Simen Owen Academic Proceedings Series

4th International Conference on Electronics, Engineering, Computer Science and Applied Development (EESD 2026)

Article

Real-Time Detection of Automated Crawlers in High-Traffic Web Environments: An Integrated Approach

Xizhou Deng[1,*]

[1] Kunming Haibei Chinese-English International School (1903 Campus), Kunming, China
* Correspondence: Xizhou Deng, Kunming Haibei Chinese-English International School (1903 Campus), Kunming, China

Abstract: Automated crawlers are widely used for legitimate Web services but can also generate resource-intensive traffic, distort analytics, and evade simple rule-based filtering in high-traffic environments. Existing approaches commonly rely on static fingerprints, aggregated session statistics, or computationally heavier sequence models, while confidence calibration and workload-dependent decision behavior are often evaluated separately. This study proposes a Load-Aware Integrated Crawler Detector (LAICD) that combines server-side session statistics with a lightweight temporal convolutional encoder, adaptive multi-view fusion, confidence calibration, and a bounded load-aware threshold controller. Evaluation on the held-out Phase 2 Web Bot Detection Dataset showed that LAICD achieved a Macro-F1 of 0.914 ± 0.008 and PR-AUC of 0.953 ± 0.006, compared with 0.892 ± 0.009 and 0.936 ± 0.008 for the strongest BiLSTM baseline. FPR@95TPR decreased from 8.8 ± 0.9% to 6.8 ± 0.7%. Removing confidence calibration increased ECE from 0.031 ± 0.006 to 0.071 ± 0.010. Under EClog replay, p95 inference latency increased from 2.8 ± 0.2 ms at 1× load to 6.2 ± 0.7 ms at 5× load. These results indicate that integrating temporal evidence, calibrated probabilities, and bounded workload adaptation can improve crawler detection while retaining practical runtime behavior and reducing dependence on additional client-side behavioral data.

Keywords: automated crawler detection; Web traffic analysis; temporal modeling; confidence calibration; load-aware classification

1. Introduction

Automated crawlers are now a routine component of Web ecosystems, supporting search indexing, monitoring, archiving, and data aggregation while also being used for aggressive scraping, credential-oriented probing, and resource-intensive access. Recent Web robot research reports that automated traffic can account for a substantial share of Web requests and that malicious bots increasingly imitate legitimate browsing patterns, making identification based only on user-agent strings or request frequency unreliable [1]. Web-log-based classification remains attractive because server-side request records are already available in most production systems and can support detection without additional client-side interaction mechanisms. Analysis of user-specific activities in Web-server logs has also demonstrated that request timing and access characteristics provide useful evidence for identifying anomalous behavior [2].

The problem becomes harder in high-traffic environments, where detection quality must be considered together with latency, false-positive control, and workload variability. Large-scale e-commerce experiments have demonstrated behavior-based bot detection on tens of millions of page visits, but they also expose a practical tension between fast heuristics, static technical fingerprints, and richer behavioral models [3]. Existing defenses remain vulnerable to evasion because bots can alter headers, browser fingerprints, timing

Received: 14 July 2026
Revised: 18 August 2026
Accepted: 31 August 2026
Published: 05 September 2026

patterns, and navigation behavior [4]. More recent session-representation methods improve recognition by modeling variable-length request sequences and spatiotemporal relationships, but they mainly optimize classification performance and do not explicitly coordinate confidence calibration with changing server load [5]. Bot detection may also involve identifiers, fingerprints, or behavioral traces that raise privacy and data-governance concerns; recent regulatory analyses therefore emphasize data minimization and proportionality in security monitoring [6]. These observations reveal a practical gap: current methods rarely integrate lightweight temporal modeling, calibrated confidence, and workload-aware decision control within one server-side pipeline, particularly when request intensity changes rapidly and deployment overhead must remain bounded.

To address this gap, this study develops a Load-Aware Integrated Crawler Detector (LAICD) with three experimentally testable objectives: (1) Multi-view server-log representation. LAICD combines session-level HTTP statistics with a lightweight temporal encoder for request order and inter-request dynamics. It is evaluated against rule-based, tree-based, boosting, and sequential baselines in Table 1, while removal of the temporal branch is examined in Table 2. (2) Bounded load-adaptive decision control. Instead of using one fixed threshold, LAICD updates the operating threshold according to normalized request rate and queue occupancy within validation-defined bounds. The mechanism is tested through the fixed-threshold ablation in Table 2. (3) Confidence-calibrated crawler probability. A calibration term and validation-estimated temperature improve the correspondence between model scores and observed crawler likelihood. Its effect is assessed through calibration ablation, expected calibration error, and confidence analysis in Section 4.5.

