XSS} = 0.60 \quad (8)$$

$$\tau_{review} = 0.30 \quad (9)$$

The thresholds are selected on validation data by grid search over the temperature-scaled probabilities, maximising macro-F1, and are then frozen for all clean-test, adversarial and latency evaluation. Section 4.4 gives the protocol and Section 5.2 the resulting operating point.

The thresholds are selected on val_fit alone and are not adjusted after seeing val_report, clean-test or adversarial results. Freezing them before any reporting split is scored avoids optimistic bias and makes the operating point auditable: the constants that govern deployment are fixed artifacts, versioned alongside the model.

The candidate threshold values form a grid over {0.40, 0.45, ..., 0.90} for each class (step 0.05), giving $11^3 = 1,331$ combinations. Each combination is scored on the calibrated validation probabilities, and the vector that maximizes validation macro-F1 is retained. The search space, the resulting operating-point comparison, and the sensitivity of macro-F1 and the Review rate to the thresholds are reported in Section 5.2.

3.5 Mitigation decision logic

The calibrated mitigation layer converts calibrated probabilities into one of three operational actions: Allow, Block, or Review. The decision rule prioritizes malicious classes before allowing traffic.

Malicious classes are evaluated before benign ones, and a request whose strongest malicious probability falls inside the review margin is withheld from Allow even when the Normal probability would otherwise clear its threshold. Using the selected constants, the decision logic is:

- (1) if $p_{SQLi} \geq 0.40 \rightarrow$ Block as SQLi
- (2) else if $p_{XSS} \geq 0.60 \rightarrow$ Block as XSS
- (3) else if $\max(p_{SQLi}, p_{XSS}) \geq 0.30 \rightarrow$ Review
- (4) else if $p_{Normal} \geq 0.40 \rightarrow$ Allow
- (5) else $\rightarrow$ Review

The rule differs from argmax classification in two ways. Malicious thresholds are checked first, so a request with a sufficiently high SQLi or XSS probability is blocked even

when another class scores similarly. The review margin in (3) then captures the near-miss region that neither blocks nor confidently allows. Without (3), the Review action is effectively unreachable at this operating point: on the clean test split it fires on 0 of 27,109 requests, because a request whose malicious probabilities are both below their thresholds almost always retains a Normal probability above 0.40. The margin is what makes the third action operative, and Section 5.3 quantifies the trade-off it buys.

3.6 Algorithm and security interpretation

Allow lets the request continue to the protected application, and Block stops it. Review marks it uncertain: not confidently malicious enough to block, not confidently benign enough to allow. In deployment, it may trigger logging, secondary WAF inspection, analyst review, rate limiting or a challenge, and its purpose is to stop low-confidence traffic from being silently treated as safe.

Algorithm 1: Risk-Calibrated Mitigation Decision

Input: x , raw payload or request content; C , canonicalization function; θ , trained BiLSTM-Attention detector; T , temperature parameter; τ_{Normal} , τ_{SQLi} and τ_{XSS} , class thresholds; τ_{review} , the review margin.

Output: Decision = Allow, Block, or Review; Label = Normal, SQLi, XSS, or Uncertain.

Procedure: apply canonicalization; encode the canonicalized input; obtain logits; apply temperature scaling; block if the SQLi probability clears its threshold; otherwise block if the XSS probability clears its threshold; otherwise route to Review if the larger malicious probability reaches the review margin; otherwise allow if the Normal probability clears its threshold; otherwise return Review.

The following examples illustrate the calibrated rule using $\tau = (0.40, 0.40, 0.60)$ and a review margin of 0.30.

Example 1: Clear SQLi. If $p_{Normal} = 0.12$, $p_{SQLi} = 0.81$ and $p_{XSS} = 0.07$, then $p_{SQLi} \geq 0.40$ and the decision is Block SQLi.

Example 2: Clear XSS. If $p_{Normal} = 0.18$, $p_{SQLi} = 0.10$, and $p_{XSS} = 0.72$, then $p_{XSS} \geq 0.6$ and the decision is Block XSS.

Example 3: Benign request. If $p_{Normal} = 0.86$, $p_{SQLi} = 0.08$, and $p_{XSS} = 0.06$, then neither malicious threshold is exceeded and $p_{Normal} \geq 0.4$. The decision is Allow.

Example 4: Uncertain request. If $p_{Normal} = 0.42$, $p_{SQLi} = 0.39$ and $p_{XSS} = 0.19$, no class clears its blocking threshold, but $\max(p_{SQLi}, p_{XSS}) = 0.39$ reaches the review margin of 0.30 without clearing the SQLi threshold of 0.40, so (3) applies and the decision is Review. This is the case raised in Section 2.2: under blocking thresholds alone the request would have been allowed.

The framework improves deployment suitability in four ways. It makes the decision policy explicit, so thresholds can be audited, versioned and changed under operational control. It separates scoring from enforcement. It routes low-confidence requests to Review rather than allowing them. And it supports different risk tolerances, since a deployment that fears false negatives can lower the malicious thresholds or treat Review as a temporary block, while one that fears false positives can do the opposite. The novelty is the explicit treatment of threshold calibration as a mitigation mechanism rather than as a reporting convention.

4. EXPERIMENTAL SETUP

4.1 Experimental objective, dataset and splits

The experiments evaluate whether a trained detector can be converted into a practical mitigation system by temperature scaling, class-specific thresholds and the Allow/Block/Review rule. The question is not only whether the model separates Normal, SQLi and XSS, but whether its calibrated probabilities support deployment decisions. Four dimensions are assessed: calibration quality, clean-test performance, adversarial robustness under an adaptive attacker, and latency.

The experiments use a three-class web payload corpus of Normal, SQLi and XSS samples assembled from three public sources. The pooled corpus contains 248,389 payloads. Unlike the previous version of this work, the corpus is deduplicated at three normalisation levels and re-split with template-level group separation before any model is trained; the deduplicated corpus contains 180,730 payloads, divided 70/15/15 into 126,512 training, 27,109 validation and 27,109 test records (Table 1).

JSON-like requests make up 14.4% of the deduplicated test split (3,917 of 27,109). Most are produced by wrapping payloads from the source datasets into realistic JSON API request bodies and the remainder are naturally JSON, so the evaluation targets a representative JSON-oriented subset rather than a fully JSON-native corpus. This reflects the framework's focus on API traffic, where malicious strings appear inside nested fields, serialized values and encoded payloads. Detection performance on the JSON-like slice is reported separately in Section 5.3.

