IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 September 2026, accepted 26 September 2026, date of publication 1 October 2026, date of current version 7 October 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3738895

RESEARCH ARTICLE

TraceLog: Session-Based Anomaly Detection for Web Server Access Logs With URL Sequence Rarity and Evidence-Guided Language-Model Interpretation

SEYEON JANG[1], JINHO PARK[1], NAYOUNG KIM[1], SOOHYUN PARK[2], AND YOUNGWOO KIM[1,2]
[1]Department of Computer Engineering, Kumoh National Institute of Technology, Gumi, Gyeongsangbuk-do 39177, Republic of Korea
[2]Department of Software Engineering, Kumoh National Institute of Technology, Gumi, Gyeongsangbuk-do 39177, Republic of Korea

Corresponding author: Youngwoo Kim (ywkim09@gmail.com)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by Korean Government [Ministry of Science and ICT (MSIT)] under Grant RS-2024-00345567; in part by Korea Institute of Industrial Technology as "Development of an Intelligent Condition Diagnosis Platform for Future Mobility" under Grant KITECH JK-26-0012; in part by the Regional Innovation System and Education (RISE)–Regional Growth Innovation LAB Program through Gyeongbuk RISE Center, funded by the Ministry of Education (MOE) and Gyeongsangbuk-do, Republic of Korea, under Grant 2025-rise-15-105; and in part by the High Performance Computing Resources of the Supercomputing Center, Kumoh National Institute of Technology, in 2026.

ABSTRACT Web server access logs record user behavior as sequences of HTTP requests, but request-level detection may not capture attack flows distributed across multiple requests. This study proposes TraceLog, a session-level anomaly detection framework that groups HTTP requests using an IP–User-Agent client key and a 30-min inactivity timeout and represents each session using 29 session-level features and one URL sequence rarity feature. URL rarity is modeled separately using a first-order Markov model, LogBERT-based masked language modeling, and self-supervised 1D-CNN next-URL prediction, with each resulting representation evaluated in a common Isolation Forest-based pipeline. Model development uses refined normal training data only, and the primary operating threshold is defined as the 99th percentile (P99) of a separate Normal Calibration score distribution without using attack labels. On 49,686 held-out normal sessions and 450 held-out LLM-generated attack sessions, the CNN-based pipeline achieved a Recall of 0.8489, an FPR of 0.0098, an AUC-ROC of 0.9960, and an average precision of 0.7415 at P99. Feature ablation showed that the incremental contribution of URL sequence rarity was modest and dataset-dependent. External validation on an independent operational web server dataset further demonstrated substantial cross-service distribution shift: service-specific rebuilding of the CNN pipeline achieved a held-out FPR of 0.0110 and a Recall of 0.3526 at the primary P99 operating point. For the 869 CNN P99 Alerts, an optional external LLM generated interpretations from locally sanitized evidence without modifying detection decisions; conditional attack-type classification accuracy among the 382 detected attack Alerts was 91.10%. A two-rater evaluation of 100 explanations yielded faithfulness rates of 90% and 67%, with unsupported-statement rates of 0.97% and 3.21%, respectively. These results support TraceLog as an auxiliary session-prioritization and interpretation framework while highlighting limitations in cross-service generalization and LLM-based explanation.

INDEX TERMS Anomaly detection, explainable artificial intelligence, HTTP access logs, session-based analysis.

The associate editor coordinating the review of this manuscript and approving it for publication was Xuebo Zhang.

I. INTRODUCTION

As web services have become key user-facing interfaces in various domains, including electronic commerce, finance,

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

and public services, security threats targeting web servers and web applications have continued to receive attention. Injection and broken access control are representative web application security risks, and Structured Query Language (SQL) injection, cross-site scripting (XSS), path traversal, and unauthorized access to sensitive paths may appear as related attack patterns [1]. Such attacks may be directly manifested in a single Hypertext Transfer Protocol (HTTP) request, but they may also be observed as a sequence of preliminary reconnaissance, repeated path access, and actual attack requests distributed across multiple requests. Therefore, a method that independently analyzes individual requests may not adequately explain whether a given request is part of a normal user navigation flow or a suspicious attack flow.

HTTP access logs recorded by web servers contain request timestamps, client information, request uniform resource identifiers (URIs), response status codes, and transferred data volumes and can therefore serve as a primary data source for analyzing web service access behavior. Previous studies on web anomaly detection have learned features such as HTTP request length, character distribution, and parameter structure to calculate an anomaly score for each request [2]. Although such request-level approaches can be used to identify anomalous requests that are difficult to detect using predefined attack signatures alone, they do not directly represent session-level context formed across multiple requests, such as request order, inter-request intervals, and uniform resource locator (URL) navigation flows. This study addresses the analysis of session-level behavioral patterns directly from access logs, with particular consideration given to web service environments in which dedicated security personnel or separate security information and event management (SIEM) infrastructure may be difficult to maintain.

