XGBoost Web Bot Detection Using Session-Level Features from HTTP Access Logs

Muhammad Imam Rafi'[1)], Danny Oka Ratmana[2)*]

[1,2] Department of Informatics Engineering, Faculty of Intelligent Technology, Dian Nuswantoro University
* Corresponding author : rdannyoka@dsn.dinus.ac.id

Abstract

Automated web traffic includes malicious programs that carry out denial-of-service attacks, data scraping, and credential stuffing, so separating bot sessions from human sessions is useful for protecting web servers. This study examined how well session-level features computed only from web server access logs, without biometric data or client network addresses, could separate bot sessions from human sessions. Raw logs from a public benchmark dataset were parsed into 833 sessions (409 human and 424 bot), and 24 features describing request timing, endpoint access, response behavior, and User-Agent characteristics were extracted. Random forest and extreme gradient boosting (XGBoost) classifiers were trained with fixed hyperparameters and without resampling, using a stratified 80:20 train-test split and five-fold stratified cross-validation. On the test set of 167 sessions, XGBoost reached an accuracy of 94.61%, a harmonic mean of precision and recall of 94.74%, and an area under the receiver operating characteristic curve of 97.58%, while random forest reached 94.61%, 94.67%, and 97.36%. The two models gave different predictions for only two test sessions, and cross-validation accuracy was 94.60% for XGBoost and 95.08% for random forest. Features derived from the User-Agent string had the highest gain-based importance in XGBoost. The XGBoost model was also implemented as a prototype with a backend service and a frontend dashboard, and five black-box test scenarios passed. These results come from a single controlled benchmark, were obtained by classifying complete sessions offline, and require validation on other data.

Keywords- Web Bot Detection, Access Log, Session Features, XGBoost, Random Forest

1. INTRODUCTION

Automated programs account for a share of web traffic. Some serve legitimate purposes such as search engine indexing, but others carry out malicious activities, including distributed denial-of-service attacks, data scraping, and credential stuffing [1], often through botnets that flood servers with requests [2]. Identifying such sessions helps limit the server resources they consume. Modern bots use headless browsers, address rotation, and behavior imitation to resemble human visitors [3], which makes them difficult to detect.

Several approaches rely on behavioral biometrics such as mouse movements [4], but collecting such signals raises privacy concerns under regulations such as the General Data Protection Regulation and the European Union Artificial Intelligence Act [5]. Web server access logs are recorded by default and offer a less intrusive source, although they can still contain personal data such as client addresses [5]. Therefore, this study utilizes only request-level and session-level attributes, excluding both biometric data and client IP addresses. The benchmark employed here does not include client addresses.

Previous studies have examined bot recognition from web logs in several ways. Rovetta et al. [6] classified sessions from a real online bookstore using clustering followed by supervised labelling of clusters, and compared it with a multi-layer perceptron and a support vector machine. Gajewski et al. [7] engineered features from publisher web logs, including attributes derived from IP addresses, fingerprints, and client-side scripts, and applied a rule-based detector and a hierarchical clustering-and-classification model with emphasis on interpretability. Iliou et al. [8] created the benchmark used in this study [9] and proposed a framework that combines a web-log module with a mouse-movement module, reporting that the mouse-movement module was more effective than the web-log module for moderate and advanced bots. These studies rely on traffic from individual commercial sites [7], on additional identifiers or client-side signals [7], or on mouse behavioral data [8]. It therefore remains to be examined how far session-level features from access logs alone can separate bot and human sessions on this public benchmark.

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

Tree-ensemble classifiers such as Random Forest and XGBoost have been applied to tabular classification tasks, and XGBoost offers efficient training, parallel computation, and built-in regularization [10]. Random Forest has been compared with XGBoost in related security tasks [11], [12], [13]. In addition, XGBoost has been applied to intrusion detection in IoT and industrial networks [14]. This study therefore evaluates both models on the same data and treats Random Forest as a comparison baseline. The parsed sessions used here (409 human and 424 bot) are nearly balanced, so no resampling was applied. Because a trained model does not by itself parse newly collected logs, the model was also implemented as a prototype system for uploading logs and viewing predictions [15].

This study aims to examine (1) how well session-level features derived only from HTTP access logs distinguish bot sessions from human sessions on this benchmark, (2) how Random Forest and XGBoost compare on this task, and (3) which features and which sessions most influence correct and incorrect decisions.

The contributions of this study are as follows. First, it develops a session-level feature extraction pipeline that converts raw HTTP access logs into 24 features without biometric data or client IP addresses, and evaluates Random Forest and XGBoost on 833 sessions using a stratified train-test split and five-fold cross-validation with fixed hyperparameters. Second, it implements the resulting XGBoost model as a prototype system with a backend service for log parsing, feature extraction, and inference, and a frontend dashboard for uploading logs and viewing predictions. The evaluation uses a single controlled benchmark and complete sessions classified offline, so the findings are not evidence of performance on other sites, bot types, or log formats.

2. METHODS AND DATA

This study aims to design and evaluate a web bot detection model that relies solely on non-invasive, session-level features derived from HTTP access logs, and to demonstrate the model through a prototype software implementation. To achieve this, the research followed a structured pipeline consisting of data source, log parsing, feature engineering, model development, model evaluation, feature importance and error analysis, and system implementation, as illustrated in Figure 1 [16]. Because the benchmark classes were nearly balanced, no resampling stage was included. Each stage is described in the following subsections.