The auxiliary JSON sets, 100 curated JSON-native SQLi payloads and 1,500 JSON hard negatives, are held outside the pooled corpus; the hard negatives are mined from the training partition only (Section 4.2).

Data cleaning, labeling, deduplication and quality control: The corpus is assembled from three public datasets, CSIC 2010 HTTP, HttpParams and a combined SQLi/XSS/command-injection dataset, contributing 298,768 parsed rows in total (Table 2). The payload is the URL-decoded query string concatenated with any request content for CSIC, the parameter value for HttpParams, and the sentence field for the mixed dataset.

Labels are unified to Normal (0), SQLi (1) and XSS (2). CSIC Normal records map to Normal, while Anomalous records are typed by a curated regular-expression matcher (SQLi signatures such as union select, or 1=1 and drop table; XSS markers such as <script>, javascript: and onerror=), defaulting to Normal when no signature is present. HttpParams uses its native attack_type field, and the mixed dataset stores per-class scores from which the maximum-scoring class is taken. Because this study targets three classes, all command-injection and path-traversal samples are removed, leaving 248,389 payloads.

Each retained payload is stripped of surrounding whitespace, has internal whitespace collapsed and null bytes removed, and is JSON-canonicalized at inference time (Section 3.1).

The public source datasets contain repeated and templated payloads, so the three splits used in the previous version of this work were not independent. Three normalisation levels are therefore defined and measured. The exact key is the raw payload. The near key is the payload after canonicalization, lower-casing, whitespace collapse, and replacement of each

digit by 0 and of quoted string literals by a single placeholder. The template key additionally replaces every alphanumeric run in the near key by a single token, preserving punctuation and operators, so that payloads differing only in identifier or literal choice collapse together. Because the corpus is JSON API traffic, both masks are applied JSON-aware: object keys are preserved and only scalar values are normalised, since masking the serialized envelope instead collapses every body sharing a schema into one key.

Measured on the conventional random split, the training partition shares 16.68% of the test split at exact level, 29.42% at near-duplicate level and 54.92% at template level, with the validation partition overlapping similarly (Table 3). More than half of the test split therefore shared a payload template with training. The pooled corpus is reduced to one record per near key, retaining the shortest payload. Near keys carrying more than one label are dropped whole: 127 keys covering 3,691 rows (1.49% of the pool), dominated by payloads that normalise to a bare number and appear with different labels in different sources. This is deliberately conservative, since it discards correctly labelled records to remove a smaller number of ambiguous ones. The remaining 180,730 records are re-split 70/15/15, class-stratified, with template groups assigned whole to one split, and exact, near-duplicate and template overlap across all three split pairs is then zero by construction and asserted after the fact.

Table 1. Deduplicated dataset partitions

Partition	Records	Normal	SQLi	XSS
Training	126,512	58,982	39,703	27,827
Validation	27,109	12,639	8,507	5,963
Test	27,109	12,639	8,507	5,963
Total	180,730	84,260	56,717	39,753

Note: Cross-Site Scripting (XSS); SQL Injection (SQLi)

Table 2. Corpus construction and deduplication pipeline

Stage	Operation	Records
Raw source rows	CSIC 61,065; HttpParams 31,067; mixed 206,636	298,768
Class conversion	Map to {Normal, SQLi, XSS}; drop command-injection and path-traversal	248,389
Text cleaning	Strip, collapse whitespace, remove null bytes	248,389
Near-duplicate collapse	One record per near key (shortest retained)	180,857
Label-conflict removal	Drop 127 near keys carrying >1 label (3,691 raw rows)	180,730
Grouped stratified split	70/15/15 with template groups kept whole	126,512 / 27,109 / 27,109

Table 3. Cross-split leakage before and after regrouping

Leakage Level	Train / Test	Train / Val	Val / Test
Exact duplicate	16.68%	16.57%	14.40%
Near duplicate	29.42%	29.47%	23.50%
Template level	54.92%	55.24%	46.97%
After regrouping	0.00%	0.00%	0.00%

Benchmark contamination of this kind is well documented in other fields and is not specific to web security. Barz and Denzler [25] found that 3.3% and 10% of the CIFAR-10 and CIFAR-100 test sets have duplicates in the corresponding training sets, and showed that the resulting bias distorts

comparisons between architectures; Lee et al. [26] report the same problem at scale in language corpora and show that removing near-duplicates changes measured performance. The template-level figure measured here is an order of magnitude larger than the CIFAR case, which is expected: web attack payloads are generated from a small number of exploit templates, so surface variation overstates the diversity of the corpus.

A residual label-quality limitation remains and is stated here rather than left implicit. After repeated URL-decoding and HTML unescaping, 6.42% of SQLi-labelled and 4.61% of XSS-labelled payloads in the pooled corpus contain no recognisable attack syntax; these are dominated by repeated-character strings, random-character fuzzing strings and bare redirect URLs inherited from the source datasets. The mixed dataset also concatenates short SQL fragments with unrelated English review text. The supervised labels are therefore noisy at the few-percent level, which bounds the precision of any result reported on this corpus, including ours. Section 5.4 reports the adversarial result separately on the subset of seed payloads whose attack syntax can be verified, so that the headline robustness number does not depend on these records.

4.2 Robustness and fine-tuning partitions

The adversarial and fine-tuning partitions are all built from the training split alone (Table 4). That split is first divided 90/10 into `train_main` (113,862 records) and `train_sel` (12,650), the latter reserved for epoch selection so that no training run reads the validation split. `train_main` is then divided 85/15, with template groups kept whole, into `train_core` (96,784) and `train_probe` (17,078); a probe detector is trained on `train_core` only, and `train_probe` is never trained on, so that fine-tuning material can be mined from a model's genuine errors without any validation or test record entering the process. The resulting few-shot adversarial tuning (FSAT) set is a hard-case adaptation set rather than a class-balanced sample: it draws on the probe's false negatives on `train_probe`, near-boundary cases (correct class predicted with true-class probability below 0.75), evasions mined by attacking the probe with `train_probe` payloads as seeds, and JSON command-injection-like hard negatives mined from `train_core`.

