Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Fine-Tuned CodeBERT for Web Threat Classification using Agentic Reasoning

Oybek Zokirjonov

zokirjonovoybek4@gmail.com

Westminster International University in Tashkent, Tashkent, Uzbekistan

Zahid Hussain Qaisar

Westminster International University in Tashkent, Tashkent, Uzbekistan

Research Article

Keywords: SQL injection; cross-site scripting; CodeBERT; transformer fine-tuning; web application firewall; agentic reasoning; bibliometric analysis; evaluation uncertainty

Posted Date: November 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11660479/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Fine-Tuned CodeBERT for Web Threat Classification using Agentic Reasoning

Oybek Zokirjonov
Westminster International University in Tashkent, Tashkent, Uzbekistan
zokirjonovoybek4@gmail.com

Zahid Hussain Qaisar
Westminster International University in Tashkent, Tashkent, Uzbekistan
zqaisar@wiut.uz

Abstract

SQL injection (SQLi) and cross-site scripting (XSS) are among the most serious vulnerabilities of web applications. Existing detection approaches often need considerable time and memory to reach high accuracy, and most treat the two attacks separately. We fine-tune CodeBERT, a transformer pre-trained on source code, to detect both attacks from raw payload text and compare it with BERT-base, RoBERTa-base and DistilBERT under one identical configuration. CodeBERT misclassified 7 of 6,259 test samples (error rate 0.11%), against 0.13% to 0.19% for the other models, and a shorter input of 64 tokens cut training time by 42% and latency by 46%. One model therefore covers both attacks at low cost, which suits lightweight web application firewalls; this binary (benign vs. malicious) study is the first step towards attack-type classification.

1 Introduction

This article studies two of the most common web injection attacks: SQL injection (SQLi), which alters the queries that an application sends to its database, and cross-site scripting (XSS), which injects client-side script into pages that other users view. It asks how well a transformer pre-trained on code can detect them from raw payload text, and what that detection costs. Both attacks occur in several subtypes; for example, SQLi appears as tautology-, union-, error- and time-based attacks. This article addresses the first step as a binary classification problem: each payload is labelled 0 (benign) or 1 (malicious), because the datasets used here carry no subtype labels. Separating the subtypes is the next step.

1.1 Background and motivation

Injection flaws, which include SQL injection (SQLi) and cross-site scripting (XSS), remain in the OWASP Top 10 list of critical web application risks [1]. SQLi alters the queries that an application sends to its database and occurs in several forms, such as tautology-, union-, error- and time-based attacks [2]; XSS injects client-side script into pages that other users view. Web application firewalls (WAFs) are the standard first line of defence, but signature-based rules can be bypassed with semantically equivalent payloads, for example by changing letter case, inserting comments or replacing whitespace [3]. Machine-learning WAFs aim to learn the underlying patterns [4]. Transformer encoders [5] that are pre-trained and then fine-tuned are strong baselines for text classification. BERT [6], RoBERTa [7] and DistilBERT [8] are general-purpose models; CodeBERT [9] has the RoBERTa architecture but is pre-trained on paired natural-language and programming-language data. SQLi and XSS payloads mix SQL, HTML and JavaScript tokens, which makes a code-domain model a natural candidate; continued pre-training on in-domain text is known to help downstream tasks [10]. Many published detectors report accuracy close to 100% on in-distribution test sets (Table 1); at that level, small differences between models can be indistinguishable from sampling noise, and accuracy alone says little about error type, robustness or cost. Published scores come from different datasets and settings, so they cannot be compared directly. We therefore compare four fine-tuned models under the same data, settings and metrics, to show whether the choice of model matters and what each choice costs.

1.2 Related work

Classical and anomaly-based detection. Early learning-based detectors modelled normal requests or SQL queries and flagged deviations [11, 12]. A survey of signature-based and machine-learning WAFs concluded that the effectiveness of ML-based WAFs against current attack patterns remains an open question [4]; a review of deep learning for web-attack detection [13] catalogues models, datasets and open challenges. **Deep learning for XSS and web attacks.** DeepXSS maps XSS payloads to word2vec features and classifies them with an LSTM (precision 99.5%, recall 97.9%) [14]. A CNN with a modified residual block and a network-in-network design reached 99.23% accuracy [15]. Maurel et al. compared code representations for statically identifying XSS [16], and hybrid RNN–CNN [17] and federated Transformer–BiLSTM [18] models have been proposed. For multilabel web traffic, Chindrus and Caruntu compared seven models on ECML/PKDD 2007 [19]. **A closely related detector for both attacks.** Tadhani et al. propose a hybrid CNN–LSTM that detects SQLi and XSS together [20]. Inputs are decoded (URL, HTML-entity, hexadecimal and Unicode encodings), generalised, tokenised and standardised, embedded with word2vec and passed through three convolutional layers and an LSTM. They evaluate on HTTP CSIC 2010, a Kaggle SQLi–XSS payload dataset and a testbed of 11,350 HTTP requests (4,537 normal), with accuracies of 99.77%, 99.84% and 99.23%. Their review of related work lists detectors that cover only SQLi or only XSS. A hybrid pipeline that passes BERT embeddings to a graph neural network targets SQLi detection [21]. **Adversarial evasion and evaluation practice.** Small perturbations can fool neural networks [22, 23], and adversarial machine learning has been surveyed extensively [24]; this includes BERT-based text classifiers [25]; WAF-A-MoLE applies semantics-preserving mutations (case swapping, whitespace substitution, comment injection) and bypassed all ML-based WAFs that its authors evaluated [3]. Duplicated samples inflate the apparent performance of models of code

[26], and fine-tuned BERT models can differ between random seeds [27].

