

Machine Learning and Deep Learning Approaches to SQL Injection Detection in Web Application Firewalls: A Comparative Empirical Synthesis

Sabil Shrestha
M2 Data Science
Polytech Nantes, Nantes Université
Nantes, France
sabil.shrestha@etu.univ-nantes.fr

Sulakshyana Ghimire
M2 Smart Computing
Faculty of Sciences and Techniques
Nantes Université, Nantes, France
sulakshyana.ghimire@etu.univ-nantes.fr

Krishiyana Bhakta
M2 Data Science
Polytech Nantes, Nantes Université
Nantes, France
krishiyana.bhakta@etu.univ-nantes.fr

Abstract—Web application firewalls (WAFs) remain the primary application-layer defense against SQL injection (SQLi), cross-site scripting (XSS), and distributed denial-of-service (DDoS) traffic, yet traditional signature-based WAFs are structurally unable to generalize to zero-day or obfuscated payloads. Over the past three years, researchers have proposed replacing or augmenting signature matching with learned classifiers—models like recurrent deep networks, classical artificial neural networks (ANNs), and, most recently, transformer-embedding ensembles. This paper synthesizes three representative studies spanning this progression: a layered long short-term memory (LSTM) architecture for combined DDoS/XSS/SQLi detection (Dawadi, Adhikari & Srivastava, 2023); an ANN-based Web Application Firewall using Artificial Neural Networks (WAFANN) targeted specifically at SQLi (Elegbeleye, Chris & Koranteng, 2022); and a DistilBERT-Stacked Ensemble hardened with adversarial training for real-time SQLi detection (Musoke, Badii & Ashlam, n.d.). We define a standardized comparative framework—detection accuracy, precision, recall, F1-score, training time, inference latency, and resource utilization (CPU and memory consumption)—and map every metric each study actually reports onto that framework. The central empirical finding is that accuracy across modern approaches has become nearly saturated (89.3%–99.8% depending on task and dataset), so the differentiating variable for production deployment is now latency and robustness rather than raw accuracy: the DistilBERT-Stacked Ensemble matches the best single transformer-embedded classifier’s accuracy (99.81% vs. 99.82%, not statistically distinguishable by McNemar’s test) while executing roughly 140× faster and retaining 99.77% accuracy under single-step FGSM adversarial perturbation. A second, equally important finding is a gap in the literature: none of the three reviewed studies measures CPU or memory utilization at inference time, a metric essential for capacity planning in a real-time WAF; this paper flags that gap explicitly rather than estimating it, and proposes a standardized control-dataset protocol for closing it in future work. We conclude with concrete recommendations for engineering teams selecting between layered sequence models, classical ANN pipelines, and transformer-embedding ensembles.

Keywords —Web application firewall, SQL injection, machine learning, deep learning, LSTM, artificial neural network DistilBERT, adversarial robustness, comparative benchmark

I. INTRODUCTION

One of the common challenges in computer science is protecting computers and networks from infiltration, theft,

and disruption [1]. As web applications grow in functionality, transaction volume and data sensitivity, the need for advanced defences becomes critical [2]. Web applications rely on rule-based and signature-matching systems as a first line of defence [3]. A Web Application Firewall (WAF) acts as a protective layer between a web application and clients on the Internet, helping to identify and prevent malicious requests and attacks targeting web applications [4], [5]. Signature-based WAFs operate by matching incoming HTTP traffic against a curated blacklist of known-malicious patterns. This paradigm is fast and interpretable but fails against zero-day payloads, string concatenation, inline-comment obfuscation, and encoding-based evasion. Attackers are smarter and they find techniques to breach the system. Hence, a signature-based protection system is not always a reasonable solution [7], [8]. This is because it can only recognize what it has already been told to recognize. Different methods of attacks employed by cyberattacks could cripple targeted systems [9]. Also, SQLi attacks are among the most damaging web threats: an adversary manipulates database queries to access sensitive data and compromise the underlying system [10]. Deep learning and NLP models can improve detection accuracy and reduce error rates, yet challenges persist for ML-integrated WAFs [11]. All three studies reviewed here identify this same structural limitation as their point of departure, and each proposes a different learned-classifier paradigm to replace or supplement the signature layer.