Table 1. Detection Performance on the Phase 2 Test Set (Mean ± SD, N = 5).

Method	Macro-F1	PR-AUC	FPR@95TPR (%)
Rule-Based	0.786 ± 0.014	0.846 ± 0.012	18.4 ± 1.7
Random Forest	0.861 ± 0.011	0.913 ± 0.009	11.5 ± 1.2
XGBoost	0.879 ± 0.010	0.928 ± 0.007	9.7 ± 1.0
BiLSTM	0.892 ± 0.009	0.936 ± 0.008	8.8 ± 0.9
LAICD	0.914 ± 0.008	0.953 ± 0.006	6.8 ± 0.7

Technically, incoming HTTP requests are timestamp-synchronized and sessionized before server-side features are extracted. Session statistics and temporal request embeddings are processed in parallel and combined through an adaptive fusion gate. The fused representation is converted into a calibrated crawler probability, after which a bounded controller determines the online threshold from the current traffic state. Training jointly optimizes discrimination, calibration, and prediction consistency under controlled load perturbations.

This design treats classification and runtime decision control as related but distinct problems. Academically, it enables analysis of how temporal evidence, confidence calibration, and system load interact in real-time Web classification. Practically, ordinary server logs, anonymized identifiers, bounded threshold updates, and interpretable feature contributions support robustness, operational explainability, and data-minimization-oriented deployment without claiming automatic regulatory compliance.

2. Related Works

2.1. Rule-Based, Log-Based, and Fingerprint-Based Detection

Operational crawler-detection systems commonly rely on rules derived from user-agent strings, IP addresses, request rates, status-code distributions, cookies, and session statistics. Their main advantage is low computational cost: decisions require limited preprocessing and remain interpretable through explicit traffic attributes. Recent log-based studies also show that clustering and interpretable classifiers can separate much

automated and human activity without complex neural architectures [7]. However, these systems assume that observed attributes remain sufficiently stable. This assumption weakens when automated clients rotate addresses, spoof headers, preserve cookies, or deliberately reduce request frequency.

Browser fingerprinting provides a richer technical view by combining browser, device, and execution characteristics, complementing basic header inspection [8]. Machine-learning classifiers built on fingerprint features can further improve discrimination between automated and legitimate sessions [9]. Nevertheless, fingerprinting shifts part of detection to client-side data collection and may depend on attributes altered by browser updates or privacy controls. Server-log analysis is easier to deploy across heterogeneous clients, but often provides weaker evidence for sophisticated browser-based crawlers.

2.2. Statistical Learning and Traffic Classification

Conventional machine-learning methods improve on fixed rules by learning nonlinear combinations of HTTP and session features. Deep HTTP-traffic classifiers have shown that learned representations can retain discriminative patterns while maintaining practical throughput [10]. Behavioural traffic analysis likewise indicates that temporal regularity and protocol-level characteristics can expose automated activity that isolated signatures miss [11].

However, feature-based models often aggregate a complete session into one fixed vector, discarding request order and local bursts. Broader traffic-classification research also identifies sensitivity to dataset selection, feature availability, encryption, and distribution shift [12]. Therefore, strong offline accuracy on one source does not ensure stable deployment performance. Large Web-activity datasets now provide hundreds of millions of requests and long-term user-agent histories [13], but scale alone does not solve ambiguous labeling or guarantee coverage of human-like crawlers.

2.3. Sequential and Context-Aware Modelling

Recent studies of HTTP request-response cycles show that sequential interactions contain structural patterns useful for behavior and security analysis [14]. Sequence-aware models preserve inter-request dependencies that aggregated statistics can lose. Semi-supervised HTTP anomaly detection further shows that contextual representations and reconstruction scoring can reduce dependence on extensive attack labels while supporting feature-level explanations [15].

The trade-off is computational. Rich sequence encoders generally require more memory and inference operations than rules or tree models, and many evaluations emphasize predictive accuracy without examining performance under increasing request rates or queue pressure. Moreover, anomaly detection and crawler detection are not equivalent: an unusual request is not necessarily crawler-generated, while a well-behaved crawler may issue individually normal requests. Thus, directly transferring anomaly detectors leaves the automated-client identification problem only partially addressed.

2.4. Research Gap and Position of This Study

The literature reveals a consistent trade-off. Rule and fingerprint methods are fast but can be brittle or collection-intensive; statistical models are efficient but compress temporal structure; sequential models preserve behavioral dynamics at higher runtime cost. More importantly, these approaches usually treat the classification threshold as fixed and evaluate confidence, robustness, and workload separately.