Table 1. *Representative related work (values as reported by the cited authors, in %; datasets and protocols differ).*

Ref.	Attack	Method	Reported result
[14]	XSS	word2vec + LSTM	Prec. 99.5, recall 97.9
[15]	XSS	Modified-ResNet-block CNN	Acc. 99.23
[18]	XSS	Federated Transformer + BiLSTM	Acc. 98.6, F1 98.5
[20]	SQLi + XSS	Decoding + word2vec + CNN–LSTM	Acc. 99.84 (payloads), 99.77 (CSIC 2010), 99.23 (testbed)
[19]	Web attacks	Seven classical and deep models	ECML/PKDD 2007 comparison
[21]	SQLi	BERT embeddings + GNN	Lower FPR and FNR reported
This work	SQLi + XSS	Fine-tuned CodeBERT vs. three encoders	Acc. 99.89 (7 errors in 6,259)

Comparison with the closest prior work. Table 2 sets our results beside those of Tadhani et al. [20]. The comparison is indicative only: their data include HTTP requests and a self-collected testbed, their test-set sizes and repetitions are not reported, and they use word2vec features with a CNN–LSTM. On payload data the two approaches are at the same level: their result (accuracy 99.84%, F1 99.82%) lies within the sampling uncertainty of ours (accuracy 99.89%, 95% Wilson interval 99.77–99.95%, Section 3.4), so the 0.05-point gap is not evidence of a difference. Our models take raw strings and need no decoding or standardisation rules, and we additionally report confidence intervals, error counts by type, repeated runs, an input-length ablation and cost.

Table 2. *Comparison with Tadhani et al. [20] (their values from their Table 6 and text; metrics in %).*

Study / model	Dataset	Acc.	Rec.	F1
Tadhani, CNN–LSTM	Payload (Kaggle)	99.84	99.88	99.82
Tadhani, CNN–LSTM	HTTP CSIC 2010	99.77	99.21	99.48
Tadhani, CNN–LSTM	Own testbed	99.23	99.15	98.97
This work, CodeBERT	Payload (n=6,259) $n = 6{,}259$	99.89	99.86	99.87
This work, RoBERTa-base	Payload (n=6,259) $n = 6{,}259$	99.87	99.93	99.86
This work, DistilBERT	Payload (n=6,259) $n = 6{,}259$	99.86	99.82	99.84
This work, BERT-base	Payload (n=6,259) $n = 6{,}259$	99.81	99.61	99.79

1.3 Research gap and contributions

Within the studies reviewed here, four gaps stand out: (i) SQLi and XSS are usually handled by separate detectors, and when handled together it is with CNN–LSTM networks rather than pre-trained transformers; (ii) we did not find a study that compares several transformer encoders under one identical protocol on SQLi and XSS data and reports uncertainty, error type and cost together; (iii) the closest prior work [20] relies on hand-designed decoding, reports point estimates without confidence intervals and does not examine robustness to obfuscation; (iv) robustness is often discussed as motivation but seldom tested next to in-distribution accuracy. Our contributions are:

1. a controlled comparison of fine-tuned CodeBERT, BERT-base, RoBERTa-base and DistilBERT on a merged, de-duplicated corpus from two public Kaggle datasets [28, 29] plus 25 hand-written benign hard negatives;
2. a ten-metric evaluation with explicit formulas and an uncertainty analysis (Wilson, Clopper–Pearson and bootstrap intervals, best-case McNemar bounds);
3. an input-length ablation and a time-factor analysis of training time and inference latency;
4. a small obfuscated-payload stress test with its limitations stated.

2 Methodology

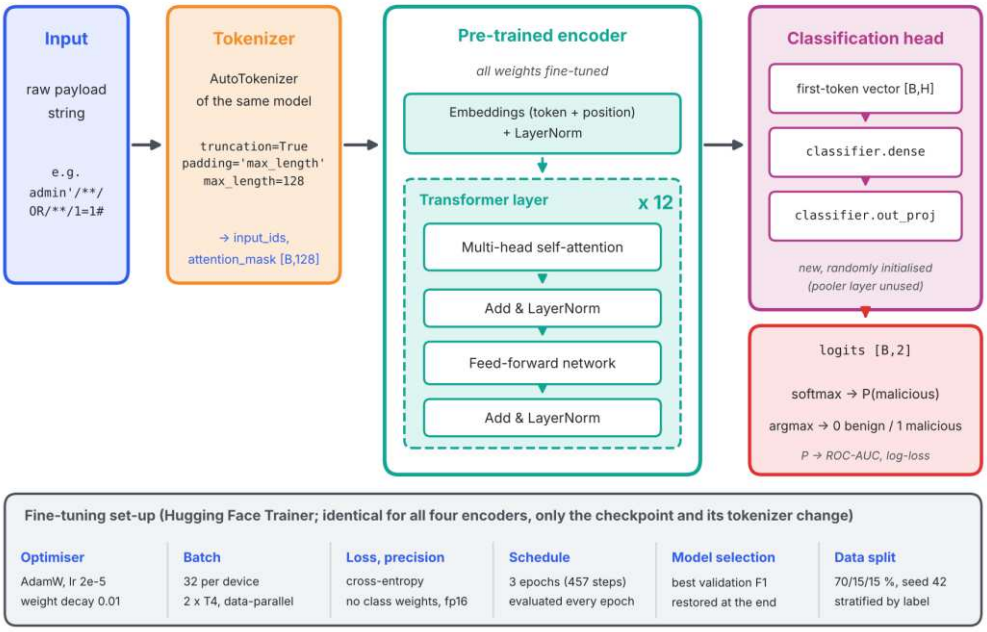


Figure 1. Architecture of the fine-tuned classifier: tokenizer, pre-trained encoder (12 Transformer layers), newly initialised classification head and fine-tuning set-up.