Few-shot adversarial fine-tuning set and protocol:

The realised provenance is recorded rather than assumed (Table 5); the probe-error column combines 22 false negatives with 8 near-boundary cases, all XSS. The mining pool yielded 7,556 mutations from 512 distinct originals, of which 2,139 evaded the probe as SQLi but only 5 as XSS, so the SQLi quota is filled entirely from mined evasions while the XSS quota falls back on errors and a class-balanced back-fill from `train_core`. That asymmetry is itself informative: under JSON-aware canonicalization, XSS payloads are markedly harder to mutate into evasions. Every candidate is filtered against the exact, near and template key sets of the validation and test splits, and the assembled set is asserted disjoint from both at all three levels.

Fine-tuning starts from the selected detector checkpoint and updates all parameters with AdamW at a learning rate of $5e-5$ for three epochs, using a class-weighted focal loss ($\gamma = 2$) to emphasise minority and hard cases; batch size and maximum sequence length follow the base configuration. Only the detector is adapted, after which the temperature and thresholds are fitted as described in Section 4.4.

Table 4. Robustness and fine-tuning partitions

Partition	Purpose	Records
<code>train_sel</code>	Epoch selection, held out of training	12,650
<code>train_core</code> / <code>train_probe</code>	Probe training and error mining	96,784 / 17,078
RL mining pool	Evasions mined against the probe detector	7,556 rows, 512 seeds
FSAT set	Few-shot adversarial fine-tuning	1,499
Attack seed pool	Final adversarial evaluation, from the test split	2,000

Table 5. FSAT composition by class and provenance

Class	RL Evasion	Probe Errors	JavaScript Object Notation (JSON) Hard Neg.	Back-Fill	Total
Normal	0	0	999	0	999
SQLi	300	0	0	0	300
XSS	5	30	0	165	200
Total	305	30	999	165	1,499

4.3 Selected detector configuration and training protocol

Earlier model development compared BiLSTM-Attention and CharCNN detectors with canonicalization enabled and disabled; here the detector is treated as the scoring component of the mitigation framework. The selected deployment configuration is BiLSTM-Attention with JSON-aware canonicalization, FSAT and calibrated thresholds. Canonicalization is retained even though disabling it gives slightly better clean-test accuracy, because it is what resists representation-level adversarial transformation (Section 5.4).

Baseline training uses three seeds {42, 73, 101} so that results do not depend on one favourable initialisation, handles class imbalance with class-weighted cross-entropy, and selects its epoch on `train_sel`, a 12,650-record slice of the training split reserved for that purpose, using macro-F1, which is preferred here because security performance depends on preserving minority-class detection rather than majority-class accuracy [9]. The final post-FSAT model is the scoring model for the mitigation evaluation, and the calibration and thresholding layer is fitted afterwards on `val_fit` alone.

4.4 Calibration protocol and mitigation decision evaluation

Calibration is performed after model selection and the detector weights are fixed throughout. In the previous version of this work the temperature, the decision thresholds and the reported validation performance were all obtained from the same validation split, so the reported validation figures were not independent of the constants fitted on them. The validation split is therefore halved, grouped-stratified by template key, into `val_fit` and `val_report`. The temperature is fitted on `val_fit` by minimising the negative log-likelihood (L-BFGS) and frozen; the threshold grid is searched on `val_fit` and frozen; the review margin is selected on `val_fit` under a fixed benign-review budget. `val_report` and the clean test split are then scored with those frozen constants and are never used for any fit. The whole procedure is repeated for two further fit/report partitions to confirm that the operating point is not an artifact

of one partition.

Calibration and threshold-selection metrics:

Calibration quality is quantified with the expected calibration error (Expected Calibration Error (ECE), 15 equal-width confidence bins), the negative log-likelihood (NLL), and the multiclass Brier score, each computed before and after temperature scaling on the validation and clean-test splits. The temperature is fitted only on the validation logits and then frozen. Threshold selection searches the grid of Section 3.4 (1,331 combinations) on the calibrated validation probabilities and keeps the macro-F1-optimal vector; the same frozen temperature and thresholds are used for all clean-test, adversarial, and latency evaluation.

Calibrated probabilities are converted into decisions using the rule of Section 3.5. Two mappings are reported and should not be confused. Classification metrics use the thresholded prediction, the highest-scoring class among those clearing their own threshold, falling back to the argmax when none does. The Allow/Block/Review distribution follows the priority order of Section 3.5, in which a malicious class clearing its threshold blocks the request even when another class scores higher; that is why the deployed action mapping reported in Section 5.3 blocks slightly more benign requests than the classifier misclassifies.

4.5 Adversarial evaluation setup

The adversarial evaluation uses a bounded black-box reinforcement-learning adversary that is re-optimized against the deployed system. This is a change of protocol from the previous version of this work, and it follows the evaluation principle set out by Tramèr et al. [11]: a defense tested only against attacks generated for a different model is not tested at all. The reason for the change is stated explicitly. Previously, a holdout of mutated payloads was generated against an earlier, canonicalization-disabled detector and the final model was then re-scored on that frozen set. A zero on such a set measures transfer from a weaker model, not robustness against an attacker who targets the deployed pipeline. In addition, an audit of the generator used for that holdout found that it retained only the first eight payloads of its pool at each search step, so the 2,882-payload holdout reported previously contained mutations of eight distinct originals; its effective sample size was eight, not 2,882. Both issues are corrected here.

The attacker's reward comes from the full mitigation pipeline: canonicalization, the final fine-tuned checkpoint, the frozen temperature, the frozen thresholds and the Allow/Block/Review rule. It observes only the resulting decision, which places it in the decision-based black-box family [12], the setting that matches a deployed gateway. A strict success is a malicious payload decided Allow; the weaker criterion, not Blocked, is reported separately. Because the independent unit is the distinct seed payload, rates are reported per payload with exact Clopper-Pearson bounds [27]. The adversary itself is an epsilon-greedy bandit over mutation chains: at each step it draws three to four transformations from an eleven-operator library, applies them to a mini-batch of 64 seed payloads, and retains the highest-reward chain found. Payloads that are not already JSON are wrapped into realistic API bodies before mutation, and JSON validity is re-checked after every transformation, so each generated example remains a syntactically valid request. Table 6 lists the full configuration.

For continuity with the previous version, the transfer protocol is also retained as a separate, explicitly labelled comparison: a holdout is generated against the pre-FSAT, canonicalization-disabled detector with the corrected generator and each configuration is re-scored on it. That holdout contains 9,923 rows from 987 distinct originals, against eight in the version reported previously.