LAICD addresses this gap by combining server-side session statistics with a lightweight temporal encoder, using adaptive fusion rather than replacing efficient log features, calibrating crawler probabilities before decision-making, and adjusting the threshold within predefined bounds according to request rate and queue occupancy. The design is evaluated through baseline comparison, ablation, cross-phase generalization, calibration, and increasing-load replay. Its contribution is therefore an integrated and

testable connection between behavioral representation, confidence quality, and real-time operating conditions rather than a larger detection network alone.

3. Methodology

3.1. Overall Architecture

This study proposes a Load-Aware Integrated Crawler Detector (LAICD) for real-time identification of automated Web crawlers under variable traffic intensity. Rather than treating crawler detection as a single classification task, LAICD separates representation learning, probability estimation, and runtime decision control. This design is intended to preserve discriminative temporal information while preventing temporary workload changes from directly distorting crawler probabilities.

As illustrated in Figure 1, the framework contains seven functional stages: request synchronization and session buffering, server-side feature extraction, statistical session encoding, temporal behavior encoding, adaptive multi-view fusion, confidence-calibrated classification, and load-aware decision control.

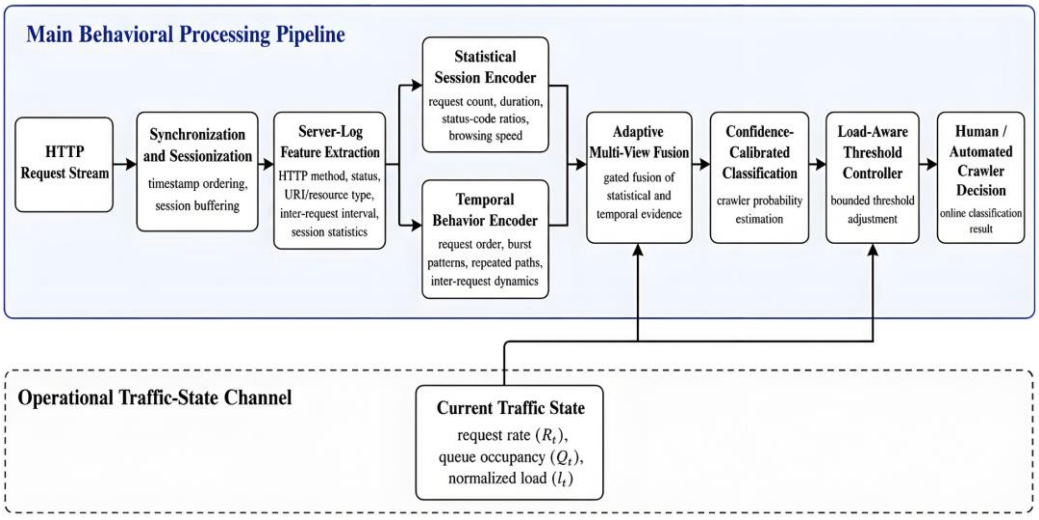

Figure 1. Overall Architecture of the Proposed Laicd Framework

A parallel traffic-state channel supplies current request rate and queue occupancy to the fusion and control modules. The model therefore distinguishes behavioral evidence from operational evidence: behavioral features provide the primary basis for crawler probability estimation, while operational load weakly conditions feature fusion and directly adjusts how the resulting probability is converted into an online decision.

3.2. Request Synchronization and Session Construction

Incoming HTTP records are first sorted by server timestamp. Each request contains an anonymized client identifier, timestamp, HTTP method, requested resource, response status, user-agent category, and derived timing attributes. Requests are grouped into sessions using an inactivity timeout τ_s .

For request r_i , session membership is defined as

$$s_i = \{s_{i-1}, \quad c_i = c_{i-1} \wedge t_i - t_{i-1} \leq \tau_s,\ s_{i-1} + 1, \quad \text{otherwise}, \qquad (1)$$

where s_i is the assigned session index, c_i is the anonymized client identifier, and t_i is the request timestamp. The same timeout is applied to training, validation, and testing data to prevent distribution-dependent session construction.

To synchronize system load with each session, a normalized load variable is computed as

$$\ell_t = \alpha \frac{R_t}{R_{\text{ref}}} + (1 - \alpha) \frac{Q_t}{Q_{\max}}, \qquad (2)$$

where R_t denotes request rate within the current monitoring window, R_{ref} is a reference throughput estimated from training traffic, Q_t is server-side queue occupancy,

$Q_{\max}$ is queue capacity, and $\alpha \in [0,1]$ controls the contribution of traffic rate and queue pressure.

3.3. Server-Side Feature Representation

LAICD uses only features derivable from ordinary server-side traffic records. Continuous attributes include inter-request interval, path depth, request count, browsing speed, status-code ratios, resource-type ratios, and session duration. Categorical attributes include request method, coarse resource category, and response-status class.