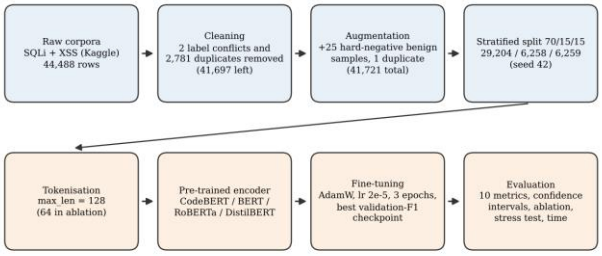


Figure 2. *Experimental workflow.*

2.1 Datasets and preprocessing

Two public Kaggle datasets were used: the SQL Injection Dataset [28] (columns Query and Label; https://www.kaggle.com/datasets/sajid576/sql-injection-dataset) and the cross-site scripting dataset for deep learning [29] (columns Sentence and Label; https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning). In both, label 1 marks a malicious payload and 0 benign text. The two sources were merged, so the task is binary malicious-payload detection; SQLi and XSS were not evaluated separately. Rows with missing text or label and empty texts were dropped (44,488 rows remain) and leading and trailing whitespace was stripped. No URL decoding, HTML-entity decoding or case folding was applied in the final pipeline. Two texts that occurred with both labels were removed, followed by 2,781 exact duplicate texts, leaving 41,697 unique samples (55.29% benign, 44.71% malicious). Twenty-five hand-written benign hard negatives were then added: harmless strings that look like attacks, such as SQL joins and aggregations, comments and apostrophes in prose, HTML meta and link tags, regular expressions, JSON and log lines, and mathematical conditions. They were placed first so that they survive de-duplication, which removed one further duplicate. The final dataset has 41,721 samples (23,080 benign and 18,641 malicious). The shuffled data (seed 42) were split into training, validation and test sets of 29,204, 6,258 and 6,259 samples (70/15/15%), stratified by label; the test set holds 3,462 benign and 2,797 malicious samples. De-duplication precedes the split, and an explicit assertion confirmed that no text occurs in two subsets (checked on the earlier 41,697-sample split). Near-duplicate variants were not removed. Samples are short (before augmentation: mean 83.2 characters, SD 137.5, median 52, maximum 5,909; nine texts exceed 256 words), which motivated a maximum input length of 128 tokens.

2.2 Models and fine-tuning setup

Four pre-trained encoders were fine-tuned: CodeBERT (`microsoft/codebert-base`) [9], BERT-base (`bert-base-uncased`) [6], RoBERTa-base (`roberta-base`) [7] and DistilBERT (`distilbert-base-uncased`) [8], using Hugging Face Transformers [30] and PyTorch [31] with a newly initialised two-class head. CodeBERT and RoBERTa-base share the architecture and use a byte-level BPE tokeniser [32], whereas BERT-base and DistilBERT use an uncased WordPiece tokeniser [33]; the CodeBERT–RoBERTa comparison therefore isolates code-domain pre-training. All models used the same splits and configuration: the Trainer with AdamW [34] (a variant of Adam [35]), peak learning rate 2×10^{-5} as in common BERT fine-tuning practice [6, 36], weight decay 0.01, per-device training batch size 32 on two NVIDIA Tesla T4 GPUs in data-parallel mode (457 optimisation steps per epoch), maximum length 128 with truncation and padding, cross-entropy loss without class weights, fp16, and three epochs for the comparison. Hyperparameters were not tuned per model. Validation metrics were computed after every epoch and the checkpoint with the highest validation F1 was restored. Two additional standalone CodeBERT runs were made: Run A (4 epochs, 100 warm-up steps, evaluation batch size 64) and Run B (3 epochs); Run C is the

CodeBERT row of the four-model comparison. The ablation used 2 epochs. Figure 1 shows the architecture of the fine-tuned classifier and the fine-tuning set-up; Figure 3 summarises the dataset.

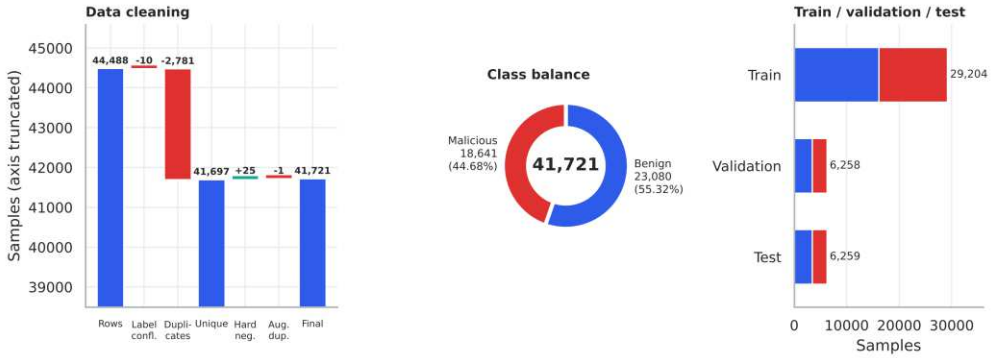


Figure 3. Dataset composition: cleaning steps, class balance and train/validation/test split.