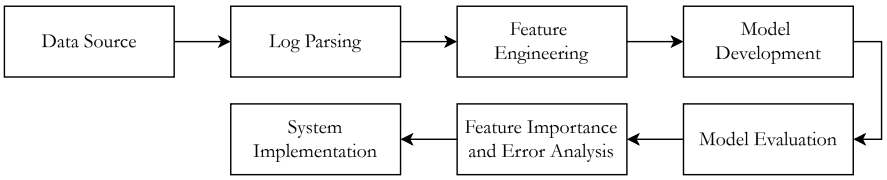

Figure 1. Flowchart of the Research Methodology

2.1 Data Source and Log Parsing

The data came from the public web bot detection benchmark created by Iliou et al. [8] and distributed by the M4D Group [9]. The benchmark consists of web server access logs generated in a controlled setting and collected in two phases, with human and bot traffic stored in separate files. Phase 1 provided five human log files and two bot log files (moderate and advanced bots), and phase 2 provided 47 human log files and one combined log file of moderate and advanced bots. Each session was labeled by its file of origin (human or bot), and all sessions came from this single benchmark, with no data from other sources. Each log line followed a combined-log-style format extended with a session identifier. Lines were tokenized with shell-style splitting, and nine fields were extracted: timestamp, request method, endpoint, protocol version, status code, response size, referrer, session identifier, and User-Agent string. The client address field was not used because the benchmark contains no client addresses. Lines with

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

fewer than nine tokens or a malformed request field were discarded, which left 305,386 parsed records (214,189 human and 91,197 bot).

A session identifier was unique only within one log file, so a session key was formed from the file name and the identifier to avoid merging unrelated sessions from different files. Records without a session identifier or with an unparsable timestamp were removed, leaving 298,472 records, and no session key contained records of both classes. Sessions were delimited only by the identifier supplied in the dataset, with no additional inactivity threshold. This produced 833 sessions (409 human and 424 bot), as shown in Table 1. The classes were nearly balanced (49.1% and 50.9%), so no resampling was applied. The train-test partition is described in Section 2.4.

Table 1. Class distribution of the full dataset and the train-test split

Split	Total Sessions	Human Sessions	Bot Sessions
Full dataset	833	409	424
Training set (80%)	666	327	339
Testing set (20%)	167	82	85

2.2 Feature Engineering

Each session was converted into one vector of 24 features computed only from the parsed log fields. The features fall into four groups: request timing, endpoint and request characteristics, response behavior, and User-Agent and protocol characteristics (Table 2). Per-request indicators were first derived from each record and then aggregated within each session. Binary indicators were aggregated as the proportion of requests in which they were true, while numeric attributes were aggregated as counts, means, or standard deviations [17]. The client address, session identifier, source file name, and collection phase were not used as features. Feature representation also plays an important role in cybersecurity detection, as models can learn patterns from network traffic features to distinguish benign behavior from malicious activity [18].

Table 2. Categorization of session-level features

Feature	Definition	Type
total_requests	Total number of requests within a session	Numeric (integer)
requests_per_sec	Number of requests divided by session duration (Eq. 1)	Numeric (continuous)
mean_interval_sec	Mean time interval between consecutive requests	Numeric (continuous)
std_interval_sec	Standard deviation of inter-request time intervals (Eq. 2)	Numeric (continuous)
unique_endpoint_ratio	Ratio of unique endpoints accessed to total requests in the session	Numeric (0–1)
pct_write	Proportion of requests using write-type HTTP methods (POST, PUT, DELETE, PATCH)	Numeric (0–1)
pct_2xx	Proportion of requests with a 2xx (success) response status	Numeric (0–1)
pct_4xx	Proportion of requests with a 4xx (client error) response status	Numeric (0–1)

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

Feature	Definition	Type
pct_5xx	Proportion of requests with a 5xx (server error) response status	Numeric (0–1)
pct_has_referrer	Proportion of requests that include a non-empty referrer header	Numeric (0–1)
unique_endpoints	Count of distinct endpoints accessed within the session	Numeric (integer)
mean_endpoint_depth	Average path depth of accessed endpoints (number of "/" segments)	Numeric (continuous)
pct_storage_req	Proportion of requests targeting storage-related paths (containing "/storage/")	Numeric (0–1)
pct_admin_req	Proportion of requests targeting admin/authentication-related paths (containing "admin", "register", or "login")	Numeric (0–1)
ua_is_bot_kw	Indicator of whether the session's User-Agent string contains a known bot/script keyword (e.g., bot, crawler, spider, curl, wget, python)	Categorical (binary: 0/1)
ua_length	Character length of the User-Agent string	Numeric (integer)
ua_unique_chars	Number of distinct characters in the User-Agent string	Numeric (integer)
ua_browser_enc	Encoded browser family detected from the User-Agent string (e.g., Chrome, Firefox, Safari, Edge, Opera, MSIE, Trident, or "other")	Categorical (encoded)
ua_os_enc	Encoded operating system detected from the User-Agent string (e.g., Windows, Linux, Android, Mac, iPhone, iPad, or "other")	Categorical (encoded)
http_ver	HTTP protocol version used by the first request in the session	Numeric (continuous)
mean_response_size	Average response payload size (bytes) across requests in the session	Numeric (continuous)
std_response_size	Standard deviation of response payload size across requests in the session	Numeric (continuous)
pct_odd_hour	Proportion of requests occurring during unusual hours (00:00–06:00)	Numeric (0–1)
dominant_hour	Most frequently occurring hour-of-day among the session's requests	Numeric (integer, 0–23)

Two categorical attributes, the browser family and the operating system family, were extracted from the lowercased User-Agent string by sequential keyword matching and encoded with fixed ordinal dictionaries that were defined in advance and not fitted to the data. The browser keywords, in matching order, were chrome, firefox, safari, edge, opera, msie, and trident, and the operating system keywords were windows, linux, android, mac, iphone, and ipad. The first keyword found in this order determined the category, and strings with no match were assigned to an additional “other” category. Browser codes ranged from 0 to 7 and operating system codes from 0 to 6. The bot-keyword indicator was set when the User-Agent string contained any of bot, crawler, spider, scraper, curl, wget, python, java/, go-http, or libwww. The User-Agent length, the number of distinct characters, the browser code, the operating system code, and the protocol version were taken from the first request of the session. The same dictionaries were stored with the trained models so that new logs could be encoded consistently at inference time. Non-numeric status codes and response sizes were set to 0, a missing protocol version was set to 1.0, a missing hour was set to 12, and any remaining missing session-level value was set to 0.