For request i, the unified representation is

$$\mathbf{e}_i = \phi\left(W_x \mathbf{x}_i + \sum_{j=1}^{J} E_j\left(c_{ij}\right) + \mathbf{b}_x\right), \quad (3)$$

where $\mathbf{x}_i \in \mathbb{R}^{d_x}$ contains normalized continuous features, c_{ij} is categorical variable j, $E_j(\cdot)$ is its trainable embedding function, W_x is a projection matrix, $\mathbf{b}_x$ is a bias vector, and $\phi(\cdot)$ denotes GELU activation.

Numerical normalization parameters are estimated only from the training partition. Unknown categorical values are assigned to a reserved token during validation and testing.

3.4. Lightweight Temporal Behavior Encoder

A crawler may reproduce plausible headers while maintaining repetitive timing or resource-transition patterns. LAICD therefore retains request order through a lightweight temporal convolutional network rather than relying exclusively on aggregated session statistics.

For a session containing m_s requests,

$$\mathbf{h}_s = \mathrm{Pool}\left[\mathrm{TCN}\left(\left[\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{m_s}\right]; \Theta_T\right)\right], \quad (4)$$

where Θ_T represents temporal-convolution parameters and (\operatorname{Pool}(\cdot)) denotes masked mean pooling over valid requests.

The TCN uses three one-dimensional convolution blocks with kernel size 3 and dilation factors 1, 2, and 4. Residual connections are retained between blocks. This configuration enlarges the effective temporal receptive field without introducing the quadratic sequence cost associated with self-attention. The resulting representation captures short bursts, repeated paths, and irregular inter-request spacing.

3.5. Adaptive Multi-View Fusion

The temporal representation is complementary to aggregated session statistics rather than a replacement for them. Let $\mathbf{v}_s$ denote the projected statistical representation and $\mathbf{h}_s$ the temporal representation. Both vectors are transformed to the same hidden dimension.

Their contribution is controlled by

$$\mathbf{g}_s = \sigma\left(W_g[\mathbf{v}_s; \mathbf{h}_s; \ell_s] + \mathbf{b}_g\right), \mathbf{z}_s = \mathbf{g}_s \odot \mathbf{h}_s + (1 - \mathbf{g}_s) \odot \mathbf{v}_s, \quad (5)$$

where $\mathbf{g}_s$ is a learned element-wise gate, ℓ_s is the synchronized load state, W_g and $\mathbf{b}_g$ are trainable parameters, $\sigma(\cdot)$ is the sigmoid function, and $\odot$ denotes element-wise multiplication.

This mechanism allows the model to increase reliance on temporal evidence when aggregate statistics are ambiguous while preserving efficient statistical features when they are sufficient.

3.6. Confidence-Calibrated Detection Head

Each session receives a binary label $y_s \in 0,1$, where 0 represents human traffic and 1 represents automated crawler traffic. Moderate and advanced bot sessions are therefore merged into the positive class for the primary detection task.

The fused representation is mapped to a calibrated probability as

$$p_s = \sigma\left(\frac{\mathbf{w}_o^{\top} \mathbf{z}_s + b_o}{T}\right), \quad (6)$$

where p_s is the crawler probability, $\mathbf{w}_o$ and b_o are classifier parameters, and $T > 0$ is a temperature parameter estimated on the validation partition after classifier training.

When $T > 1$, overconfident logits are softened; when $T < 1$, probability separation is increased. Calibration therefore modifies confidence interpretation without retraining the feature encoder.

3.7. Bounded Load-Aware Decision Controller

A fixed threshold may produce unstable false-positive behavior when request intensity changes substantially. LAICD therefore adjusts the operating threshold using the synchronized load state but restricts updates to validation-defined bounds.

The threshold at time t is updated by

$$\theta_t = \mathrm{clip}\left[\beta\theta_{t-1} + (1-\beta)\left(\theta_0 + \kappa(\ell_t - \ell_{\mathrm{ref}})\right), \theta_{\min}, \theta_{\max}\right], \quad (7)$$

where θ_0 is the validation-selected base threshold, ℓ_{ref} is reference load, $\beta \in [0,1]$ controls temporal smoothing, κ determines load sensitivity, and $\theta_{\min}$ and $\theta_{\max}$ restrict threshold variation.

A session is classified as automated when its calibrated probability is not lower than θ_t . Because the threshold is clipped and exponentially smoothed, a short traffic spike cannot cause unrestricted decision-boundary changes.