The three studies do not converge on a single benchmark methodology. They differ in target attack scope (combined DDoS+XSS+SQLi vs. SQLi-only), in dataset provenance (public network-traffic corpora vs. a public Kaggle text corpus vs. researcher-generated web-application traffic), in classifier family (recurrent deep network vs. shallow feed-forward ANN vs. transformer-embedding ensemble), and in what they measure (two of the three studies report no formal precision/recall/F1 breakdown at all). This heterogeneity is itself the central problem this paper addresses: a reader trying to select an architecture for a production WAF cannot directly compare the headline accuracy figures of these papers without

first understanding what was actually measured, on what data, and under what conditions. Table I summarizes the model categories and scope of each study before any cross-study numeric comparison is attempted.

A. Model and Study Overview

A second recurring theme across the literature these papers cite (e.g., Ito & Iyatomi, 2018; Gharibeh, Melhem et al., 2020; Althubiti, Yuan et al., 2017) is that shallow, non-sequential classifiers such as Naïve Bayes and, to a lesser extent, plain logistic regression on raw or sparse features tend to underperform relative to models that either (a) exploit sequential/temporal structure in traffic (LSTM) or (b) exploit dense contextual embeddings of the request text (DistilBERT). This motivates the shift, visible across the three studies in chronological order, from classical ANN pattern-matching (2022) toward deep sequential modeling (2023) and finally toward transformer-embedding ensembles (most recent).

II. METHODOLOGY AND EXPERIMENTAL SETUP

This section has two parts. Section II-A documents, for full transparency, the actual experimental conditions under which each source study produced its numbers. The conditions are not uniform, and any comparative table built later in this paper (Section IV) must be read against this context rather than as a controlled head-to-head benchmark. Section II-B then proposes a standardized control protocol that a future replication study would need to adopt in order to produce a genuinely controlled comparison, including a common dataset schema and the resource-utilization metrics that none of the three studies currently reports.

A. Experimental Conditions as Actually Reported

Because Table II shows no shared dataset, split protocol, or hardware baseline, any single master table that places all reported numbers side by side (Section IV-A) must be interpreted as a survey of what each study found under its own conditions, not as evidence that one architecture would beat another on identical input. Only the internal comparisons performed within a single study—for example, Musoke et al.’s comparison of seven base classifiers against a common DistilBERT embedding and a common test set—constitute a controlled experiment in the strict sense.

B. Proposed Standardized Control Protocol (for Future Replication)

To make the three architectures genuinely comparable, a future benchmark study should fix the following before any model is trained, rather than allow each architecture to define its own dataset and metrics as the current literature does:

- **Control dataset:** a single labelled HTTP request corpus containing request line, headers, cookies, and POST body, with an explicit SQLi/XSS/DDoS/benign label per record, and a fixed minimum size (e.g., 100,000 records) balanced across classes so that no single study’s oversampling strategy (SMOTE vs. class-weighting) confounds the comparison.
- **Table/record structure:** request_id (int, PK), timestamp (datetime), source_ip (varchar), method (enum: GET/POST), raw_query (text), header_length (int), ttl (int), flags (bitmask/int), label (enum: benign/sql_i/xss/ddos), attack_subtype (varchar, nullable).
- **System isolation:** every model trained and evaluated on the same containerized environment (identical CPU core count, RAM allocation, and—where GPU-dependent—identical GPU model), with all other processes suspended during timed runs, and each timing run repeated at least 10 times to report mean $\pm$ standard deviation rather than a single wall-clock reading.
- **Common evaluation harness:** a single scoring script computing accuracy, precision, recall, F1-score, training wall-clock time, per-request inference latency, peak CPU utilization (%), and peak resident memory (MB), sampled at fixed intervals (e.g., every 100 ms) throughout both training and inference using a standard process-monitoring utility (e.g., psutil in Python).

No results in Section IV of this paper were produced by executing this protocol; it is presented as the concrete next step needed to close the comparability gap identified in Section II-A, and specifically to obtain the CPU/memory figures that are absent from all three source studies.