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

For a session with n requests at timestamps $t_1 \leq \ldots \leq t_n$, the request rate was defined as

$$r = \begin{cases} \frac{n}{t_n - t_1}, t_n > t_1 \\ n, t_n = t_1 \end{cases} \quad (1)$$

where the second case applies when all requests fall within the same second. With $m = n - 1$ intervals $\Delta_i = t_{i+1} - t_i$ between consecutive requests and their mean $\bar{\Delta}$ (set to 0 when $m = 0$), the standard deviation of the intervals was defined as

$$\sigma_\Delta = \begin{cases} \sqrt{\frac{\sum_{i=1}^{m}(\Delta_i - \bar{\Delta})^2}{m-1}}, m > 1 \\ 0, m \leq 1 \end{cases} \quad (2)$$

2.3 Model Development

Two tree-ensemble classifiers were trained on the 24 session-level features: XGBoost as the primary model and Random Forest as a comparison baseline. Random Forest builds many decision trees on bootstrap samples of the training data, lets each split consider a random subset of the features, and averages the class probabilities predicted by the trees [19]. XGBoost builds trees sequentially, with each new tree fitted to correct the errors of the ensemble built so far, and it includes regularization and efficient parallel training [10]. Random Forest has been compared with XGBoost in related security tasks, which motivated evaluating both models on the same data. Both models operate directly on the tabular features, so no feature scaling was applied.

The hyperparameters in Table 3 were fixed before training and were not tuned on the test set or any validation data, and parameters marked N/A do not apply to the corresponding model. All other parameters were left at the library defaults, and no resampling or class weighting was applied because the classes were nearly balanced (Section 2.1). XGBoost used logloss as its evaluation metric without early stopping, so the number of boosting rounds was fixed at 300. Both models were fitted on the 666 training sessions without using the test set for training or hyperparameter selection, and the same configurations were refitted on each training fold in the cross-validation (Section 2.4). A random seed of 42 was used for the models, the train-test split, and the cross-validation folds. The models were implemented in Python with scikit-learn and XGBoost (version 3.4.1) in Google Colab.

Table 3. Fixed hyperparameters of the two models

Hyperparameter	Random Forest	XGBoost
n_estimators	200	300
max_depth	None (unlimited)	6
min_samples_split	5	N/A
min_samples_leaf	2	N/A
learning_rate	N/A	0.05
subsample	N/A	0.8
colsample_bytree	N/A	0.8
eval_metric	N/A	logloss
random_state	42	42

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

2.4 Evaluation Protocol

The 833 sessions were divided into training and test sets with a stratified 80:20 split at the session level (random seed 42), which gave 666 training sessions and 167 test sessions with the same class proportions (Table 1). Both models were fitted on the training set and evaluated once on the test set. Because a single split of 167 sessions can be sensitive to which sessions fall in the test set, a five-fold stratified cross-validation was also performed on all 833 sessions, with shuffling and a random seed of 42. In each fold, the models were refitted with the same fixed hyperparameters (Table 3), and the mean and standard deviation of each metric across the five folds were reported. Since the cross-validation used all sessions, including those in the test set, it served as a supplementary check of stability and not as an independent test.

Bot was treated as the positive class, and class labels were assigned with the default decision threshold of 0.5. Performance was measured with a confusion matrix and with accuracy, precision, recall, F1-score, and ROC-AUC. Accuracy, the proportion of all sessions classified correctly, was defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \qquad (3)$$

where TP, TN, FP, and FN are the numbers of true positive, true negative, false positive, and false negative predictions. Precision, the proportion of sessions predicted as bots that were truly bots, was defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad (4)$$

Recall, the proportion of actual bot sessions that were detected, was defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad (5)$$

F1-score, the harmonic mean of precision and recall, was defined as

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (6)$$

The ROC curve plots the true positive rate against the false positive rate across all decision thresholds, using the predicted probability of the bot class. ROC-AUC is the area under this curve and summarizes how well a model ranks bot sessions above human sessions, independent of a fixed threshold. All metrics are reported as percentages with two decimal places. The two models were compared descriptively, and no statistical significance test was performed.

2.5 Feature Importance and Error Analysis

Feature importance was computed from the models fitted on the training set. For Random Forest, impurity-based importance (mean decrease in impurity) was used, which sums the reduction in node impurity produced by the splits on each feature across all trees. For XGBoost, gain-based importance was used (the importance type was set explicitly to gain), which is the average improvement in the training loss obtained when a feature is used in a split. Both sets of values were normalized so that the importance of all 24 features summed to 1, and the features were ranked within each model. Because the two measures are defined differently, their absolute values are not directly comparable. The importance values describe how much each fitted model relied on a feature, they do not indicate a causal effect, and they were not used to select features or to retrain the models.