Table 6. Adaptive black-box adversary configuration

Parameter	Setting
Attacker model	Epsilon-greedy bandit over mutation chains
Knowledge	Black box; observes only the Allow/Block/Review decision
Target	Canonicalization → final checkpoint → frozen temperature → frozen thresholds
Mutation library	11 semantics-preserving operators
Chain length	3–4 mutators per attempt
Search budget	3,000 steps × 64-payload mini-batch per campaign
Campaigns per configuration	9 (epsilon ∈ {0.1, 0.2, 0.3} × seeds {42, 73, 101})
Queries per configuration	1,728,000
Seed pool	2,000 malicious payloads from the deduplicated test split
Validity constraint	JavaScript Object Notation (JSON) re-parsed after each mutation; invalid mutations rejected
Strict success	Malicious payload decided Allow
Weak success	Malicious payload not blocked (Allow or Review)
Statistics	Per distinct seed payload, with exact Clopper-Pearson bounds

4.6 Evaluation metrics, latency benchmarking and leakage prevention

The evaluation reports precision, recall, F1-score, macro-F1, accuracy, P50 latency, and P95 latency. Macro-F1 is emphasized because the dataset is imbalanced and because security performance depends on maintaining strong detection across attack classes, not only on majority-class accuracy [9]. Threshold-based evaluation is also important because deployment decisions depend on selected operating points rather than argmax classification alone [10]. Calibration quality is additionally reported using the expected calibration error, negative log-likelihood, and Brier score (Section 5.1). For the adversarial evaluation the reported quantity is the proportion of distinct seed payloads that the attacker drives to Allow. Because that proportion is estimated from a finite pool and is small, it is reported with an exact Clopper-Pearson interval [27] rather than a normal approximation, which is unreliable near the boundary of the unit interval; the upper bound is the figure that should be carried into any deployment argument.

Latency is measured to assess whether the calibrated mitigation pipeline can operate in API-gateway or WAF-like environments. The selected configuration is benchmarked at payload sizes of 0.5 KB, 2.0 KB and 8.0 KB, with 200 iterations per size and a batch size of one, since a gateway scores a single request at a time. Both GPU and CPU figures are reported, and the measurement is taken in-process rather than through an HTTP wrapper, so it excludes network and serialisation overhead and isolates the cost the mitigation layer itself adds.

The benchmark is performed after model selection and exercises the same path assumed for deployment:

canonicalization, model inference, temperature scaling and the threshold decision. Canonicalization is timed separately as well, because it is the component whose cost grows with payload size.

Strict separation is maintained between training, calibration, fine-tuning and final evaluation, and is enforced by assertion rather than by convention. No revision run reads the validation split for training or epoch selection: base checkpoints select their epoch on `train_sel`, a 12,650-record slice held out of the training split for that purpose alone. FSAT material is mined only from `train_probe` and `train_core` and is asserted disjoint from validation and test at exact, near and template level. The temperature, thresholds and review margin are fitted on `val_fit` alone. The clean test split is used only for final reporting, and the adversarial seed pool is drawn from it after the operating point has been frozen.

The evaluation order is corpus deduplication and regrouped split → base training → probe training and error mining → FSAT construction → fine-tuning → temperature and threshold fitting on `val_fit` → reporting on `val_report` and clean test → adaptive adversarial evaluation → latency benchmark. Because the attacker targets the frozen pipeline, the adversarial stage comes last and cannot influence any constant used to report clean performance.

5. RESULTS AND DISCUSSION

5.1 Threshold calibration results and calibration quality

The first evaluation objective is whether the selected robust detector can be converted into a stable mitigation operating point using temperature scaling and class-specific thresholds. The temperature, the thresholds and the review margin are all fitted on `val_fit`; `val_report` and the clean test split are scored with those frozen constants and are used for reporting only.

The selected robust configuration is BiLSTM-Attention + JSON-aware canonicalization + FSAT + calibrated thresholds. The fitted temperature is $T = 0.4889$ and the selected threshold vector is $\tau = (0.40, 0.40, 0.60)$ for [Normal, SQLi, XSS], with a review margin of 0.30. On `val_fit`, this operating point attains macro-F1 = 0.9881; on the held-out half of the validation split it attains macro-F1 = 0.9859 and accuracy = 0.9848. The difference between the two, 0.0023, is the direct measure of how much the threshold search exploited the split it was tuned on.

Repeating the whole fit on two further validation partitions gives the same threshold vector in all three cases, temperatures of 0.4889, 0.4706, 0.4800 and `val_report` macro-F1 of 0.9859, 0.9866, 0.9871; the fit-to-report gap has mean 0.0009 and never exceeds 0.0023. The operating point is therefore a property of the model rather than of one partition (Table 7).

Temperature scaling improves the reliability of the detector's confidence without changing the ranking of classes within a request.

On the clean test split, ECE falls from 0.0382 to 0.0109, NLL from 0.0809 to 0.0625 and the Brier score from 0.0398 to 0.0287; `val_report` follows the same pattern (ECE 0.0402 to 0.0102 (Table 8)). Figure 1 shows the corresponding reliability diagrams: before scaling the accuracy exceeds the mean confidence in every populated bin, and after scaling the bin accuracies track the diagonal more closely. Because temperature scaling is strictly monotone, it preserves each sample's argmax label, so the classification metrics are

essentially unchanged; what changes is the confidence that the thresholds act on. The maximum calibration error is reduced far less than the average one on the test split (0.6231 to 0.5863), so a small number of sparsely populated high-confidence bins remain miscalibrated after scaling.

Table 7. Nested calibration protocol and performance by split

Split	Role	Macro-F1	Accuracy
val_fit	Fitted here	0.9881	0.9873
val_report	Reported only	0.9859	0.9848
Clean test	Reported only	0.9862	0.9851

Table 8. Calibration metrics before and after temperature scaling

Split	Metric	Before (T = 1.0)	After (T = 0.4889)
val_fit	ECE	0.0377	0.0121
val_fit	NLL	0.0758	0.0579
val_report	ECE	0.0402	0.0102
val_report	NLL	0.0814	0.0596
Test	ECE	0.0382	0.0109
Test	NLL	0.0809	0.0625
Test	Brier	0.0398	0.0287

Note: Expected Calibration Error (ECE); negative log-likelihood (NLL)

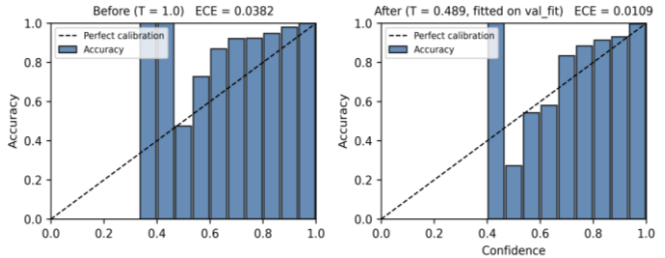


Figure 1. Reliability diagrams for the selected robust model on the clean test split, before (left) and after (right) temperature scaling ($T = 0.4889$)

Note: Bars show per-bin accuracy; the dashed line is perfect calibration.