2.3 Evaluation metrics

The malicious class is positive. With TP, TN, FP and FN the confusion-matrix counts, ten metrics were computed with scikit-learn [37]: accuracy, precision, recall, specificity, FPR, FNR, F1, MCC, ROC-AUC and log-loss. A false negative is an attack that gets through and a false positive is legitimate traffic that gets blocked, so FNR and FPR are reported separately. Equation 1 defines accuracy, precision, recall, specificity, FPR, FNR and F1 from these counts. Accuracy, precision, recall, specificity, F1, FPR and FNR are given in percent (the ratios below multiplied by 100); MCC, ROC-AUC and log-loss are unitless coefficients.

$$\begin{aligned} \text{Accuracy} &= \frac{TP+TN}{TP+TN+FP+FN}, & \text{F1} &= \frac{2PR}{P+R}, \\ P &= \frac{TP}{TP+FP}, & R &= \frac{TP}{TP+FN}, \\ \text{Spec.} &= \frac{TN}{TN+FP}, & \text{FPR} &= 1-\text{Spec.}, \\ \text{FNR} &= 1-R, \end{aligned} \tag{1}$$

$$\text{MCC} = \frac{TP\cdot TN - FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, \tag{2}$$

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1-y_i)\log(1-p_i)], \qquad p_1 = \frac{e^{z_1}}{e^{z_0}+e^{z_1}}. \tag{3}$$

The Matthews correlation coefficient (MCC) [38] uses all four cells (Equation 2) and is more informative than accuracy or F1 for uneven classes [39]. Equation 3 gives the cross-entropy loss L used for training and for log-loss, where N is the number of samples, y_i the true label and p_i the predicted malicious probability obtained from the softmax of the two logits z_0 and z_1. ROC-AUC [40] measures ranking quality independent of the threshold. The amortised latency per sample is T_{eval}/N, where T_{eval} is the wall-clock time of the evaluation pass.

2.4 Statistical analysis

Per-sample predictions were not stored in the training runs, but the confusion-matrix cells can be recovered exactly: the test class sizes are known (3,462 and 2,797), and for each run only one integer pair (FP, FN) reproduces all eight logged threshold metrics at their reported precision. All intervals below are computed from these cells. Accuracy carries the Wilson score interval [41] (Equation 4), which gives a 95% range for the true accuracy from the observed accuracy $\hat{p}$, the test size n and $z = 1.96$,

$$\text{CI} = \frac{\hat{p} + z^2/(2n) \pm z\sqrt{\hat{p}(1-\hat{p})/n + z^2/(4n^2)}}{1+z^2/n}, \quad z = 1.96,\ n = 6{,}259; \tag{4}$$

FPR and FNR carry exact Clopper–Pearson intervals [42]; F1 and MCC carry percentile bootstrap intervals [43] from 20,000 resamples. Paired comparison would normally use McNemar's test [44], which needs per-sample predictions. We therefore report the smallest attainable two-sided exact p-value: if two models make e_1 and e_2 errors, the discordant counts satisfy $b - c = e_1 - e_2$ and the p-value is smallest when $c = 0$, giving Equation 5,

$$p_{\min} = 2 \cdot 0.5^{|e_1 - e_2|}. \tag{5}$$

This bound is the best case for the better model, so it only shows whether significance is possible. A pair with $p_{\min} > 0.05$ cannot be significant however the errors overlap. Run-to-run variability was measured from three CodeBERT runs.

2.5 Ablation, stress test and time factors

Input-length ablation. CodeBERT was fine-tuned with maximum lengths 64 and 128 for two epochs on the same splits, all else unchanged. **Stress test.** A hand-built set of 15 samples probes robustness: five obfuscated SQLi payloads (version-conditional comment, inline comments, stacked time-delay query, parenthesis trick, string concatenation), five WAF-bypass XSS payloads, and five benign but keyword-heavy samples. None was used in training. The set exists in two versions: version 1 was used in a single-model CodeBERT probe with per-sample logging, and version 2 (nine samples reworded) in the four-model comparison. Each model classified the samples one at a time with maximum length 128. **Time factors.** Training time is the wall-clock time of `Trainer.train`, including per-epoch validation and checkpoint saving. Inference time is the wall-clock time of `Trainer.predict` on the test set with GPU synchronisation, so the amortised latency is end-to-end throughput including tokenisation and padding. For the stress test, latency was measured per sample at batch size 1 (the four-model script times the forward pass and the logit transfer; the single-model script also times tokenisation and softmax); neither used a warm-up phase.

3 Results and Discussion

3.1 Training dynamics

Validation loss for CodeBERT reached its minimum after the second epoch in the 3-epoch run (0.0061) and the 4-epoch run (0.0068) and rose in later epochs (to 0.0097 and 0.0089), while training loss was already close to zero (0.0004 in epoch 3 of the 3-epoch run and 0.0037 in epoch 4 of the 4-epoch run). BERT-base and DistilBERT show the same pattern; RoBERTa-base improved slightly through epoch 3 (0.0038 to 0.0032). The highest validation F1 occurred at epoch 2 for BERT-base, RoBERTa-base (tied with epoch 3) and the 3-epoch CodeBERT run, and at epoch 3 for DistilBERT.

3.2 CodeBERT test performance and variability

Table 3 reports three CodeBERT runs. Accuracy ranged from 99.82% to 99.90% and F1 from 99.80% to 99.89%, i.e. 6 to 11 misclassified test samples (Run A: 5 false positives and 6 false negatives; Run B: 4 and 2; Run C: 3 and 4). ROC-AUC was 1.0000 (rounded) in all runs. The runs differ in design (training length and warm-up), so their spread reflects training length and random variation together.

Table 3. *CodeBERT on the test set ($n = 6{,}259$) in three runs (in %; MCC unitless).*

Metric	Run A	Run B	Run C	Mean ± SD
Accuracy	99.82	99.90	99.89	99.87 ± 0.04
Precision	99.82	99.86	99.89	99.86 ± 0.04
Recall	99.79	99.93	99.86	99.86 ± 0.07
Specificity	99.86	99.88	99.91	99.88 ± 0.03
F1	99.80	99.89	99.87	99.86 ± 0.05
FPR	0.14	0.12	0.09	0.12 ± 0.03
FNR	0.21	0.07	0.14	0.14 ± 0.07
MCC	0.9964	0.9981	0.9977	0.9974 ± 0.0009

3.3 Comparison with other transformer architectures

Under the identical protocol (Table 4, Figure 4), all four models scored above 99.8% accuracy. CodeBERT had the highest accuracy (99.89%), F1 (99.87%) and MCC (0.9977); RoBERTa-base followed (99.87%, 99.86%, 0.9974), then DistilBERT (99.86%, 99.84%, 0.9971) and BERT-base (99.81%, 99.79%, 0.9961). ROC-AUC was 1.0000 for all four. The error profiles differ more than the headline scores: BERT-base had the lowest FPR (0.029%) but the highest FNR (0.393%), RoBERTa-base the opposite pattern (FNR 0.072%, FPR 0.173%), and CodeBERT was the most balanced (FPR 0.090%, FNR 0.140%). These are single-run values (next subsection).