C. Evaluation Metrics Defined

III. METHOD DESCRIPTIONS

A. Layered LSTM Architecture (Dawadi, Adhikari & Srivastava, 2023)

The system cascades two independently trained bidirectional LSTM classifiers. Layer 1 screens all incoming traffic for DDoS behavior using network-level features (protocol, time-to-live, header length, Don’t-Fragment flag, source/destination port); because DDoS volumes vastly exceed legitimate traffic volumes during an attack, filtering here first removes the majority of malicious volume before the more expensive Layer 2 inspection. Only traffic that Layer 1 passes as normal is forwarded to Layer 2, which re-parses the HTTP header and body for XSS/SQLi indicators.

Implementation detail: the DDoS layer was trained to 40 epochs, reaching 97.57% training/test accuracy; the XSS/SQLi layer was trained to 100 epochs, reaching 89.34% accuracy. Live-traffic validation on a held-out proxy correctly classified 96% of new DDoS traffic and 90% of new HTTP payloads passed to the second layer. Framework/library versions are not specified in the source paper. Note that these training-epoch accuracy figures (97.57%, 89.34%) are the values as originally reported by the source authors during model fitting; the confusion-matrix-derived figures used later in Table IV (97.14%, 90.04%) are computed by this synthesis from separately published live-traffic confusion matrices (Figures 13–14) and should be read as a distinct, independently-sourced measurement rather than a contradiction of the training-time figures above.

TABLE I
CLASSIFIER FAMILY AND ATTACK SCOPE OF THE THREE SYNTHESIZED STUDIES.

Study	Classifier Family	Attack Scope	Deployment Context
Dawadi, Adhikari & Srivastava (2023)	Layered / cascaded LSTM (deep recurrent network)	DDoS (Layer 1) → XSS + SQLi (Layer 2)	Network-traffic-facing WAF; live-traffic proxy validation
Elegbeleye, Chris & Koranteng (2022)	Two-layer feed-forward Artificial Neural Network (backpropagation)	SQLi (primary), limited XSS discussion	Java servlet/JSP WAF (WAFANN, built on webCastellum)
Musoke, Badii & Ashlam (n.d.)	DistilBERT contextual embeddings → stacked ensemble (LR + SVM + XGBoost → neural meta-learner), FGSM-hardened	SQLi (text-classification framing)	Real-time WAF detector; latency- and adversarial-robustness-benchmarked

TABLE II
AS-REPORTED EXPERIMENTAL CONDITIONS FOR EACH SOURCE STUDY. NOTE THE ABSENCE OF ANY COMMON DATASET, SPLIT PROTOCOL, OR HARDWARE SPECIFICATION ACROSS STUDIES.

Study	Dataset(s) Used	Train / Test Split	Preprocessing Pipeline	Compute Environment
Dawadi et al. (2023)	IDS ISCX 2012; CIC-DDoS 2019; CSIC 2010; researcher-generated dataset (DVWA + Burp Suite + Wireshark capture, ~5,700 XSS/SQLi payloads; LOIC/hulk-generated DDoS traffic)	60% train / 20% validation / 20% test (generated dataset); live-traffic test of 2,279 samples (XSS/SQLi layer) and ~53,133 samples (DDoS layer)	Wireshark / URL decoding → feature & parameter selection → numericalization (GET=1, POST=2; flags → {0,1}) → min-max normalization → sliding-window reshaping for sequential input	Not reported (no hardware, framework version, or library specification given)
Elegbeleye et al. (2022)	Demonstration XOR-pattern dataset (40,000 training iterations); live testing on 10 researcher-built web applications (5 weak-security, 5 strong-security)	Not reported as a formal split; qualitative demonstration only	SQL keyword / special-character signature extraction; PreparedStatement/JDBC parameter binding for baseline SQLi prevention; black-/white-list signature matching prior to ANN classification	MATLAB (for ANN training); Java servlets/JSP/JavaBeans (for the deployed WAFANN)
Musoke et al. (n.d.)	Kaggle SQL-injection dataset, 30,919 labelled instances (63.2% benign / 36.8% malicious)	80% / 20% train-test split; 5-fold cross-validation for hyperparameter search and meta-feature generation	Lower-casing, whitespace stripping, DistilBERT tokenization (padding/truncation), 768-dimensional embedding extraction; SMOTE oversampling for TF-IDF baselines, class-weighting for the embedding pipeline	Google Colab Pro, NVIDIA Tesla T4 GPU (embeddings and meta-learner); scikit-learn on CPU runtime (base classifiers); Optuna for hyperparameter search

TABLE III
METRIC DEFINITIONS AND THEIR ACTUAL REPORTING STATUS ACROSS THE THREE SOURCE STUDIES. ROWS SHADED BY ABSENCE ARE THE SPECIFIC GAP THIS PAPER FLAGS RATHER THAN FABRICATES.