Algorithm 1. Online LAICD Inference

Input: request stream R, trained model Θ, timeout τs, θ0, β, κ, θmin, θmax, Rref, Qmax

Output: calibrated probabilities P, crawler labels Y

Initialize session cache C, threshold θ ← θ0
Initialize output lists P and Y
for each request ri in R do
  synchronize ri according to server timestamp
  assign ri to session using identifier ci and τs
  update request-level and session-level features
  compute current request rate Rt and queue state Qt
  calculate normalized load ℓt
  if the session reaches the decision condition then
    encode statistical representation vs
    encode temporal representation hs
    fuse vs and hs to obtain zs
    compute calibrated crawler probability ps
    update bounded threshold θ and classify ps
    append ps and predicted label to P and Y
return P, Y

3.8. Training Objective and Parameter Optimization

The optimization objective must support both discrimination and confidence reliability. LAICD therefore combines weighted binary classification, calibration regularization, and consistency regularization:

$$\mathcal{L} = -\frac{1}{N}\sum_{s=1}^{N} w_{y_s}\left[y_s \log p_s + (1-y_s)\log(1-p_s)\right] + \lambda_c \frac{1}{N}\sum_{s=1}^{N}(p_s - y_s)^2 + \lambda_r \frac{1}{N}\sum_{s=1}^{N}[p_s(\ell) - p_s(\ell')]^2, \quad (8)$$

where N is the number of training sessions, w_{y_s} is the class weight, λ_c controls probability-calibration regularization, and λ_r controls prediction consistency. The terms $p_s(\ell)$ and $p_s(\ell')$ denote predictions for the same session under original and perturbed load states.

The first term learns the human-versus-crawler decision boundary. The second penalizes poorly aligned confidence estimates. The third prevents modest changes in operational load from unnecessarily changing the behavioral prediction itself. This distinction is important because server load is regularized to exert only a limited influence on behavioral probability estimation and primarily affects the final decision through the bounded threshold controller.

Model parameters are optimized with AdamW using validation PR-AUC for model selection and early stopping. The threshold controller is not optimized through back-propagation; θ_0 , κ , and the threshold bounds are selected from the validation partition under predefined false-positive constraints. Consequently, the proposed architecture integrates three mechanisms that can be independently tested: temporal behavioral representation, confidence calibration, and bounded load-adaptive control.

4. Results and Analysis

4.1. Experimental Setup

Two public Web-traffic datasets were used for complementary evaluation. The Web Bot Detection Dataset served as the supervised classification benchmark. It contains two collection phases involving human users, moderate bots, and advanced human-like bots. Phase 1 includes 50 sessions per class, whereas Phase 2 contains 56 sessions per class, yielding 318 sessions overall. The dataset is publicly downloadable and distributed under the CC BY-NC-SA license. LAICD used only server-side Web-log information. Phase 1 was stratified into 120 training and 30 validation sessions, while all 168 Phase 2 sessions were retained for cross-scenario testing. Human sessions were labeled 0 and both bot categories were merged as label 1.

The EClog dataset was used for high-load runtime evaluation rather than supervised crawler classification. It contains 35,157,691 anonymized HTTP requests collected from a real e-commerce service over 183 days and is publicly available through Harvard Dataverse. The original chronological order was retained and replayed at 1×–5× arrival intensity.

Requests were sorted by timestamp and sessionized using a 30-min inactivity threshold. Numerical variables were standardized from training statistics only, unseen categorical values were mapped to an unknown token, and sequences were limited to 256 requests. Experiments used an AMD Ryzen 9 7900X CPU, 64 GB RAM, and an NVIDIA RTX 4070 12 GB GPU with Python 3.11.9, PyTorch 2.4.1, and CUDA 12.1. The temporal encoder used three 64-channel convolution blocks with dilation factors 1, 2, and 4. AdamW was configured with a learning rate of 3×10^{-4} , batch size 16, and weight decay 1×10^{-4} . Training was limited to 100 epochs with patience 10.

Four baselines were included: Rule-Based Heuristic, Random Forest, XGBoost, and BiLSTM. The primary metrics were Macro-F1, PR-AUC, and FPR@95TPR. All learning experiments were independently repeated five times ($n = 5$); values are reported as mean ± SD. Two-sided paired t -tests were used at $\alpha = 0.05$, with 95% confidence intervals obtained from 1,000 bootstrap resamples.

4.2. Detection Performance

Table 1 compares all methods on the held-out Phase 2 test set.

As shown in Table 1, LAICD achieved a Macro-F1 of 0.914 ± 0.008 and PR-AUC of 0.953 ± 0.006. Compared with BiLSTM, the strongest baseline, the improvements were 0.022 and 0.017, respectively, with significant differences for Macro-F1 ($p = 0.007$) and PR-AUC ($p = 0.012$). FPR@95TPR decreased from 8.8 ± 0.9% to 6.8 ± 0.7% ($p = 0.005$).