Table 4. Test-set metrics, cost and calibration of the four encoders (3 epochs, maximum length 128). Metrics in %; train time in seconds, latency in ms per sample.

Model	Acc.	Prec.	Rec.	Spec.	F1	FPR	FNR	MCC	Log-loss	Train (s)	Lat. (ms)
CodeBERT	99.89	99.89	99.86	99.91	99.87	0.087	0.143	0.9977	0.0071	1143.55	4.20
BERT-base	99.81	99.96	99.61	99.97	99.79	0.029	0.393	0.9961	0.0091	1148.17	4.59
RoBERTa-base	99.87	99.79	99.93	99.83	99.86	0.173	0.072	0.9974	0.0066	1165.83	4.28
DistilBERT	99.86	99.86	99.82	99.88	99.84	0.116	0.179	0.9971	0.0086	601.15	2.39

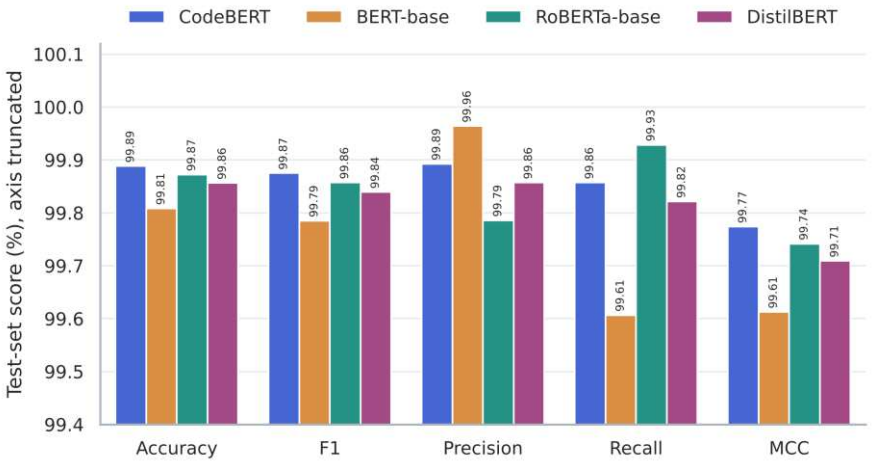


Figure 4. Test-set metrics of the four encoders (truncated axis).

3.4 Statistical analysis

Table 5 gives the recovered confusion matrices and intervals. The four architectures make between 7 and 12 errors on 6,259 test samples (0.11–0.19%): CodeBERT 7, RoBERTa-base 8, DistilBERT 9, BERT-base 12. The 95% Wilson intervals for accuracy overlap almost completely, as do the bootstrap intervals for F1 and MCC. CodeBERT's own three runs span 6 to 11 errors, a range of five errors that equals the range across the four architectures (7 to 12), so run-to-run variation is as large as the difference between models.

Table 5. Recovered test confusion matrices (positive = malicious; 3,462 benign and 2,797 malicious samples), accuracy with 95% Wilson intervals, F1 and MCC with 95% bootstrap intervals, and FPR/FNR with exact 95% Clopper–Pearson intervals (in %).

Model	TN	FP	FN	TP	Err.	Accuracy [95% CI]	F1 [95% CI]	MCC [95% CI]
CodeBERT (Run C)	3459	3	4	2793	7	99.89 [99.77, 99.95]	99.87 [99.77, 99.96]	0.9977 [0.9958, 0.9994]
BERT-base	3461	1	11	2786	12	99.81 [99.67, 99.89]	99.79 [99.66, 99.89]	0.9961 [0.9939, 0.9981]
RoBERTa-base	3456	6	2	2795	8	99.87 [99.75, 99.94]	99.86 [99.75, 99.95]	0.9974 [0.9955, 0.9990]
DistilBERT	3458	4	5	2792	9	99.86 [99.73, 99.92]	99.84 [99.72, 99.93]	0.9971 [0.9951, 0.9987]
CodeBERT, Run A	3457	5	6	2791	11	99.82 [99.69, 99.90]	99.80 [99.68, 99.91]	0.9964 [0.9942, 0.9984]
CodeBERT, Run B	3458	4	2	2795	6	99.90 [99.79, 99.96]	99.89 [99.80, 99.96]	0.9981 [0.9964, 0.9994]
			FPR and FNR (%), exact 95% Clopper–Pearson intervals					
CodeBERT (Run C)		FP 3/3,462: FPR 0.087 [0.018, 0.253]					FN 4/2,797: FNR 0.143 [0.039, 0.366]	
BERT-base		FP 1/3,462: FPR 0.029 [0.001, 0.161]					FN 11/2,797: FNR 0.393 [0.196, 0.703]	
RoBERTa-base		FP 6/3,462: FPR 0.173 [0.064, 0.377]					FN 2/2,797: FNR 0.072 [0.009, 0.258]	
DistilBERT		FP 4/3,462: FPR 0.116 [0.031, 0.296]					FN 5/2,797: FNR 0.179 [0.058, 0.417]	
CodeBERT, Run A		FP 5/3,462: FPR 0.144 [0.047, 0.337]					FN 6/2,797: FNR 0.215 [0.079, 0.466]	
CodeBERT, Run B		FP 4/3,462: FPR 0.116 [0.031, 0.296]					FN 2/2,797: FNR 0.072 [0.009, 0.258]	