The error analysis used only the test set and the predictions of the models already trained, so no retraining and no additional data were involved. Test sessions were grouped by class and, for bots, by source log

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

file: human sessions, phase 1 advanced bots, phase 1 moderate bots, and phase 2 bots, whose moderate and advanced sessions are stored in one file and cannot be separated by bot type. For each model, the number and percentage of misclassified sessions in each group were calculated. In addition, the mean values of six features (total number of requests, request rate, standard deviation of the request interval, proportion of requests with a referrer, number of unique endpoints, and the bot-keyword indicator) were compared between bot sessions classified as human (false negatives) and bot sessions classified correctly (true positives). The sessions misclassified by at least one model were also listed with their group, true label, and predictions. Because each group in the test set contains few sessions, the results of this analysis are indicative and not statistical evidence.

2.6 System Implementation and Testing

The trained XGBoost model, together with its categorical encoding scheme and expected feature order, was exported and integrated into a two-tier prototype system comprising a backend service and a frontend interface [16]. The backend was built with FastAPI [20], [21] and performed log parsing, feature engineering, and session classification. The frontend was built with React [22] and allowed users to upload log files and optional ground truth data and to view the resulting classification and evaluation metrics, as illustrated in Figure 2. The system classified uploaded log files in batch and did not perform real-time classification.

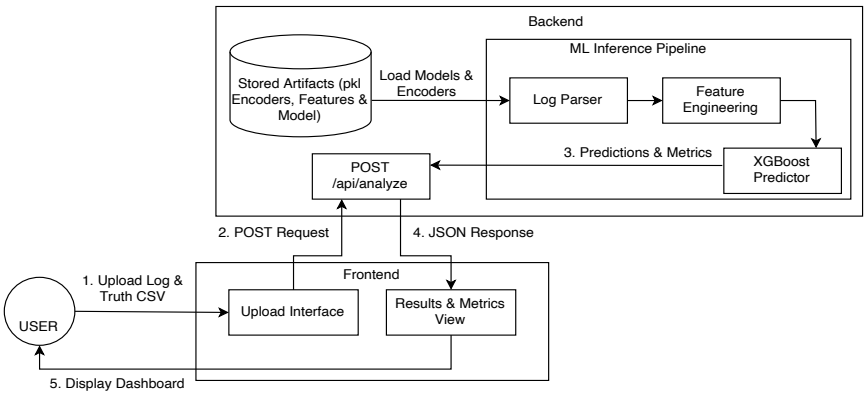

Figure 2. Data Flow Diagram of the Web Bot Detection System

The prototype was tested using black-box functional testing, in which the behavior of the system was compared with the expected results without inspecting its code [23]. Five scenarios were defined: uploading a valid log without validation mode, uploading a malformed log, uploading a valid log with validation mode enabled, calling the analysis endpoint (POST /api/analyze) directly, and uploading an empty file or selecting no file. A scenario was marked as passed if the observed behavior matched the expected result [24], [25]. The test results are reported in Section 3.3.

3. RESULTS AND DISCUSSION

This section reports the evaluation results of the XGBoost and Random Forest models on the test set and in cross-validation, followed by the feature importance and error analysis, the functional testing of the prototype system, and the limitations of the study. Each result is accompanied by its interpretation, and the findings are interpreted within the scope of the single benchmark dataset used in this study.

3.1 Model Performance and Cross-Validation

On the test set of 167 sessions (82 human and 85 bot), both models reached an accuracy of 94.61% (Table 4). Random Forest had a higher precision (95.24% versus 94.19%), whereas XGBoost had a higher recall (95.29% versus 94.12%), and the F1-scores were 94.67% and 94.74%, respectively. Each model misclassified nine sessions, with four false positives and five false negatives for Random Forest

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

and five false positives and four false negatives for XGBoost (Table 5), and the two models gave different predictions for only two test sessions. A single session shifts the recall by about 1.2 percentage points in a test set of this size, so these differences are small, and the results do not show that either model performed better.

Table 4. Performance of the two models on the test set and in cross-validation

Evaluation	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Test set (80:20)	Random Forest	94.61%	95.24%	94.12%	94.67%	97.36%
	XGBoost	94.61%	94.19%	95.29%	94.74%	97.58%
5-fold CV (mean ± SD)	Random Forest	95.08% ± 2.12%	96.83% ± 1.97%	93.41% ± 2.93%	95.07% ± 2.13%	98.41% ± 0.78%
	XGBoost	94.60% ± 1.56%	96.13% ± 1.34%	93.17% ± 2.71%	94.60% ± 1.59%	98.45% ± 0.60%

Table 5. Confusion matrices of the two models on the test set

Model	Actual class	Predicted: Human	Predicted: Bot
Random Forest	Human	78	4
	Bot	5	80
XGBoost	Human	77	5
	Bot	4	81

The ROC curves of both models lie close to the upper-left corner (Figure 3), with ROC-AUC values of 97.58% for XGBoost and 97.36% for Random Forest. These values mean that a randomly chosen bot session received a higher bot probability than a randomly chosen human session in roughly 97% of such pairs, regardless of the decision threshold, and the difference of 0.22 percentage points between the models is too small to indicate a consistent advantage.