The margin over XGBoost and BiLSTM was moderate rather than uniform. Tree-based models remained competitive because request frequency and session statistics already provided useful separation for a substantial portion of automated traffic. The additional improvement of LAICD suggests that temporal evidence, including request spacing and repeated resource transitions, provides complementary information when aggregated session statistics alone are insufficient.

4.3. Ablation and Mechanism Verification

Table 2 evaluates the contribution of the major LAICD components.

Table 2. Ablation Results on the Phase 2 Test Set (Mean ± SD, N = 5).

Variant	Macro-F1	PR-AUC	FPR@95TPR (%)	ECE
Full LAICD	0.914 ± 0.008	0.953 ± 0.006	6.8 ± 0.7	0.031 ± 0.006
w/o Temporal Encoder	0.886 ± 0.013	0.930 ± 0.009	9.6 ± 1.1	0.039 ± 0.007
w/o Adaptive Fusion	0.899 ± 0.010	0.941 ± 0.007	8.1 ± 0.9	0.036 ± 0.006
Fixed Threshold	0.909 ± 0.009	0.949 ± 0.007	8.5 ± 0.8	0.033 ± 0.006
w/o Confidence Calibration	0.911 ± 0.009	0.951 ± 0.007	7.0 ± 0.8	0.071 ± 0.010

Removing the temporal encoder caused the largest Macro-F1 reduction, from 0.914 to 0.886 ($p = 0.004$), confirming that sequential information complemented aggregate traffic statistics. Removing adaptive fusion reduced Macro-F1 by 0.015 ($p = 0.013$), suggesting that fixed weighting could not exploit statistical and temporal evidence equally across sessions.

The fixed-threshold variant retained similar PR-AUC but increased FPR@95TPR to 8.5 ± 0.8% ($p = 0.006$). This supports the interpretation that the controller primarily affects operating behavior rather than representation quality. Removing confidence calibration, including the calibration regularizer and post-hoc temperature scaling, produced only a 0.003 Macro-F1 decrease but increased ECE from 0.031 ± 0.006 to 0.071 ± 0.010 ($p = 0.002$). Thus, calibration mainly improved confidence reliability rather than raw discrimination.

4.4. Convergence Analysis

Figure 2 reports training loss and validation PR-AUC across the five runs.

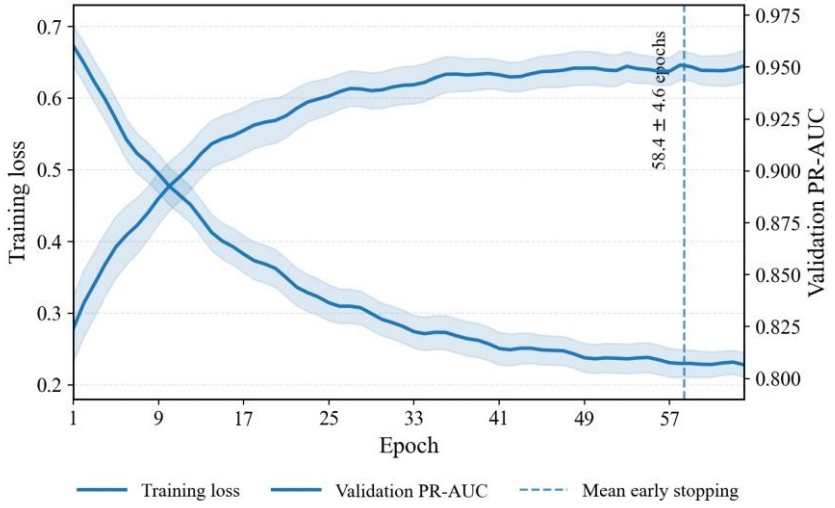

Figure 2. Training Convergence of Laicd Across Five Independent Runs (Mean ± SD, N = 5).

Training loss decreased from 0.673 ± 0.028 at the first epoch to 0.228 ± 0.017 at termination. Validation PR-AUC increased from 0.824 ± 0.016 to a peak of 0.951 ± 0.007. The mean early-stopping point was 58.4 ± 4.6 epochs. After approximately epoch 52, training loss continued to decline slowly while validation PR-AUC fluctuated between 0.947 and 0.953, indicating limited benefit from additional optimization. The relatively narrow SD after epoch 40 also suggests that the five runs reached comparable solutions despite the small labeled training set.

4.5. Stability, Generalization, and Interpretability

High-load behavior was evaluated through EClog replay. As shown in Figure 3, p95 inference latency increased nonlinearly with request intensity.