5.2 Threshold operating-point analysis

The class-specific thresholds are chosen by grid search over $\{0.40, 0.45, \dots, 0.90\}$ per class (1,331 combinations) on the temperature-scaled `val_fit` probabilities, maximising macro-F1. The maximum, 0.9881, is attained by 21 grid points, all sharing $\tau_{\text{Normal}} = 0.40$ and $\tau_{\text{SQLi}} \leq 0.50$; the tie is broken toward the smallest vector in grid order, giving (0.40, 0.40, 0.60). Compared with the operating point reported in the previous version, the SQLi threshold moves from 0.60 to 0.40. The change is a consequence of deduplication rather than of a different search: with template-level duplicates removed the Normal-SQLi boundary is harder, and macro-F1 now prefers the more sensitive SQLi threshold.

Macro-F1 is nearly flat across a broad plateau of malicious thresholds (Figure 2), so the operating point is chosen for its decision behaviour rather than for a macro-F1 difference that the plateau cannot resolve. The Normal threshold acts as a safety dial (Figure 3): macro-F1 is stable up to about 0.40 and declines beyond it. The review margin is selected separately, on `val_fit`, as the widest band whose benign review burden stays within a fixed 1% budget; that yields $\tau_{\text{review}} = 0.30$, which on `val_fit` routes 0.39% of requests to Review. Table 9 reports the effect of this margin on the clean test split.

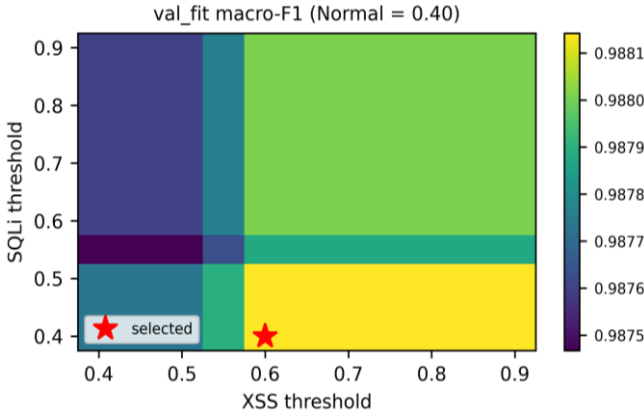


Figure 2. Macro-F1 on val_fit as a function of the SQL Injection (SQLi) and Cross-Site Scripting (XSS) thresholds (Normal threshold fixed at 0.40)

Note: The selected point lies on the macro-F1 plateau.

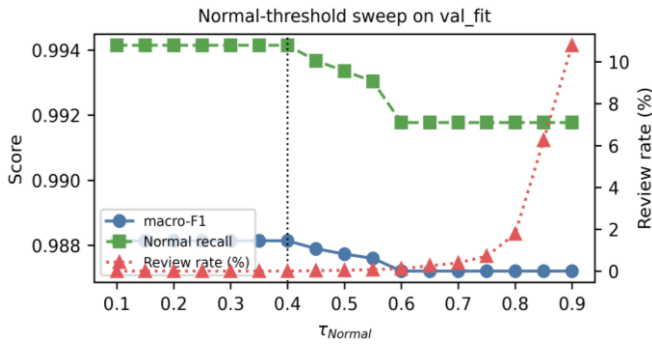


Figure 3. Effect of the Normal threshold on val_fit macro-F1 and on the Review rate

Table 9. Effect of the review margin on the clean test split

Margin τ_{review}	Review Rate	SQLi Held	Benign Held
none	0.000%	0 of 245	0
0.20	1.808%	43 of 245	444
0.30 (selected)	0.240%	6 of 245	59
0.40	0.059%	0 of 245	16
0.50	0.041%	0 of 245	11

Note: SQL Injection (SQLi)

5.3 Per-class, clean-test and diagnostic performance

Per-class performance on the held-out half of the validation split is strong across all three classes (Table 10). XSS reaches the highest recall and the Normal class the highest support, so benign traffic is handled reliably. SQLi has the lowest recall, confirming that the Normal-SQLi boundary is the difficult region: benign inputs frequently contain SQL-like terms, quotes and identifiers without being malicious, and after deduplication the easy repeated templates that previously inflated this class are gone. SQLi mitigation therefore requires careful thresholding to balance unsafe false negatives against excessive false positives.

With the temperature, thresholds and margin frozen, the selected configuration is evaluated on the held-out clean test split of 27,109 requests. It reaches macro precision 0.9867, macro recall 0.9857, macro-F1 0.9862 and accuracy 0.9851, closely matching val_report and confirming that the operating point transfers to unseen traffic.

These figures are lower than those reported in the previous

version because of the corpus, not the model. Retraining the identical architecture on the deduplicated corpus moves clean-test macro-F1 from 0.9889 ± 0.0006 to 0.9858 ± 0.0006 over three seeds, a drop of 0.0031. The previously reported performance was therefore only mildly inflated by cross-split duplication. Still, the deduplicated figure is the one that generalises, and it is the figure reported throughout this section.

The clean-test confusion matrix (Table 11) locates the residual errors. The model maps 250 SQLi payloads to Normal and 95 Normal requests to SQLi, while XSS is almost perfectly separated. The Normal-SQLi boundary therefore accounts for the overwhelming majority of errors, which is what limits SQLi recall while XSS recall stays near unity.

Table 12 translates these predictions into the deployed actions. Under the blocking thresholds alone, the split resolves into 12,745 Allow, 14,364 Block and 0 Review, so the Review action never fires and 245 SQLi and 10 XSS payloads are allowed. This is the concrete cost of an inert third action, and it motivates the review margin introduced in Section 3.5: at $\tau_{\text{review}} = 0.30$ the Review rate becomes 0.240%, 6 of the allowed SQLi payloads are withheld for inspection and 59 benign requests are diverted; at $\tau_{\text{review}} = 0.20$ the figures are 1.808%, 43 and 444. Neither band is free, and the choice is an explicit operational trade-off rather than a modelling result.