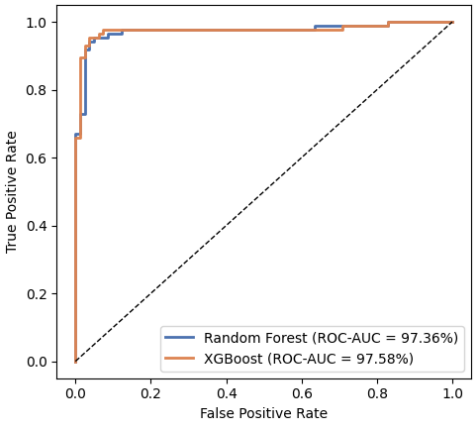

Figure 3. ROC curves of Random Forest and XGBoost on the test set

Five-fold cross-validation on all 833 sessions gave results close to those of the single split (Table 4). Mean accuracy was 94.60% ± 1.56% for XGBoost and 95.08% ± 2.12% for Random Forest, and mean ROC-AUC was 98.45% ± 0.60% and 98.41% ± 0.78%, respectively. The accuracy order was the opposite of that on the test set, and the differences between means were smaller than the standard deviations across folds, so no consistent difference was found. The standard deviation of recall (2.71% to 2.93%) shows that the detection rate varied between folds. Because the cross-validation included the

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

test sessions, it indicates that the results did not depend on a single partition of this dataset, but it is not an independent test.

The two models differed slightly in opposite directions. XGBoost had slightly higher test recall, F1-score, and ROC-AUC, a slightly higher cross-validation ROC-AUC, and smaller standard deviations across folds for all five metrics. Random Forest had slightly higher test precision with fewer false positives (four versus five human sessions), and higher mean cross-validation accuracy, precision, recall, and F1-score. These differences are small and no significance test was performed, so they are treated as descriptive observations, not evidence that one model is better. For the configurations used, the serialized XGBoost model (412 KB) was about one third the size of the Random Forest model (1,261 KB), and smaller models generally require less storage and are easier to deploy [26], [27]. This practical consideration, not predictive performance, favored XGBoost for the prototype system (Section 3.3), while Random Forest remained a comparable alternative on this dataset.

These results are not directly comparable with related studies because the datasets and protocols differ [28]. Rovetta et al. [6] reported accuracies of 98.6% for a multi-layer perceptron and 99.2% for a support vector machine on 13,397 sessions from a real online bookstore with a 50:50 train-test split, which is higher than the accuracy obtained here on a smaller, controlled dataset. On the benchmark used in this study, the web-log module of Iliou et al. [8] reported accuracies of 0.90 for moderate bots and 0.87 for advanced bots, but with a 30-minute session timeout and accuracy tracked over the number of requests, whereas this study classified complete sessions. Gajewski et al. [7] addressed interpretable models for publisher logs with uncertain labels, so their results are not compared numerically. The accuracy obtained here should therefore not be read as better or worse than that of earlier studies, and evaluation on other datasets is needed (Section 3.4).

3.2 Feature Importance and Error Analysis

Figure 4 shows the top 12 features of each model. The feature ua_unique_chars (number of distinct characters in the User-Agent string) ranked first in both models, accounting for 30.79% of the total gain in XGBoost and 13.31% of the mean decrease in impurity in Random Forest. Importance was concentrated in XGBoost, where ua_unique_chars, ua_length, and ua_os_enc together accounted for 68.22% of the total gain, but more evenly spread in Random Forest, where the same three features accounted for 25.23%, alongside dominant_hour (9.00%), unique_endpoints (7.51%), and std_interval_sec (7.30%). Both models ranked unique_endpoints high (third in Random Forest, fourth in XGBoost), whereas time-of-day features diverged: dominant_hour ranked second in Random Forest but seventh in XGBoost, and pct_odd_hour ranked fifth in XGBoost but seventeenth in Random Forest. Because the two importance measures are defined differently, absolute values are not directly comparable.

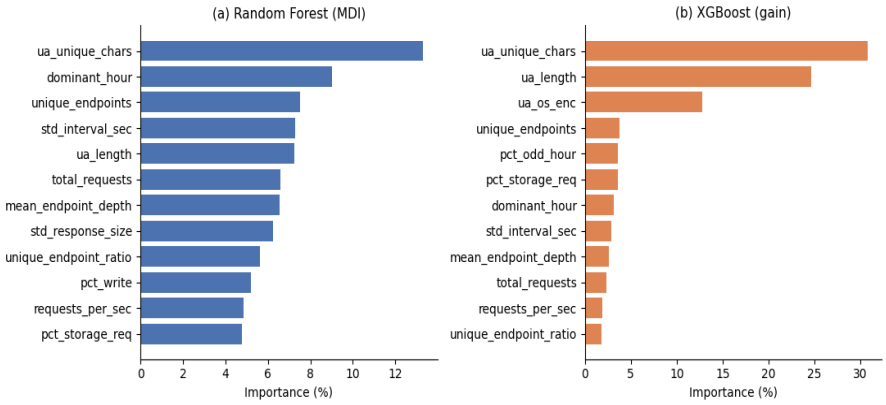

Figure 4. Top 12 Feature Importance of Random Forest and XGBoost

Six features had zero importance in both models: pct_2xx, pct_4xx, pct_5xx, pct_admin_req, ua_is_bot_kw, and http_ver. XGBoost also assigned zero gain to ua_browser_enc and pct_has_referrer,

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

which had 1.00% and 0.28% importance in Random Forest. The bot-keyword indicator was not set for any bot session in the test set, whether detected or missed, so the User-Agent information was used through its length and character composition and not through explicit bot keywords. These importance values describe how the fitted models used the features on this dataset and should not be read as evidence of which features separate bots from humans in general. The prominence of User-Agent and time-of-day features may partly reflect how the benchmark was generated, and both importance measures can favor numeric features with many distinct values and divide importance among correlated features (Section 3.4).

The misclassified test sessions are summarized in Table 6. Each model misclassified nine sessions, and eight of them (four human and four bot sessions) were misclassified by both models. The highest error rates were among the phase 1 bots: one of nine advanced bot sessions (11.11%) for both models, and three (12.00%) and two (8.00%) of 25 moderate bot sessions for Random Forest and XGBoost, respectively. In comparison, one of 51 phase 2 bot sessions (1.96%) and four (4.88%) and five (6.10%) of 82 human sessions were misclassified. Errors were therefore not concentrated among the advanced bots. The lower error rate in phase 2 may be related to the greater similarity among bots in that phase reported by Iliou et al. [8], but this was not tested. Since one session represents more than 10% of the smallest groups, these rates are indicative only.