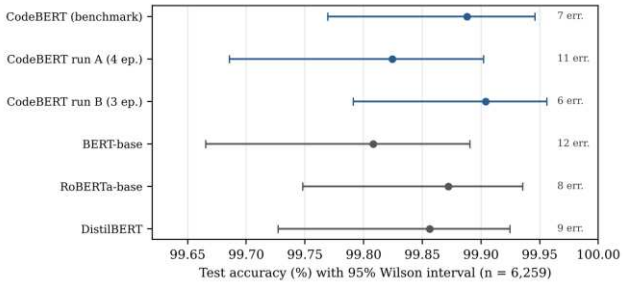


Figure 5. *Test accuracy (%) with 95% Wilson intervals; the number of test errors is printed next to each interval.*

For none of the six architecture pairs can the exact McNemar test reach $p < 0.05$ on total errors: the smallest attainable p-values are 0.0625 for CodeBERT versus BERT-base (five errors apart), 0.125 for BERT-base versus RoBERTa-base, and 0.0625 for CodeBERT's Run A versus Run B (11 versus 6 errors), so CodeBERT differs from itself as much as from the weakest competitor. The data therefore support comparable accuracy, not superiority. The error types differ more: BERT-base missed 11 of 2,797 malicious samples (FNR 0.39%, 95% interval 0.20–0.70%), against 4 for CodeBERT, 5 for DistilBERT and 2 for RoBERTa-base. Restricted to malicious samples, the best-case McNemar p-value for BERT-base versus RoBERTa-base is 0.0039, below a Bonferroni-corrected $0.05/6 \approx 0.0083$; this is the only contrast that could survive such a correction, and confirming it requires per-sample predictions. Figure 6 shows the confusion matrices of the four encoders.

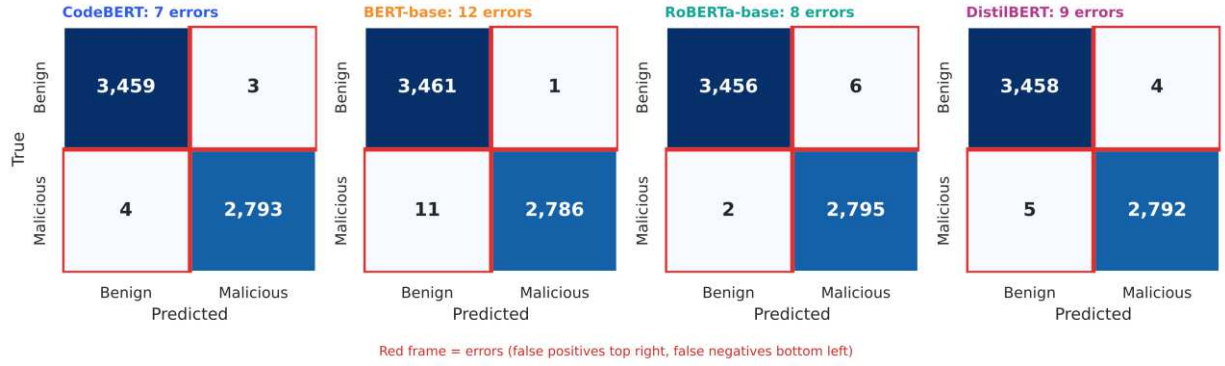


Figure 6. Test-set confusion matrices of the four encoders (n = 6,259).

3.5 Ablation: maximum input length

Reducing the maximum length from 128 to 64 tokens left accuracy essentially unchanged (99.89% versus 99.87%), lowered FNR (0.143% versus 0.179%) and raised MCC marginally (0.9977 versus 0.9974), while training time fell from 779.1 s to 448.2 s (−42%) and latency from 4.16 ms to 2.25 ms (−46%) (Table 6). The two settings differ by a single test sample: both make three false-positive errors, and the 64-token model misses four malicious samples against five for the 128-token model. Because the payloads are short (median 52 characters), 64 tokens is not worse on this data.

Table 6. *Ablation on maximum sequence length for CodeBERT (2 epochs; metrics in %).*

Metric	64 tokens	128 tokens
Accuracy	99.89	99.87
F1	99.87	99.86
Precision	99.89	99.89
Recall	99.86	99.82
Specificity	99.91	99.91
FPR	0.087	0.087
FNR	0.143	0.179
MCC	0.9977	0.9974
ROC-AUC	1.0000	1.0000
Log-loss	0.0064	0.0070
Training time (s)	448.2	779.1
Latency (ms/sample)	2.25	4.16

Figure 7 shows training time and latency.

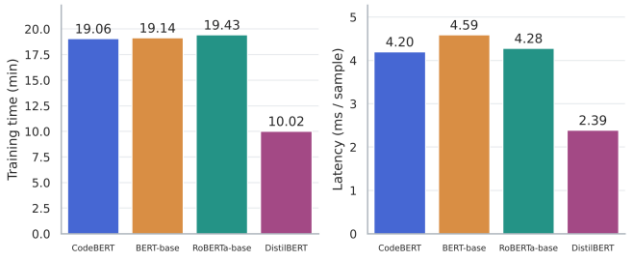


Figure 7. *Training time and amortised latency of the four encoders.*

3.6 Time factors

Training time was almost identical for the three 12-layer models (about 19 minutes), whereas DistilBERT trained in about 10 minutes (0.53× the CodeBERT time) and had the lowest latency (2.39 ms, 0.57× CodeBERT) at an accuracy 0.03 percentage points below CodeBERT (Table 4). For the standalone CodeBERT Run B, training took 1171.04 s and evaluating the 6,259 test samples took 26.75 s, i.e. 4.27 ms per sample. These are end-to-end throughput figures measured in a shared cloud notebook, not service latencies. Figure 8 shows the stress-test confusion counts.