Because the framework is positioned for JSON-oriented API traffic, Table 13 reports performance on the JSON-like slice of the clean test split. On the 3,917 JSON-like requests (14.4% of the split), macro-F1 is 0.9852, against 0.9863 on the non-JSON remainder, so detection does not degrade on JSON-structured inputs. Precision and recall are macro-averaged over the three classes.

Table 10. Per-class performance on val_report

Class	Precision	Recall	F1-score	Support
Normal	0.9787	0.9897	0.9842	6,319
SQL Injection (SQLi)	0.9894	0.9680	0.9786	4,253
XSS	0.9913	0.9983	0.9948	2,982

Note: Cross-Site Scripting (XSS); SQL Injection (SQLi)

Table 11. Clean-test confusion matrix

True \ Predicted	Normal	SQLi	XSS
Normal	12,502	95	42
SQL Injection (SQLi)	250	8,253	4
XSS	10	3	5,950

Note: Cross-Site Scripting (XSS); SQL Injection (SQLi)

Table 12. Allow/Block/Review distribution by true class

True Class	Allow	Block SQLi	Block XSS	Review	Review, Margin
Normal	12,490	107	42	0	59
SQLi	245	8,258	4	0	6
XSS	10	3	5,950	0	0

Note: Cross-Site Scripting (XSS); SQL Injection (SQLi)

Table 13. Clean-test performance on the JavaScript Object Notation (JSON)-like slice

Slice	Precision	Recall	Macro-F1	Accuracy
JSON-like	0.9845	0.9859	0.9852	0.9832
Non-JSON	0.9871	0.9857	0.9863	0.9854

5.4 Adversarial robustness and per-mutator ablation

The selected configuration is then attacked directly. The adversary of Section 4.5 is re-optimized against the deployed pipeline with its temperature and thresholds frozen, using 2,000 malicious seed payloads drawn from the deduplicated test split. A payload counts as evaded if any of the nine campaigns drives it to Allow. Under this protocol, the selected configuration allows 80 of 2,000 seed payloads, an evasion rate of 4.00% with an exact 95% Clopper-Pearson upper bound of 4.95%. Under the weaker not-Blocked criterion, the rate is 4.10%. SQLi remains the more malleable class under mutation (Table 14). This replaces the zero-evasion result reported in the previous version (Table 15), which was obtained by re-scoring a frozen holdout generated against a weaker earlier model and is not evidence about an attacker who targets the deployed system.

One control guards against the result being an artifact of the seed pool: 92.4% of the 2,000 seeds contain attack syntax that can be verified independently of the corpus labels after repeated URL-decoding and HTML unescaping, and restricting the measurement to that subset gives an evasion rate of 4.17%, statistically indistinguishable from the pooled figure. The result therefore does not rest on the noisy records described in Section 4.1.

Each configuration is attacked through its own operating point, fitted on val_fit by the same protocol, so no configuration is handicapped by another's thresholds. The comparison isolates where the robustness comes from. Without JSON-aware canonicalization, the same architecture is evaded on 94.10% of seeds before fine-tuning and 80.30% after; with canonicalization the figures are 5.30% and 4.00%. Canonicalization removes roughly nineteen twentieths of the attack surface and fine-tuning removes about a quarter of what remains, so canonicalization is the dominant contributor and few-shot fine-tuning is a secondary refinement. Fine-tuning also helps far more when canonicalization is absent, which is consistent with it learning to absorb encoding variance that canonicalization otherwise removes outright.

Table 14. Adaptive evasion rate by configuration

Configuration	Rate	95% UB	SQLi	XSS
canon_off, pre-FSAT	94.10%	95.09%	89.97%	100.00%
canon_off + FSAT	80.30%	82.02%	67.18%	99.03%
canon_on, pre-FSAT	5.30%	6.37%	8.16%	1.21%
Selected	4.00%	4.95%	5.95%	1.21%

Note: Cross-Site Scripting (XSS); SQL Injection (SQLi)

Table 15. The selected configuration under three evaluation protocols

Protocol	Selected Configuration	Note
Adaptive (this work)	4.00% of 2,000 seed payloads allowed	Attacker re-optimized against the deployed pipeline
Transfer, corrected generator	6.40% of 9,923 mutated rows allowed	Holdout generated against the pre-FSAT canon_off model; 987 distinct originals
Transfer, as reported previously	0.00%	Same protocol, but the holdout contained mutations of 8 distinct originals

The decision margins show that the successful evasions are a thin tail rather than a general weakness. Taking, for each seed payload, the highest P(Normal) reached anywhere in the campaign, the median is below 0.0001 and the 95th percentile is 0.3379, both far below the 0.40 Allow threshold, while the maximum reaches 0.9994. The distribution is therefore sharply bimodal: the attacker cannot move most payloads at all, but on the small minority it does defeat, it drives them well past the threshold rather than just over it (Figure 4). The aggregate rate alone conceals both halves of that behaviour, which is why the margin distribution is reported beside it.

The per-mutator ablation explains why chained search is needed at all. Each operator is applied singly to 2,000 malicious test payloads and compared against an unmutated control, so the control measures the model's plain false-negative rate on the same payloads (Table 16).

For the selected canonicalization-enabled configuration, the control allows 2.10% of the payloads, and no single mutator raises this materially: the strongest, SQL-comment insertion, reaches 2.35%, and several operators leave the rate unchanged. Single-operator mutation is therefore ineffective once canonicalization is applied, and the 4% reached in Section 5.4 is produced by chained mutation rather than by any individual transformation.

Without canonicalization, the picture is different: the same control allows 1.95% while random casing alone reaches 5.80% and double URL-encoding 2.55%. Representation-level operators are exactly what canonicalization neutralises, which is consistent with the configuration comparison in Table 14.