Table 6. Misclassified test sessions by group and model

Group	Test sessions	Random Forest	XGBoost
Human	82	4 (4.88%)	5 (6.10%)
Bot, advanced (phase 1)	9	1 (11.11%)	1 (11.11%)
Bot, moderate (phase 1)	25	3 (12.00%)	2 (8.00%)
Bot, moderate and advanced (phase 2)	51	1 (1.96%)	1 (1.96%)
Total	167	9 (5.39%)	9 (5.39%)

Compared with the bot sessions that were detected, the missed bot sessions were shorter, issued requests at a higher rate, and visited fewer distinct endpoints. The mean number of requests was 62.80 for the five missed sessions of Random Forest and 67.25 for the four missed sessions of XGBoost, against 217.78 and 215.64 for the detected sessions. The mean request rate was 2.03 and 2.23 requests per second against 1.02 for the detected sessions, and the mean number of unique endpoints was 12.00 and 11.75 against 18.23 and 18.16. These observations are descriptive, are based on four to five sessions per model, and do not indicate that these characteristics caused the errors.

3.3 System Functional Testing

The five black-box scenarios defined in Section 2.6 were run on the prototype system (Figure 5), and all five produced the expected results (Table 7). A valid log uploaded without validation mode was parsed and its session predictions were displayed. A malformed log was rejected with an informative error message and did not crash the system. With validation mode enabled, the evaluation metrics were computed against the ground truth and displayed. A direct call to the POST /api/analyze endpoint returned a response that matched the expected JSON schema, and an empty file or a missing file was rejected with a validation error.

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

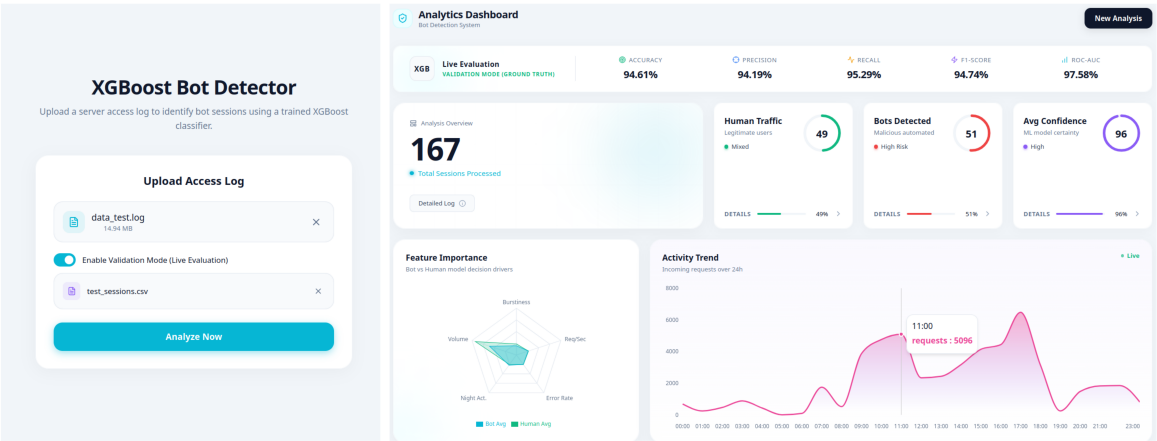

Figure 5. Dashboard Interface of the Prototype System

Table 7. Black-Box Testing Results

Feature/Scenario	Expected Result	Status
Upload valid log without validation mode	The system parses the log and returns the visualized session predictions	Passed
Upload malformed or invalid log	The system rejects the file with an informative error instead of crashing	Passed
Upload valid log with validation mode enabled	The system computes and displays the evaluation metrics against the ground truth	Passed
Direct API call to POST /api/analyze	The response matches the expected JSON schema	Passed
Upload empty file or no file selected	The system blocks or rejects the request with a clear validation error	Passed

These results show that log upload, parsing, feature extraction, inference, validation mode, and the handling of the tested invalid inputs worked as specified in these five scenarios. The testing was functional only. It did not measure response time, throughput, or behavior with large logs or concurrent users, and it did not include security or usability testing. The results therefore demonstrate that the trained XGBoost model can be used through the prototype, but they do not establish readiness for production deployment, and the predictions of the prototype remain subject to the limitations of the model discussed in Sections 3.1 and 3.4.

3.4 Limitations

This study has several limitations related to the data and the evaluation. All sessions came from a single public benchmark generated in a controlled setting, in which the human sessions were produced by the benchmark's authors in phase 1 and by 28 additional participants in phase 2 who browsed a server of mostly text pages [8]. The models were not tested on other websites, bot types, or log formats, so their performance on other datasets was not examined and may be lower than reported here. The test set contained only 167 sessions, and one session shifts the recall by about 1.2 percentage points. The differences between XGBoost and Random Forest are therefore within the range of uncertainty, no significance test was performed, and the group-level error rates are indicative only. The train-test split and the cross-validation folds were random at the session level and were not grouped by log file or participant, and the cross-validation included the test sessions, so the reported results may be optimistic.

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

The hyperparameters were fixed in advance and not tuned, and the sessions were defined only by the session identifier in the dataset without the 30-minute timeout used by Iliou et al. [8].