Metric	Definition	Reported by which source studies?
Accuracy	$(TP+TN) / (TP+TN+FP+FN)$ on the held-out test set	All three (Elegbeleye et al. qualitatively only)
Precision	$TP / (TP+FP)$ for the malicious class	Musoke et al. directly; derivable from Dawadi et al.'s reported confusion matrices; not reported by Elegbeleye et al.
Recall	$TP / (TP+FN)$ for the malicious class	Same as Precision
F1-Score	Harmonic mean of Precision and Recall	Same as Precision
Training Time	Wall-clock time to fit the model on the training split	Musoke et al. only
Execution / Inference Latency	Wall-clock time to classify the full test set (or per-request latency)	Musoke et al. only
Adversarial Robustness	Accuracy retained under a defined perturbation (e.g., FGSM) or query-level obfuscation	Musoke et al. only
Max CPU Usage (%)	Peak CPU utilization sampled during training/inference	Not reported by any of the three studies
Max Memory Usage (%)	Peak resident memory sampled during training/inference	Not reported by any of the three studies

Algorithm 1 Layered LSTM Detection

```

1: INPUT: raw_packet_stream
2:  $decoded \leftarrow \text{decode}(raw\_packet\_stream)$   $\triangleright$  Wireshark capture / URL-decoding
3:  $features \leftarrow \text{select\_features}(decoded, attack\_type)$ 
4: DDoS: protocol, TTL, header_len, DF_flag, src/dst_port
5: XSS/SQLi: parsed HTTP header + body tokens
6:  $features \leftarrow \text{numericalize}(features)$   $\triangleright$  GET=1, POST=2; flags  $\rightarrow \{0, 1\}$ 
7:  $features_{norm} \leftarrow \text{minmax\_normalize}(features)$   $\triangleright$   $z_i = (x_i - \min(x)) / (\max(x) - \min(x))$ 
8:  $X \leftarrow \text{sliding\_window}(features_{norm}, size = z)$   $\triangleright$  reshape to  $(m - z) \times z \times n$ 
9: split  $X$  into train (60%) / validation (20%) / test (20%)
10:  $model \leftarrow \text{BiLSTM}(activation='tanh', out\_activation='sigmoid', regularizer=L2, optimizer=Adam, loss=binary\_crossentropy)$ 
11:  $\text{train}(model, X_{train}, y_{train})$ 
12: for each new_request  $r$  do
13:   if  $DDoS\_model(r) == \text{malicious}$  then
14:     DROP( $r$ ); continue
15:   else
16:     if  $XSS\_SQLi\_model(r) == \text{malicious}$  then
17:       DROP( $r$ )
18:     else
19:       FORWARD( $r, webserver$ )
20:     end if
21:   end if
22: end for
23: end for

```

B. ANN-Based WAFANN (Elegbeleye, Chris & Koranteng, 2022)

WAFANN is a two-layer feed-forward network trained with backpropagation, deployed inside a Java servlet/JSP web application firewall descended from webCastellum. Before the ANN is invoked, incoming request components (headers, cookies, POST data, query strings) are checked against a black-list/white-list signature table; only signatures that do not match a known-safe pattern are passed to the ANN, which classifies the parameter content as malicious or legitimate based on the distribution of SQL keywords and special characters. The study additionally demonstrates a complementary, non-learned mitigation—replacing string-concatenated SQL statements with parameterized PreparedStatement queries—which prevents the classic Boolean-based ($' or '1'='1$) and union-based injection patterns regardless of the ANN's classification.