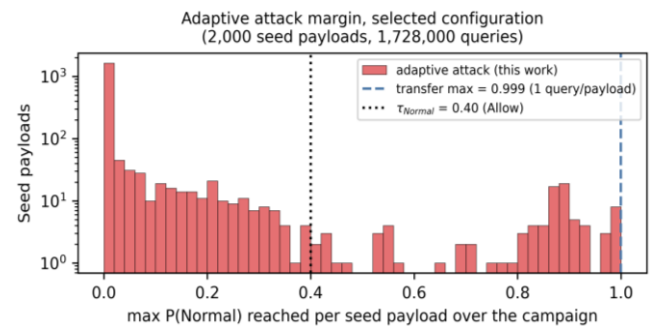


Figure 4. Distribution over the 2,000 seed payloads of the highest calibrated P(Normal) the adaptive attacker reached for each payload

Note: The dashed line is the 0.40 Allow threshold.

Table 16. Per-mutator ablation with an unmutated control

Mutator (Applied Singly)	Canon_on + FSAT, Allowed	Canon_off + FSAT, Allowed
none (control)	2.10%	1.95%
random case	2.10%	5.80%
double url encode	2.10%	2.55%
html encode	2.10%	2.50%
url encode	2.10%	0.75%
insert sql comments	2.35%	0.10%
unicode homoglyphs	2.10%	0.10%
JavaScript Object Notation (JSON) wrap	2.10%	2.05%

5.5 Latency and deployment feasibility

Latency is measured on the deployed path (canonicalization, inference, temperature scaling and the

threshold decision) at batch size 1, since an API gateway scores one request at a time. Measurements are taken in-process rather than through an HTTP wrapper, so they exclude network and serialisation overhead and represent the model-side cost only.

At 0.5 KB the selected configuration reaches $P50 = 4.0$ ms and $P95 = 6.8$ ms on GPU, rising to $P50 = 9.4$ ms at 8 KB, with CPU figures of 6.5 ms and 12.9 ms at the same points (Table 17). Canonicalization takes 6.2 ms on average at 8 KB, so it rather than inference dominates at large payload sizes, and batched throughput reaches 1,426 requests per second. Both are well inside the budget of an inline gateway component.

Table 17. End-to-end latency by payload size

Payload	GPU P50	GPU P95	CPU P50	CPU P95	Canon.
0.5 KB	4.0	6.8	6.5	9.7	0.85
2.0 KB	4.9	7.3	7.9	13.5	2.07
8.0 KB	9.4	11.8	12.9	19.3	6.23

5.6 Tradeoffs, security interpretation and discussion

The clean-best and the robust-best configurations are not the same model, and the adaptive evaluation shows the size of the gap. Disabling canonicalization gains roughly two thousandths of clean macro-F1 and pays for it with an evasion rate of 80.30% against 4.00%, a factor of twenty. A model chosen on clean accuracy alone would therefore have been the wrong deployment choice by a wide margin, which is why the selected configuration is fixed on clean performance, adversarial behaviour, calibration and latency together rather than on macro-F1 alone.

The Review decision is the part of the framework that separates a classifier from a mitigation policy, but it only functions if it is reachable. Section 5.3 showed that under blocking thresholds alone it is not: it fires on 0 of 27,109 clean test requests, and every uncertain request is therefore silently allowed.

The review margin repairs this. At $\tau_{\text{review}} = 0.30$, the deployed system routes 0.240% of clean traffic to Review, withholding 6 of the 245 SQLi payloads that the blocking thresholds would have allowed, at the cost of 59 benign requests. Widening the margin to 0.20 raises the recovery to 43 payloads for 444 benign diversions. The margin is thus a tunable security dial with an explicit and measurable price, rather than a mechanism that is nominally present but never active.

The results support four conclusions. First, class-specific thresholding with an explicit review margin converts a detector into an auditable mitigation policy, and the margin is necessary: without it the Review action is unreachable at the selected operating point.

Second, calibration belongs to the security pipeline rather than to post-processing, and it must be fitted and reported on disjoint data. Fitting the temperature and thresholds on one half of the validation split and reporting on the other costs 0.0023 macro-F1. That is a small quantity, but it is the part of the previously reported figure that was not independent evidence.

Third, the evaluation protocol dominates both conclusions. The same system reports no evasions under the transfer holdout used previously, 6.40% once that holdout is regenerated without the sampling defect described in Section 4.5, and 4.00% under an attacker re-optimized against the deployed pipeline; and more than half of the test split shared a

payload template with training before deduplication, so any comparison drawn on such a split measures memorisation as much as generalisation.

Overall, the framework converts detector outputs into deployable security decisions, and the revised evaluation shows what that conversion is and is not worth: a well-calibrated, auditable policy with a measurable residual attack surface, rather than an impermeable defence.

6. CONCLUSIONS

This paper presented a risk-calibrated mitigation framework that converts a deep-learning detector's output into Allow, Block and Review actions for JSON-oriented API traffic. A BiLSTM-Attention detector with JSON-aware canonicalization and few-shot adversarial fine-tuning supplies the score; temperature scaling ($T = 0.4889$), the threshold vector 0.40 / 0.40 / 0.60 and a review margin of 0.30 supply the policy. All four constants are fitted on one half of the validation split and frozen before any reported figure is computed. On a corpus deduplicated at exact, near-duplicate and template level, the system reaches macro-F1 0.9862 and accuracy 0.9851 on a test split that shares no payload template with training, and under an adversary re-optimized against the deployed pipeline it allows 4.00% of malicious seed payloads (95% upper bound 4.95%) against 80.30% for the same architecture without canonicalization. Calibrated thresholding is therefore not a statistical afterthought: it defines when a request is safe enough to allow, malicious enough to block, or uncertain enough to review. The paper also shows that two evaluation choices, namely how the corpus is split and whether the attacker targets the deployed system, move the headline numbers more than any modelling choice examined here.

Five limitations qualify these results. First, the supervised labels are noisy: after decoding, 6.42% of SQLi-labelled and 4.61% of XSS-labelled payloads contain no recognisable attack syntax, so absolute performance is bounded by the source annotations, which is why the adversarial result is also reported on the syntax-verified subset. Second, the robustness claim is bounded by the mutation library, the chain length and the JSON-validity constraint, and the attacker is hard-label; a score-based or gradient-transfer adversary may do better. Third, the evasion rate is measured on 2,000 seed payloads, so the confidence bound rather than the point estimate is the quantity to carry forward. Fourth, the recurrent detector's backward pass consumes padding, so a request's score depends on the batch it is served in; all results here pin the sequence length to a constant so that batched and single-request scoring agree, and a deployment that batches variable-length requests without this precaution will not reproduce them. Fifth, the corpus remains a JSON-oriented subset rather than a fully JSON-native capture of production traffic. Collecting real API traffic, extending the attacker to score-based feedback and re-examining the Normal-SQLi boundary with cleaner labels are the natural next steps.