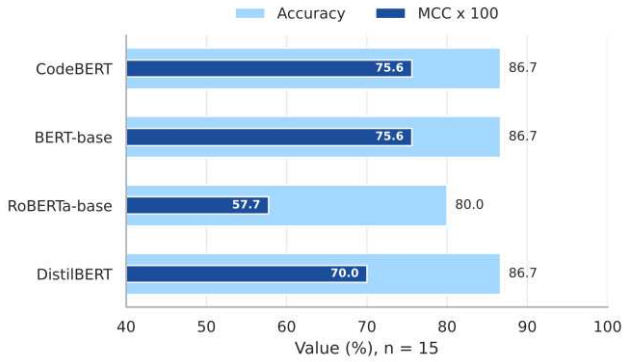


Figure 8. *Obfuscation stress test: confusion counts of the four encoders (15 samples).*

3.7 Obfuscation stress test

On the 15 stress samples (version 2, Table 7), CodeBERT and BERT-base both reached 86.67% accuracy, 88.89% F1 and MCC 0.7559, with precision 100% and recall 80%; DistilBERT reached 86.67% accuracy with F1 90.00% and MCC 0.7000, and RoBERTa-base was lowest (80.00% accuracy, MCC 0.5774). This is a clear drop from the test-set scores. One sample changes accuracy by 6.7 percentage points on a set this size, so the ranking among models should not be interpreted. The comparison is also confounded: the script loaded the most recently modified checkpoint folder of each model, which corresponds to different training points (Table 7). The per-sample log for version 1 shows that CodeBERT classified all five benign look-alikes correctly and missed two of ten malicious samples: a UNION query behind a MySQL version-conditional comment, and a string-concatenation query ending in `(SELECT @@version)`. In version 2 the fifth SQLi payload was replaced by `|| (SELECT 'a')`, whose label should be re-checked if version 2 is kept. Mean per-sample latency at batch size 1 was 16.39 ms in the single-model script and 8.33 ms for CodeBERT in the four-model script; the scripts time different operations and include first-call overhead.

Table 7. *Obfuscation stress test (15 samples, version 2; in %). Lat. = mean ms per sample at batch size 1; Step = training step of the loaded checkpoint (CodeBERT: epoch 4 of 4; BERT-base and RoBERTa-base: epoch 2 of 3; DistilBERT: epoch 3 of 3).*

Model	Acc.	F1	Prec.	Rec.	MCC	Lat.	Step
CodeBERT	86.67	88.89	100.00	80.00	0.7559	8.33	1828
BERT-base	86.67	88.89	100.00	80.00	0.7559	8.66	914
RoBERTa-base	80.00	84.21	88.89	80.00	0.5774	8.37	914
DistilBERT	86.67	90.00	90.00	90.00	0.7000	4.30	1371

3.8 Limitations

(i) The task is binary; SQLi and XSS were not evaluated separately. (ii) Only one pair of public datasets was used and there is no external test set. (iii) Per-sample predictions were not stored, so confusion matrices were recovered from logged metrics and paired tests are limited to best-case bounds. (iv) Run-to-run variability was measured for CodeBERT only, with three runs of unequal design. (v) The 25 hard negatives are too few to change aggregate statistics and their effect was not isolated. (vi) No text normalisation was applied. (vii) The stress set has 15 samples in two versions, the loaded checkpoints correspond to different training lengths, and no automated attack such as WAF-A-MoLE [3] was run. (viii) Latency was measured end-to-end in a shared cloud notebook, and the ablation used two epochs whereas the comparison used three. (ix) One learning rate was used for all models without per-model tuning.

4 Conclusions

Fine-tuned CodeBERT detected malicious SQLi and XSS payloads with 99.89% accuracy, 99.87% F1 and MCC 0.9977 on a de-duplicated test set of 6,259 samples, with 7 errors. Under an identical protocol RoBERTa-base, DistilBERT and BERT-base made 8, 9 and 12 errors; their 95% intervals overlap, no pairwise McNemar test could reach significance for the observed error counts, and CodeBERT's own run-to-run range (6 to 11 errors) is as wide as the range across architectures. The evidence supports comparable performance, not superiority. The models differ in error profile (BERT-base missed the most attacks, RoBERTa-base raised the most false alarms) and in cost: DistilBERT needed about half the training time and latency, and limiting CodeBERT to 64 tokens cut training time by 42% and latency by 46% while changing a single test prediction. The small obfuscation stress test shows that in-distribution scores overstate robustness to evasion.

5 Future Work

Classifying the subtypes of SQLi and XSS, that is, a multi-class setting that separates benign samples from each attack subtype, is the next step of this work.

Data Availability

The two source datasets are public on Kaggle: the SQL Injection Dataset [28] and the cross-site scripting dataset for deep learning [29] (files `Modified_SQL_Dataset.csv` and `XSS_dataset.csv`). The 25 hard-negative samples and the code will be released after the review period to preserve anonymity.

References

[1] OWASP Foundation. OWASP Top 10:2021. https://owasp.org/Top10/ (2021).

[2] Halfond, W. G. J., Viegas, J. & Orso, A. A classification of SQL-injection attacks and countermeasures. In Proc. IEEE International Symposium on Secure Software Engineering (2006).

[3] Demetrio, L., Valenza, A., Costa, G. & Lagorio, G. WAF-A-MoLE: evading web application firewalls through adversarial machine learning. In Proc. 35th Annual ACM Symposium on Applied Computing (SAC '20), 1745–1752 (2020). https://doi.org/10.1145/3341105.3373962

[4] Applebaum, S., Gaber, T. & Ahmed, A. Signature-based and machine-learning-based web application firewalls: a short survey. In International Workshop on Machine Learning and Natural Language Processing for Cybersecurity (ML-NLP4Cybersecurity) (2021).