Reporting limitation: the source paper trains the network on an XOR-pattern demonstration for 40,000 iterations and reports only that “the expected value and the prediction value are close to each other,” without a numerical accuracy, precision, recall, or F1 figure on the SQLi detection task itself, and without any timing or resource measurement. The paper's empirical evidence for real-world efficacy is qualitative: successful

Algorithm 2 ANN-Based WAFANN Detection

```

1: INPUT: HTTP_request
2:  $params \leftarrow \text{extract\_parameters}(request.headers, request.cookies, request.postData, request.queryString)$ 
3: for each  $param$  in  $params$  do
4:   if matches_whitelist( $param$ ) then
5:     continue
6:   end if
7:   if matches_blacklist( $param$ ) then
8:     block( $request$ ); return
9:   end if
10:   $feature\_vector \leftarrow \text{encode}(param)$   $\triangleright$  SQL keyword / special-character distribution
11:   $result \leftarrow \text{ANN\_forward\_pass}(feature\_vector, W, b)$   $\triangleright$  2-layer backprop network
12:  if  $result == \text{malicious}$  then
13:    block_and_block( $request$ ); return
14:  end if
15: end for
16: end for
17: forward( $request, webserver$ )

```

blocking of Boolean-based and union-based SQLi against five weak-security and five strong-security test applications, with an explicitly stated limitation that non-Boolean, non-union injection techniques were not demonstrated as prevented.

C. DistilBERT-Stacked Ensemble (Musoke, Badii & Ashlam, n.d.)

This architecture first encodes each raw SQL query with DistilBERT into a 768-dimensional contextual embedding. Seven classical classifiers (Logistic Regression, Decision Tree, Random Forest, SVM, Naïve Bayes, XGBoost, Gradient Boosting) are trained on this embedding and ranked by cross-validated accuracy, precision, recall, and F1; the top three (Logistic Regression, SVM, XGBoost) contribute their class probabilities as six meta-features to a compact feed-forward neural meta-learner. The meta-learner is hardened against evasion using Fast Gradient Sign Method (FGSM) adversarial examples during training and tuned with the Optuna hyperparameter-optimization framework.

Implementation detail: embeddings and the neural meta-learner were computed on an NVIDIA Tesla T4 GPU (Google Colab Pro); the seven base classifiers ran on the CPU runtime via scikit-learn. Optuna jointly tuned hidden-layer width, dropout, learning rate, epoch count, and the FGSM perturbation budget ϵ over the cross-validated accuracy objective. A McNemar's exact test was used to assess whether the final ensemble's accuracy differed significantly from the best single base model.

IV. DISCUSSION AND EMPIRICAL COMPARATIVE ANALYSIS

This section reports every metric that is actually documented in the three source studies. Cells marked “N/R” (Not Reported) indicate the metric was not measured or not disclosed in that

Algorithm 3 DistilBERT-Stacked Ensemble Detection

```

1: INPUT: SQL_query_text
2:  $tokens \leftarrow \text{DistilBERT\_tokenizer}(query\_text, padding=Truncating)$ 
3:  $E \leftarrow \text{DistilBERT}(tokens)$   $\triangleright$  768-dim contextual embedding
4: for  $model$  in  $\{LogisticRegression, SVM, XGBoost\}$ 
   do  $\triangleright$  top-3 of 7 candidates, GridSearchCV-tuned
5:    $p_{model} \leftarrow model.predict\_proba(E)$ 
6: end for
7:  $meta\_features \leftarrow \text{concat}(p_{PLR}, p_{SVM}, p_{XGB})$   $\triangleright$  6-dimensional vector [training only]
8:  $meta\_features_{adv} \leftarrow \text{FGSM\_perturb}(meta\_features, \epsilon=0.10)$ 
9:  $y_{pred} \leftarrow \text{MetaLearner}(meta\_features)$ 
    $\text{in}(6) \rightarrow \text{hidden}(170 \text{ units, ReLU}) \rightarrow \text{dropout}(0.421)$ 
    $\rightarrow \text{out}(2, \text{Adam}(lr=0.0083))$ 
10: RETURN  $y_{pred}$   $\triangleright$  benign / malicious

```

study—these cells are deliberately left unfilled rather than estimated, per the position taken in Section II-A. Accuracy, precision, recall, and F1 for the two LSTM layers are computed by the authors of this synthesis directly from the raw confusion-matrix counts published in the source paper (Dawadi et al., 2023, Figures 13–14), and are marked accordingly; all Musoke et al. figures are quoted directly from that paper’s own tables.