Several limitations concern the features and the application. The most important features were derived from the User-Agent string and the time of day, which may reflect how the benchmark was generated and not the behavior of bots in general, and the importance measures themselves can favor numeric features with many distinct values [29]. The approach avoids biometric data and client addresses, but it still uses the session identifier and the User-Agent string. The models classified complete sessions offline, so early or real-time detection was not evaluated. The system testing was functional only and did not cover performance, security, or usability. Validation on independent datasets, with splits grouped by source, should be carried out before the findings are applied to other environments.

4. CONCLUSION

This study investigated session-level web bot detection using features derived solely from HTTP access logs, without biometric data or client IP addresses, and compared XGBoost with Random Forest. On a public benchmark dataset of 833 sessions, both models achieved an accuracy of 94.61% on the test set, with nine misclassified sessions out of 167. XGBoost achieved marginally higher F1-score and ROC-AUC values (94.74% and 97.58%, respectively), whereas Random Forest achieved a higher precision (95.24%) and higher mean accuracy, precision, recall, and F1-score in five-fold cross-validation. The confusion matrices revealed a trade-off between false positives and false negatives: Random Forest produced fewer false positives, while XGBoost produced fewer false negatives. Overall, the results indicate comparable classification performance, without clear evidence of the superiority of either model. The XGBoost model was also integrated into a prototype system that passed all five black-box functional test scenarios.

These findings are limited to a single controlled benchmark dataset, a relatively small test set, and offline classification of complete sessions; therefore, generalization to other websites, bot types, and log formats remains unverified. Future work should evaluate the models on independent datasets using source-grouped splits, investigate detection from partial sessions, measure prototype latency and resource consumption under realistic workloads, and assess whether User-Agent-derived features remain informative when bots imitate common browser strings.

ACKNOWLEDGEMENTS

The authors thank the M4D Group and Iliou et al. for making the web bot detection benchmark dataset publicly available. This research did not receive funding from any party.

REFERENCES

[1] H. Owen, J. Zarrin, and S. M. Pour, "A Survey on Botnets, Issues, Threats, Methods, Detection and Prevention," *J. Cybersecurity Priv.*, vol. 2, no. 1, pp. 74–88, 2022. https://doi.org/10.3390/jcp2010006

[2] L. Liu, W. Yu, Z. Wu, and S. Peng, "XGBoost-Based Detection of DDoS Attacks in Named Data Networking," *Future Internet*, vol. 17, no. 5, pp. 206, 2025. https://doi.org/10.3390/fi17050206

[3] G. Jarad and K. Bicakci, "When Handshakes Tell the Truth: Detecting Web Bad Bots via TLS Fingerprints," [preprint] arXiv, 2026. https://doi.org/10.48550/arXiv.2602.09606

[4] S. Sadeghpour and N. Vlajic, "ReMouse Dataset: On the Efficacy of Measuring the Similarity of Human-Generated Trajectories for the Detection of Session-Replay Bots," *J. Cybersecurity Priv.*, vol. 3, no. 1, pp. 95–117, 2023.https://doi.org/10.3390/jcp3010007

[5] 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 Res. Eur.*, vol. 5, pp. 76, 2025. https://doi.org/10.12688/openreseurope.19347.1

Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

[6] S. Rovetta, G. Suchacka, and F. Masulli, "Bot recognition in a Web store: An approach based on unsupervised learning," *J. Netw. Comput. Appl.*, vol. 157, p. 102577, 2020. https://doi.org/10.1016/j.jnca.2020.102577

[7] M. Gajewski *et al.*, "Data-driven human and bot recognition from web activity logs based on hybrid learning techniques," *Digit. Commun. Netw.*, vol. 10, no. 4, pp. 1178–1188, 2024. https://doi.org/10.1016/j.dcan.2023.01.020

[8] C. Iliou, T. Kostoulas, T. Tsikrika, V. Katos, S. Vrochidis, and I. Kompatsiaris, "Detection of Advanced Web Bots by Combining Web Logs with Mouse Behavioural Biometrics," *Digit. Threats Res. Pract.*, vol. 2, no. 3, pp. 1–26, 2021. https://doi.org/10.1145/3447815

[9] M4D Group, ITI-CERTH, "Web Bot Detection Dataset," MultiModal Data Fusion and Analytics Group (M4D). Accessed: 2026. [Online]. https://m4d.iti.gr/web-bot-detection-dataset/

[10] Z. Arif Ali, Z. H. Abduljabbar, H. A. Tahir, A. Bibo Sallow, and S. M. Almufti, "eXtreme Gradient Boosting Algorithm with Machine Learning: a Review," *Acad. J. Nawroz Univ.*, vol. 12, no. 2, pp. 320-334, 2023. https://doi.org/10.25007/ajnu.v12n2a1612

[11] S. T. Hamidou and A. Mehdi, "Enhancing IDS performance through a comparative analysis of Random Forest, XGBoost, and Deep Neural Networks," *Mach. Learn. Appl.*, vol. 22, pp. 100738, 2025. https://doi.org/10.1016/j.mlwa.2025.100738

[12] K. A. Binsaeed and A. M. Hafez, "Enhancing Intrusion Detection Systems with XGBoost Feature Selection and Deep Learning Approaches," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 5, pp. 1084–1098, 2023. https://doi.org/10.14569/IJACSA.2023.01405112

[13] T.-T.-H. Le, Y. E. Oktian, and H. Kim, "XGBoost for Imbalanced Multiclass Classification-Based Industrial Internet of Things Intrusion Detection Systems," *Sustainability*, vol. 14, no. 14, p. 8707, 2022. https://doi.org/10.3390/su14148707

[14] Y. Hu, K. Xiao, L. Luo, and L. Chen, "An XGBoost-Based Intrusion Detection Framework with Interpretability Analysis for IoT Networks," *Appl. Sci.*, vol. 16, no. 2, p. 980, 2026. https://doi.org/10.3390/app16020980

[15] D. O. Ratmana, M. S. Rohman, G. W. Saraswati, F. O. Saputra, A. N. Safitri, and I. Harkespan, "Three-Tier Disaster Logistics System Integrating GIS and MILP Optimization," *J. Appl. Informatics Comput.*, vol. 10, no. 3, pp. 2209–2218, 2026. https://doi.org/10.30871/jaic.v10i3.12727.