[5] Vaswani, A. et al. Attention is all you need. In Advances in Neural Information Processing Systems 30 (2017).

[6] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In Proc. NAACL-HLT, 4171–4186 (2019).

[7] Liu, Y. et al. RoBERTa: a robustly optimized BERT pretraining approach. arXiv:1907.11692 (2019).

[8] Sanh, V., Debut, L., Chaumond, J. & Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 (2019).

[9] Feng, Z. et al. CodeBERT: a pre-trained model for programming and natural languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, 1536–1547 (2020).

[10] Gururangan, S. et al. Don't stop pretraining: adapt language models to domains and tasks. In Proc. 58th Annual Meeting of the Association for Computational Linguistics, 8342–8360 (2020).

[11] Kruegel, C. & Vigna, G. Anomaly detection of web-based attacks. In Proc. 10th ACM Conference on Computer and Communications Security (CCS), 251–261 (2003).

[12] Valeur, F., Mutz, D. & Vigna, G. A learning-based approach to the detection of SQL attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2005), Lecture Notes in Computer Science vol. 3548, 123–140 (Springer, 2005).

[13] Alghofaili, L. N. & Ibrahim, D. M. Web-based attacks detection using deep learning techniques: a comprehensive review. Indones. J. Electr. Eng. Comput. Sci. 39, 466–484 (2025). https://doi.org/10.11591/ijeecs.v39.i1.pp466-484

[14] Fang, Y., Li, Y., Liu, L. & Huang, C. DeepXSS: cross site scripting detection based on deep learning. In Proc. 2018 International Conference on Computing and Artificial Intelligence (ICCAI), 47–51 (2018).

[15] Yan, H. et al. Cross-site scripting attack detection based on a modified convolution neural network. Front. Comput. Neurosci. 16, 981739 (2022). https://doi.org/10.3389/fncom.2022.981739

[16] Maurel, H., Vidal, S. & Rezk, T. Statically identifying XSS using deep learning. In Proc. 18th International Conference on Security and Cryptography (SECRYPT) (2021). https://doi.org/10.5220/0010537000002998

[17] Odeh, A. & Abu Taleb, A. XSSer: hybrid deep learning for enhanced cross-site scripting detection. Bull. Electr. Eng. Inform. 13(5) (2024). https://doi.org/10.11591/eei.v13i5.7905

[18] Bharath, M. B. & Latha, N. R. Privacy-preserving federated hybrid deep learning framework for robust cross-site scripting detection in distributed web applications. J. Intell. Fuzzy Syst. (2026). https://doi.org/10.1177/18758967261462971

[19] Chindrus, C. & Caruntu, C.-F. Machine learning-based multilabel classification for web application firewalls: a comparative study. Electronics 14, 4172 (2025).

[20] Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S. & El-Shafai, W. Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Sci. Rep. 14, 1803 (2024). https://doi.org/10.1038/s41598-023-48845-4

[21] Musoke, L. L. & Badii, A. Enhancing web application firewalls with BERT-GNN for SQL injection detection. arXiv:2608.28882 (2026).

[22] Szegedy, C. et al. Intriguing properties of neural networks. In 2nd International Conference on Learning Representations (ICLR) (2014).

[23] Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. In 3rd International Conference on Learning Representations (ICLR) (2015).

[24] Biggio, B. & Roli, F. Wild patterns: ten years after the rise of adversarial machine learning. Pattern Recognit. 84, 317–331 (2018).

[25] Jin, D., Jin, Z., Zhou, J. T. & Szolovits, P. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. Proc. AAAI Conf. Artif. Intell. 34, 8018–8025 (2020).

[26] Allamanis, M. The adverse effects of code duplication in machine learning models of code. In Proc. ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!), 143–153 (2019).

[27] Mosbach, M., Andriushchenko, M. & Klakow, D. On the stability of fine-tuning BERT: misconceptions, explanations, and strong baselines. In 9th International Conference on Learning Representations (ICLR) (2021).

[28] sajid576. SQL Injection Dataset. Kaggle. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset (accessed 5 November 2026).

[29] syedsaqlainhussain. Cross site scripting XSS dataset for deep learning. Kaggle. https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning (accessed 5 November 2026).

[30] Wolf, T. et al. Transformers: state-of-the-art natural language processing. In Proc. 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38–45 (2020).

[31] Paszke, A. et al. PyTorch: an imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32 (2019).

[32] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units. In Proc. 54th Annual Meeting of the Association for Computational Linguistics, 1715–1725 (2016).

[33] Wu, Y. et al. Google's neural machine translation system: bridging the gap between human and machine translation. arXiv:1609.08144 (2016).

[34] Loshchilov, I. & Hutter, F. Decoupled weight decay regularization. In 7th International Conference on Learning Representations (ICLR) (2019).

[35] Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. In 3rd International Conference on Learning Representations (ICLR) (2015).

[36] Sun, C., Qiu, X., Xu, Y. & Huang, X. How to fine-tune BERT for text classification? In Chinese Computational Linguistics (CCL 2019), Lecture Notes in Computer Science vol. 11856, 194–206 (Springer, 2019).

[37] Pedregosa, F. et al. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830 (2011).

[38] Matthews, B. W. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim. Biophys. Acta 405, 442–451 (1975).

[39] Chicco, D. & Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics 21, 6 (2020).

[40] Fawcett, T. An introduction to ROC analysis. Pattern Recognit. Lett. 27, 861–874 (2006).

[41] Wilson, E. B. Probable inference, the law of succession, and statistical inference. J. Am. Stat. Assoc. 22, 209–212 (1927).

[42] Clopper, C. J. & Pearson, E. S. The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika 26, 404–413 (1934).

[43] Efron, B. Bootstrap methods: another look at the jackknife. Ann. Stat. 7, 1–26 (1979).

[44] McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika 12, 153–157 (1947).