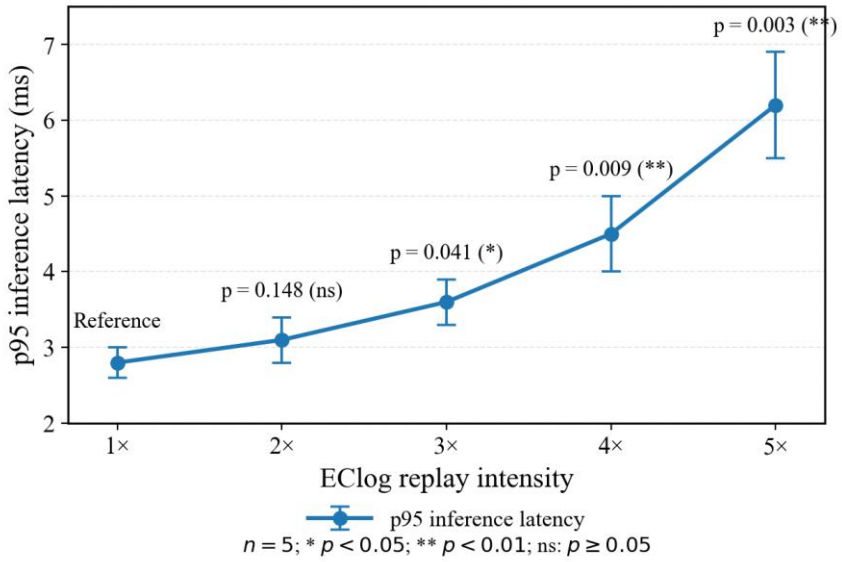

Figure 3. P95 Inference Latency under Increasing EClog Replay Intensity (Mean ± SD, N = 5).

At 1×, 2×, 3×, 4×, and 5× replay intensity, p95 latency was 2.8 ± 0.2, 3.1 ± 0.3, 3.6 ± 0.3, 4.5 ± 0.5, and 6.2 ± 0.7 ms, respectively. The 2× condition was not significantly different from 1× ($p = 0.148$), whereas the first significant increase occurred at 3× ($p = 0.041$). Differences became clearer at 4× ($p = 0.009$) and 5× ($p = 0.003$). Throughput rose from 11.8 ± 0.4 thousand requests/s at 1× to 41.0 ± 1.1 thousand requests/s at 4×, but reached only 43.4 ± 1.5 thousand requests/s at 5×, indicating queue-related saturation rather than constant linear scaling.

Cross-scenario generalization was assessed using the independent Phase 2 data. Macro-F1 decreased from 0.928 ± 0.007 on Phase 1 validation data to 0.914 ± 0.008 on Phase 2, an absolute reduction of 0.014. This modest decline indicates some sensitivity to changes in Web content and participant behavior while retaining most of the validation performance.

SHAP analysis ranked inter-request interval variability, sequential-resource ratio, browsing speed, status-code distribution, and path depth as the five most influential feature groups, contributing 21.7%, 17.9%, 15.8%, 12.6%, and 9.4% of normalized mean absolute importance, respectively. The dominance of temporal variables is consistent with the ablation results: this pattern suggests that longer-term request dynamics provide information that is complementary to individual HTTP attributes.

Model trustworthiness was further supported by the low calibrated ECE, bounded threshold updates, and stable inference up to moderate replay loads. Because LAICD operates on server-side logs and anonymized identifiers rather than raw mouse trajectories or persistent behavioral biometrics, it also reduces additional client-side data collection. This does not establish regulatory compliance by itself, but supports a data-minimization-oriented deployment strategy.

5. Conclusion

This study evaluated LAICD as a server-side crawler detection framework that integrates multi-view traffic representation, bounded load-aware decision control, and confidence calibration. The results correspond directly to the three objectives defined in the Introduction.

First, combining session-level statistics with temporal request representations improved cross-scenario crawler detection. On the held-out Phase 2 dataset, LAICD achieved a Macro-F1 of 0.914 ± 0.008 and a PR-AUC of 0.953 ± 0.006, compared with 0.892 ± 0.009 and 0.936 ± 0.008 for the strongest BiLSTM baseline. FPR@95TPR was reduced from 8.8 ± 0.9% to 6.8 ± 0.7%. The temporal-encoder ablation decreased Macro-F1 to 0.886

± 0.013, indicating that request ordering and inter-request dynamics contributed information not captured by aggregated Web-log statistics alone.

Second, the bounded load-aware controller improved operating stability without materially changing offline ranking performance. Replacing it with a fixed threshold increased FPR@95TPR to 8.5 ± 0.8%. Under EClog replay, p95 inference latency increased from 2.8 ± 0.2 ms at 1× load to 6.2 ± 0.7 ms at 5×, while throughput began to saturate beyond 4× intensity. These results indicate that the controller is most useful for maintaining decision behavior as server pressure changes rather than for increasing classification accuracy itself.