A. Master Comparison Table

B. Per-Technique Performance Discussion

Layered LSTM (Dawadi et al., 2023). The DDoS layer achieves very high recall (99.50% by our derivation), meaning it misses very few actual DDoS packets [12]—appropriate given that an undetected DDoS flood is far more damaging than an occasional false positive on legitimate traffic. The XSS/SQLi layer shows the inverse profile: very high precision (99.25%) but comparatively low recall (80.67%), meaning that when it flags a request as malicious it is almost always right, but it misses roughly one in five actual attacks (matching the 221 false negatives out of 1,143 malicious test samples reported in the paper’s own confusion matrix). No training time, inference latency, or resource-utilization figures are given, which limits this study’s usefulness for direct capacity planning.

ANN / WAFANN (Elegbeleye et al., 2022). This study’s contribution is architectural and procedural rather than quantitatively benchmarked: it demonstrates that combining ANN-based signature classification with parameterized-query enforcement (PreparedStatement) closes two named attack vectors (Boolean-based and union-based SQLi) across ten test applications. Because no accuracy, precision/recall, latency, or resource figures are reported, this study cannot be ranked numerically against the other two in Table IV; it is included in this synthesis for architectural completeness and because it is explicitly cited as related prior work by Musoke et al.

DistilBERT-Stacked Ensemble family (Musoke et al.).

Among the individual base classifiers, DistilBERT-SVM is the most accurate (99.82%) but also the slowest at inference (1.896 s to classify the full test set) and among the slowest to train (471.57 s). DistilBERT-Logistic Regression is the fastest to both train (28.32 s) and infer (0.0168 s) while sacrificing only 0.06 percentage points of accuracy relative to SVM. The optimized stacked ensemble sits almost exactly at the SVM’s accuracy ceiling (99.81% vs. 99.82%, a difference the paper’s own McNemar’s exact test found not statistically significant, $p \approx 1.0$) while requiring only 0.01356 s to classify the same test set—roughly 140 $\times$ faster than SVM alone. The ensemble additionally retains 99.77% accuracy under a single-step FGSM adversarial perturbation of its internal meta-feature representation, and a supplementary query-level obfuscation test shows recall holding at 99.74% under case/whitespace rewrites but degrading to 96.38% under inline SQL comment (/ ** /) obfuscation—identified by the original authors themselves as a concrete remaining evasion vector rather than a solved problem.

C. Best Technique by Objective

V. CONCLUSION AND PRACTICAL RECOMMENDATIONS

Three conclusions follow from this synthesis. First, detection accuracy alone is no longer a strongly differentiating criterion among modern SQLi/WAF classifiers: every architecture reviewed here that reports a numeric accuracy figure clears 89% at minimum, and the strongest transformer-embedding classifiers exceed 99.7%. Engineering teams selecting a detection architecture in 2025 and beyond should therefore weight latency, training/operational cost, and adversarial robustness at least as heavily as headline accuracy.

Second, architectural pattern matters independently of the underlying classifier. Dawadi et al.’s layered cascade—filtering high-volume, cheaply-classified traffic (DDoS) before invoking a more expensive second-stage classifier (XSS/SQLi)—is a sound systems-design pattern that could, in principle, be combined with the DistilBERT-Stacked Ensemble’s meta-learning approach: use a fast first-pass filter to shed volumetric attacks, then apply the higher-accuracy, low-latency ensemble only to the traffic that survives the first pass. Neither source paper tests this combination, and it is a natural direction for future empirical work.

Third, and most concretely for engineering teams: the current published literature on ML-based WAFs, including all three studies reviewed here, does not report CPU or memory utilization at training or inference time. This is a meaningful gap, because a classifier’s wall-clock latency figure (as reported by Musoke et al.) does not by itself indicate whether that latency was achieved on a single CPU core, a GPU, or a large memory footprint—all of which materially affect the cost and scalability of deploying the model inline in front of production traffic. Teams should not assume the DistilBERT-based approach is “cheap” simply because its measured latency is low: transformer embedding extraction is typically GPU-memory-intensive relative to the LSTM or