[16] D. O. Ratmana, M. Syaifur Rohman, F. Firdausillah, and G. Wilujeng Saraswati, "Perancangan Aplikasi Perhitungan Beban Kerja Dosen Terintegrasi dengan Pendekatan Waterfall" (Design of a Lecturer Workload Calculation Application Integrated with the Waterfall Approach), *J. Teknoif Tek. Inform. Inst. Teknol. Padang*, vol. 12, no. 2, pp. 139–148, 2024. (in Indonesian) https://doi.org/10.21063/jtif.2024.V12.2.139-148

[17] E. P. Estupiñán Cuesta, J. C. Martínez Quintero, and J. D. Avilés Palma, "DDoS Attacks Detection in SDN Through Network Traffic Feature Selection and Machine Learning Models," *Telecom*, vol. 6, no. 3, p. 69, 2025. https://doi.org/10.3390/telecom6030069

[18] B. Vijetha, "Agentic Intelligence for Unified Cyber Defense: A Self-Adaptive Framework for Threat Detection Across Cloud, Edge, and IoT Systems," *IEEE Access*, vol. 14, pp. 5104–5118, 2026. https://doi.org/10.1109/ACCESS.2026.3650833

[19] M. Aria, C. Cuccurullo, and A. Gnasso, "A comparison among interpretative proposals for Random Forests," *Mach. Learn. Appl.*, vol. 6, p. 100094, 2021. https://doi.org/10.1016/j.mlwa.2021.100094

[20] R. Y. Azhari, "Web Service Framework: Flask dan FastAPI" (Web Service Framework: Flask and FastAPI), *Technol. Inform. Insight J.*, vol. 1, no. 1, pp. 58–65, 2022. https://doi.org/10.32639/tiij.v1i1.54

[21] A. N. Safitri and I. Harkespan, "Pengembangan Web Service Menggunakan Framework FastAPI untuk Meningkatkan Kemudahan Integrasi Sistem Informasi Akademik Multiplatform" (Web Service Development Using the FastAPI Framework to Improve the Ease of Integrating Multiplatform Academic Information Systems), *J. Teknoif Tek. Inform. Inst. Teknol. Padang*, vol. 12, no. 2, pp. 149–157, 2024. (in Indonesian). https://doi.org/10.21063/jtif.2024.V12.2.149-157

[22] T. Alfa Rizzi, R. Aziz Abdillah, Y. Efani Yancandra, M. Wijaya, and A. Voutama, "Implementasi React Vite dalam Pengembangan Antarmuka Sistem Pemesanan Tiket Pesawat dengan Metode Scrum" (Implementation of React Vite in Developing the Interface of a Flight Ticket Booking

Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)

System Using the Scrum Method), *J. Process.*, vol. 20, no. 1, pp. 121–130, 2025. (in Indonesian). https://doi.org/10.33998/processor.2025.20.1.2231

[23] Z. M. Alshamaa and N. N. Saleem, "Black box software testing techniques: a literature review," *Passer J. Basic Appl. Sci.*, vol. 7, no. 2, pp. 1001–1011, 2025. https://doi.org/10.24271/psr.2025.508216.1963

[24] A. Amrulloh, A. D. Septiadi, M. Septiara, and P. A. Wicaksono, "Black Box Testing Using the Equivalence Partitions Technique to Test the Functionality of the Ternaku.id Website," *J. Multimed. Trend Technol.*, vol. 2, no. 3, pp. 171–178, 2023. https://doi.org/10.35671/jmtt.v2i3.43

[25] S. A. Kinasih, R. Sahputra, and C. Anwar, "Functional Suitability Testing of Web-Based Warehouse Inventory Application Using Black Box Testing," *Ambidextrous J. Innov. Effic. Technol. Organ.*, vol. 3, no. 02, pp. 138–154, 2026. https://doi.org/10.61536/ambidextrous.v3i02.391

[26] M. Suppa, K. Benešová, and A. Švec, "Cost-effective Deployment of BERT Models in Serverless Environment," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, Online: Association for Computational Linguistics, 2021, pp. 187–191 https://doi.org/10.18653/v1/2021.naacl-industry.24

[27] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *ACM Comput. Surv.*, vol. 55, no. 12, p. 259, 2023. https://doi.org/10.1145/3578938

[28] D. F. Doghramachi and S. Y. Ameen, "Internet of Things (IoT) Security Enhancement Using XGboost Machine Learning Techniques," *Comput. Mater. Contin.*, vol. 77, no. 1, pp. 717–732, 2023. https://doi.org/10.32604/cmc.2023.041186

[29] R. Nwachukwu, "Rethinking Graph-Based Approaches: An Empirical Study of Feature Engineering in Network Intrusion Detection," *J. Teknoif Tek. Inform. Inst. Teknol. Padang*, vol. 14, no. 1, pp. 1–10, 2026. https://doi.org/10.21063/jtif.2026.V14.1.1-10

DOI : https://doi.org/10.21063/jtif.2026.V14.2.110-123
Creative Commons Attribution-Share Alike 4.0 International License (CC BY-SA 4.0)