Third, confidence calibration improved probability reliability. Removing the calibration term increased ECE from 0.031 ± 0.006 to 0.071 ± 0.010, while Macro-F1 changed only slightly, supporting the distinction between classification quality and confidence quality.

Several limitations remain. The supervised dataset contains only 318 sessions, and cross-scenario testing is limited to two collection phases. EClog provides realistic traffic volume but not equivalent crawler labels. LAICD also introduces more computation than rule-based filtering, and its behavior under long-term crawler evolution was not evaluated.

Future work should therefore examine larger independently labeled Web datasets, continual adaptation to behavior drift, production deployment under real request queues, and selective verification for low-confidence sessions. These extensions would further clarify the trade-offs among detection accuracy, false-positive control, calibration, and runtime cost.

References

1. R. R. Jagat, D. S. Sisodia, and P. Singh, "Exploiting web content semantic features to detect web robots from weblogs," *Journal of Network and Computer Applications*, vol. 230, p. 103975, 2024.
2. L. Benova and L. Hudec, "Comprehensive analysis and evaluation of anomalous user activity in web server logs," *Sensors*, vol. 24, no. 3, p. 746, 2024.
3. J. Kadel, A. See, R. Sinha, and M. Fischer, "BOTracle: A framework for discriminating bots and humans," in *European Symposium on Research in Computer Security*, Cham, Switzerland: Springer Nature Switzerland, Sep. 2024, pp. 49–67.
4. K. Yasuhara, N. Kodama, and T. Saito, "Challenges in web bot detection and detection evasion technologies," in *International Conference on Network-Based Information Systems*, Cham, Switzerland: Springer Nature Switzerland, Sep. 2024, pp. 162–173.
5. J. Zhang, S. Pan, D. Han, Z. Wang, L. Yao, and Y. Lu, "Session2vec: Session Modeling with Multi-Instance Learning for Accurate Malicious Web Robot Detection," *Electronics*, vol. 14, no. 10, p. 1945, 2025.
6. J. M. Llamas, K. Vranckaert, D. Preuveneers, and W. Joosen, "Balancing security and privacy: Web bot detection, privacy challenges, and regulatory compliance under the GDPR and AI act," *Open Research Europe*, vol. 5, p. 76, 2025.
7. M. Gajewski, O. Hryniewicz, A. Jastrzębska, M. Kozakiewicz, K. Opara, J. W. Owsiński, et al., "Data-driven human and bot recognition from web activity logs based on hybrid learning techniques," *Digital Communications and Networks*, vol. 10, no. 4, pp. 1178–1188, 2024.
8. M. Hölbl, V. Zadorozhny, T. Welzer Družovec, M. Komapara, and L. Nemec Zlatolas, "Browser fingerprinting: overview and open challenges," in *Information Modelling and Knowledge Bases XXXV*, 2024, pp. 297–307.
9. G. S. Pathirage and K. Manathunga, "Machine Learning and Browser Fingerprinting Based Approach for Web Bot Detection," in *2024 6th International Conference on Advancements in Computing (ICAC)*, IEEE, Dec. 2024, pp. 175–180.
10. T. Chen, X. Qiu, Z. Weng, T. Zhu, M. Lv, and K. Sun, "Httpsmell: a deep learning approach on malicious Http traffic detection via data augmentation and label refactoring," *Security and Communication Networks*, vol. 2024, no. 1, p. 6964759, 2024.
11. A. H. H. Kabla, A. H. Thamrin, M. Anbar, S. Manickam, and S. Karuppayah, "Peer-to-peer botnets: exploring behavioural characteristics and machine/deep learning-based detection," *EURASIP Journal on Information Security*, vol. 2024, no. 1, p. 20, 2024.
12. A. Azab, M. Khasawneh, S. Alrabaee, K. K. R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digital Communications and Networks*, vol. 10, no. 3, pp. 676–692, 2024.
13. G. Lucz and B. Forstner, "Aggregate web activity dataset for user-agent behavior classification," *Data in Brief*, vol. 59, p. 111297, 2025.
14. A. Alarood, "Machine Learning-Based Prediction in HTTP Request–Response Cycles: Impacts on Webpage Quality Metrics," *Expert Systems*, vol. 42, no. 8, p. e70085, 2025.
15. M. I. Wasundara, J. Zhou, Y. Yang, D. Zhao, and J. Xiang, "Semi-supervised method for anomaly detection in HTTP traffic," *EURASIP Journal on Information Security*, vol. 2025, no. 1, p. 25, 2025.