TABLE IV

MASTER COMPARISON ACROSS ALL TECHNIQUES FOR WHICH QUANTITATIVE FIGURES ARE AVAILABLE. * = VALUE COMPUTED BY THE AUTHORS OF THIS SYNTHESIS FROM THE RAW CONFUSION-MATRIX COUNTS REPORTED IN DAWADI ET AL. (2023), NOT STATED VERBATIM BY THE ORIGINAL AUTHORS. † = THE SOURCE STUDY REPORTS ONLY A QUALITATIVE SUCCESS CLAIM (PREDICTED VALUES “CLOSE” TO EXPECTED VALUES ON AN XOR DEMONSTRATION), WITH NO NUMERICAL DETECTION-ACCURACY FIGURE FOR THE SQLi TASK ITSELF. ‡ = META-LEARNER TRAINING TIME ONLY; DOES NOT INCLUDE UPSTREAM DISTILBERT EMBEDDING EXTRACTION OR BASE-CLASSIFIER TRAINING TIME (471.57 s FOR THE SLOWEST CONTRIBUTING BASE MODEL, SVM). LATENCY FIGURES FOR THE MUSOKE ET AL. ROWS ARE WALL-CLOCK TIME TO CLASSIFY THE FULL HELD-OUT TEST SET (6,184 SAMPLES) ON THE HARDWARE DESCRIBED IN SECTION II-A, AND ARE NOT DIRECTLY COMPARABLE TO A PER-REQUEST FIGURE WITHOUT DIVIDING BY TEST-SET SIZE.

Technique	Acc.	Prec.	Recall	F1	Train Time	Latency	Adv. Robustness	Max CPU %	Max Mem %
LSTM – DDoS layer (Dawadi et al., derived)	97.14%*	95.03%*	99.50%*	97.22%*	N/R	N/R	N/R	N/R	N/R
LSTM – XSS/SQLi layer (Dawadi et al., derived)	90.04%*	99.25%*	80.67%*	88.98%*	N/R	N/R	N/R	N/R	N/R
ANN / WAFANN (Elegbel- eye et al.)	N/R†	N/R	N/R	N/R	N/R	N/R	N/R	N/R	N/R
DistilBERT-SVM (Musoke et al.)	99.82%	99.82%	99.82%	99.82%	471.57 s	1.8964 s	N/R	N/R	N/R
DistilBERT-Logistic Regression (Musoke et al.)	99.76%	99.76%	99.76%	99.76%	28.32 s	0.0168 s	N/R	N/R	N/R
DistilBERT-XGBoost (Mu- soke et al.)	99.74%	99.74%	99.74%	99.74%	69.02 s	0.0420 s	N/R	N/R	N/R
DistilBERT-Stacked Ensemble, optimized (Musoke et al.)	99.81%	99.81%	99.81%	99.81%	0.4038 s‡	0.01356 s	99.77% (FGSM, $\epsilon=0.10$)	N/R	N/R

TABLE V

BEST-IN-CLASS TECHNIQUE BY OBJECTIVE, RESTRICTED STRICTLY TO WHAT EACH SOURCE STUDY MEASURED.

Objective	Best-performing technique (as reported)	Basis
Highest raw accuracy	DistilBERT-SVM (99.82%)	Table IV, Musoke et al.
Best speed / lowest inference latency at near-top accuracy	DistilBERT-Stacked Ensemble (0.01356 s per test-set pass; 99.81% accuracy, statistically indistinguishable from the best single model)	Table IV; McNemar’s exact test, $p \approx 1.0$
Fastest to train	DistilBERT-Logistic Regression (28.32 s)	Table IV
Best recall on the malicious class (highest-stakes for security)	LSTM DDoS layer (99.50%, derived)	Confusion matrix, Dawadi et al.
Best precision on the malicious class (fewest false alarms)	LSTM XSS/SQLi layer (99.25%, derived)	Confusion matrix, Dawadi et al.
Only technique with quantified adversarial robustness	DistilBERT-Stacked Ensemble (99.77% under FGSM)	Musoke et al., Section 3.7/4.2
CPU / memory efficiency	Cannot be determined — not measured by any of the three studies	Table III gap

shallow-ANN pipelines, and this cost is invisible in the tables published to date. We recommend that any team adopting one of these architectures run the standardized control protocol proposed in Section II-B—fixed hardware, repeated timing runs, and explicit CPU/memory sampling—against its own production-representative traffic before committing to a deployment choice, rather than relying on the accuracy figures in Table IV in isolation.