REFERENCES

- [1] OWASP Foundation. (2023). OWASP API security top 10—2023. Open Worldwide Application Security Project. <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>.

- [2] Fadlalla, F.F., Elshoush, H.T. (2023). Input validation vulnerabilities in web applications: Systematic review, classification, and analysis of the current state-of-the-art. *IEEE Access*, 11: 40128-40152. <https://doi.org/10.1109/ACCESS.2023.3266385>
- [3] Kaur, J., Garg, U., Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: A review. *Artificial Intelligence Review*, 56: 12725-12769. <https://doi.org/10.1007/s10462-023-10433-3>
- [4] Tadhani, J.R., Vekariya, V., Sorathiya, V., Alshathri, S., El-Shafai, W. (2024). Securing web applications against XSS and SQLi attacks using a novel deep learning approach. *Scientific Reports*, 14: 1803. <https://doi.org/10.1038/s41598-023-48845-4>
- [5] Bakir, R. (2025). UniEmbed: A novel approach to detect XSS and SQL injection attacks leveraging multiple feature fusion with machine learning techniques. *Arabian Journal for Science and Engineering*, 50: 15591-15604. <https://doi.org/10.1007/s13369-024-09916-4>
- [6] Wang, Q.H., Yang, H., Wu, G.H., et al. (2022). Black-box adversarial attacks on XSS attack detection model. *Computers & Security*, 113: 102554. <https://doi.org/10.1016/j.cose.2021.102554>
- [7] Guan, Y.T., He, J.J., Li, T., Zhao, H., Ma, B.Q. (2023). SSQLi: A black-box adversarial attack method for SQL injection based on reinforcement learning. *Future Internet*, 15(4): 133. <https://doi.org/10.3390/fi15040133>
- [8] Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, Sydney, NSW, Australia, pp. 1321-1330. <https://dl.acm.org/doi/10.5555/3305381.3305518>
- [9] Sokolova, M., Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4): 427-437. <https://doi.org/10.1016/j.ipm.2009.03.002>
- [10] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8): 861-874. <https://doi.org/10.1016/j.patrec.2005.10.010>
- [11] Tramèr, F., Carlini, N., Brendel, W., Madry, A. (2020). On adaptive attacks to adversarial example defenses. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Vancouver, BC, Canada, pp. 1633-1645. <https://dl.acm.org/doi/abs/10.5555/3495724.3495862>
- [12] Brendel, W., Rauber, J., Bethge, M. (2018). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *6th International Conference on Learning Representations (ICLR 2018)*, pp. 1-12. <https://openreview.net/pdf/da19d08fdc4c341975c0042678b941d40ccbea3d.pdf>
- [13] Macas, M., Wu, C.M., Fuertes, W. (2024). Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems. *Expert Systems with Applications*, 238: 122223. <https://doi.org/10.1016/j.eswa.2023.122223>
- [14] Rosenberg, I., Shabtai, A., Elovici, Y., Rokach, L. (2021). Adversarial machine learning attacks and defense methods in the cybersecurity domain. *ACM Computing Surveys*, 54(5): 1-36. <https://doi.org/10.1145/3453158>
- [15] Chen, L., Tang, C., He, J.J., Zhao, H., Lan, X.L., Li, T. (2022). XSS adversarial example attacks based on deep reinforcement learning. *Computers & Security*, 120: 102831. <https://doi.org/10.1016/j.cose.2022.102831>
- [16] Isife, O.F., Okokpujie, K., Okokpujie, I.P., Subair, R.E., Vincent, A.A., Awomoyi, M.E. (2023). Development of a malicious network traffic intrusion detection system using deep learning. *International Journal of Safety and Security Engineering*, 13(4): 587-595. <https://doi.org/10.18280/ijss.130401>
- [17] Sridharan, S., Patil, S., Shobha, T., Pai, P. (2025). Hybrid machine learning-based intrusion detection for zero-day attack prevention in digital education networks. *International Journal of Safety and Security Engineering*, 15(8): 1703-1713. <https://doi.org/10.18280/ijss.150815>
- [18] Pallikonda, A.K., Bandarapalli, V.K., Vipparla, A. (2025). Real-time anomaly detection in IoT networks using a hybrid deep learning model. *Acadlore Transactions on AI and Machine Learning*, 4(4): 235-246. <https://doi.org/10.56578/ataiml040401>
- [19] Ibrahim, Z.B., Ghanim, M.F. (2024). Leveraging artificial intelligence for blackhole attack detection in MANETs: A comparative study. *Information Dynamics and Applications*, 3(4): 245-257. <https://doi.org/10.56578/ida030404>
- [20] OWASP Foundation. (2025). OWASP ModSecurity Core Rule Set. Open Worldwide Application Security Project. <https://owasp.org/www-project-modsecurity-core-rule-set/>
- [21] Coscia, A., Dentamaro, V., Galantucci, S., Maci, A., Pirlo, G. (2024). PROGESI: A PROxy grammar to enhance web application firewall for SQL injection prevention. *IEEE Access*, 12: 107689-107713. <https://doi.org/10.1109/ACCESS.2024.3438092>
- [22] Floris, G., Scano, C., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L. (2025). ModSec-AdvLearn: Countering adversarial SQL injections with robust machine learning. *IEEE Transactions on Information Forensics and Security*, 20: 6693-6705. <https://doi.org/10.1109/TIFS.2025.3583234>
- [23] Platt, J.C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, MIT Press, pp. 61-74. <https://home.cs.colorado.edu/~mozer/Teaching/syllabi/6622/papers/Platt1999.pdf>
- [24] Zadrozny, B., Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, pp. 694-699. <https://doi.org/10.1145/775047.775151>
- [25] Barz, B., Denzler, J. (2020). Do we train on test data? Purging CIFAR of near-duplicates. *Journal of Imaging*, 6(6): 41. <https://doi.org/10.3390/jimaging6060041>
- [26] Lee, K., Ippolito, D., Nystrom, A., et al. (2022). Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 8424-8445. <https://doi.org/10.18653/v1/2022.acl-long.577>
- [27] Clopper, C.J., Pearson, E.S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4): 404-413. <https://doi.org/10.1093/biomet/26.4.404>