In summary: for teams facing combined volumetric (DDoS) and application-layer (XSS/SQLi) threats, a layered cascade architecture in the style of Dawadi et al. remains a reasonable default, provided its unreported latency and resource characteristics are independently benchmarked. For teams facing SQLi-only threats in a latency-sensitive, real-time inline WAF, the DistilBERT-Stacked Ensemble methodology of Musoke et al. currently offers the strongest published combination of accuracy, latency, and adversarially-quantified robustness, with the explicit caveat—acknowledged by its own authors—that inline-comment obfuscation remains an unresolved evasion

vector requiring further hardening. Classical ANN pipelines such as WAFANN remain architecturally instructive but, on the evidence currently published, lack the quantitative validation needed to be recommended over either alternative without further empirical work.

REFERENCES

- [1] M. Krishnan, Y. Lim, S. Perumal, and G. Palanisamy, “Detection and defending the XSS attack using novel hybrid stacking ensemble learning-based DNN approach,” *Digital Communications and Networks*, vol. 10, no. 3, pp. 716–727, 2024. doi: 10.1016/j.dcan.2022.09.024.
- [2] H. Alsobhi and R. Alshareef, “SQL injection countermeasures methods,” in *2020 International Conference on Computing and Information Technology (ICIT-1441)*, 2020.
- [3] S. Toprak and A. Yavuz, “Web Application Firewall Based on Anomaly Detection using Deep Learning,” *Acta Infologica*, 2022.
- [4] A. Moradi Vartouni, M. Teshnehlab, and S. Sedighian Kashi, “Leveraging deep neural networks for anomaly-based web application firewall,” *IET Information Security*, vol. 13, no. 4, pp. 352–361, 2019. doi: 10.1049/iet-ifs.2018.5404.
- [5] D. Appelt, C. D. Nguyen, A. Panichella, and L. C. Briand, “A machine learning-driven evolutionary approach for testing web application fire-

- walls,” *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 733–757, 2018. doi: 10.1109/TR.2018.2805763.
- [6] M. Ito and H. Iyatomi, “Web application firewall using character-level convolutional neural network,” in *2018 IEEE 14th International Colloquium on Signal Processing and Its Applications (CSPA)*, 2018. doi: 10.1109/CSPA.2018.8368694.
- [7] J. Liang, W. Zhao, and W. Ye, “Anomaly-based web attack detection: A deep learning approach,” in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, Kunming, China, 8–10 December 2017, pp. 80–85. doi: 10.1145/3171592.3171594.
- [8] K. O. A. Alimi, K. Ouahada, A. M. Abu-Mahfouz, S. Rimer, and O. A. Alimi, “Refined LSTM based intrusion detection for denial-of-service attack in Internet of Things,” *Journal of Sensor and Actuator Networks*, vol. 11, no. 3, p. 32, 2022. doi: 10.3390/jsan11030032.
- [9] C. Nelson, “Cyber Warfare: The Newest Battlefield,” Washington University in St. Louis, 2011.
- [10] M. Amouei, M. Rezvani, and M. Fateh, “RAT: Reinforcement-Learning-Driven and Adaptive Testing for Vulnerability Discovery in Web Application Firewalls,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3371–3386, 2022.
- [11] A. Shaheed and M. H. D. B. Kurdy, “Web application firewall using machine learning and features engineering,” *Security and Communication Networks*, 2022, pp. 1–14.
- [12] B. R. Dawadi, B. Adhikari, and D. K. Srivastava, “Deep learning technique-enabled web application firewall for the detection of web attacks,” *Sensors*, vol. 23, no. 4, p. 2073, 2023. [Online]. Available: <https://doi.org/10.3390/s23042073>
- [13] F. Elegbeleye, M. Chris, and K. Koranteng, “Machine learning approach to address SQL injection attacks in web applications,” *Ponte International Journal of Sciences and Research*, vol. 78, no. 6/1, 2022. [Online]. Available: <https://doi.org/10.21506/j.ponte.2022.6.3>
- [14] L. L. Musoke, A. Badii, and A. Ashlam, “Enhancing web application firewalls with machine learning for SQL injection detection,” University of Reading, n.d. Dataset and code: <https://github.com/mlily2024/Final-project-SQL-injection-pipeline>