Studies that learn patterns from normal log sequences have also been proposed to incorporate sequential context. DeepLog uses a long short-term memory (LSTM) network to predict the next log event in a normal system log sequence and identifies sequences that deviate from the learned normal patterns as anomalous [3]. LogBERT models the bidirectional context of normal log sequences through self-supervised learning based on bidirectional encoder representations from transformers (BERT) [4]. Although these studies have the advantage of accounting for the order and context of log events, they primarily analyze sequences of log keys in system logs. For web access logs, session-level behavioral features such as request volume, session duration, response-status distribution, URI structure, and access to security-related paths should be considered together with the order of URL access.

Because reliable labels for attack cases may be limited in operational environments, unsupervised anomaly detection methods that can be trained using only normal data may provide a practical alternative. Isolation Forest (IF) is an unsupervised anomaly detection method that repeatedly performs random partitioning and identifies observations that are isolated through shorter paths than other observations as anomalies. It can therefore be applied even when attack cases are not included in the training data [5]. However, the anomaly score produced by IF quantifies only how readily a session is isolated relative to other sessions and does not directly indicate which input features contributed to the decision.

SHapley Additive exPlanations (SHAP) is a feature attribution method that quantifies the contribution of each input feature to an individual model output [6]. Applying SHAP makes it possible to identify the primary features that affected the anomaly score of a particular Alert session; however, connecting feature names and contribution values to the actual HTTP request flow or a potential attack type still requires interpretation by an operator. Recent studies have investigated the use of large language models (LLMs) for security operations center (SOC) tasks such as security alert collection, investigation, triage, and report generation [7]. However, LLMs may generate content that is inconsistent with the provided evidence, and this potential for hallucination has been reported as a factor limiting the reliability of LLM outputs [8]. Therefore, rather than allowing an LLM to make anomaly detection decisions independently, the quantitative results of the detection model and only the minimum privacy-filtered request evidence required for interpretation should be provided to the LLM. The role of the LLM is restricted to generating an auxiliary natural-language explanation after the core anomaly detection decision has already been made.

Previous studies have separately addressed web request-level anomaly detection, log sequence modeling, unsupervised anomaly detection, and the interpretability of detection results. Building on these directions, this study integrates session-level web-access analysis, alternative URL-sequence rarity representations, unsupervised anomaly detection, and evidence-guided interpretation within a unified workflow. The evaluation protocol further separates model development, operating-point determination, and held-out performance assessment to reduce dependence on labeled attack information.

To address these limitations, this study proposes TraceLog, which reconstructs HTTP access logs into sessions using an inactivity timeout and a client key based on an Internet Protocol (IP) address and User-Agent string and represents each session using 29 statistical features together with one URL-sequence rarity feature. The rarity feature is derived separately using Markov transition probabilities, LogBERT-based masked language modeling (MLM), or self-supervised one-dimensional convolutional neural network (1D-CNN) next-URL prediction, allowing the three candidate pipelines to be compared under the same statistical features, standardization procedure, and downstream IF structure.

For sessions classified as Alerts, TraceLog optionally provides an LLM with a minimized evidence summary consisting of selected session statistics, Top-K SHAP

contributions, and locally sanitized request evidence. The LLM generates an auxiliary natural-language interpretation but does not modify anomaly scoring or Alert classification. The core detection pipeline therefore remains independent of the external interpretation module.

The main contributions of this study are as follows.

1) We propose a session-based anomaly detection framework that reconstructs web server HTTP access logs at the session level using a client key and an inactivity timeout and analyzes each session by combining 29 session-level statistical features with one URL sequence rarity feature. The proposed framework is intended for use in small- and medium-scale web service environments in which dedicated security personnel or separate security infrastructure may be difficult to maintain.
2) We compare three methods for calculating URL sequence rarity: a Markov-based transition-probability model, LogBERT-based masked language modeling, and self-supervised 1D-CNN-based next-URL prediction. These methods are evaluated using common session statistical features and a common downstream anomaly detection structure. This comparison evaluates detection performance at the candidate-pipeline level while accounting for differences in each rarity representation and its associated model-specific training and inference configurations.
3) We establish a calibration-and-evaluation protocol that separates model development, operating-point determination, and held-out performance assessment. The primary 99th-percentile (P99) threshold for each pipeline is derived exclusively from Normal Calibration without attack labels. At the auxiliary FN=0 point, the false negative (FN) count is zero; at the auxiliary F_2-max point, the F_2-score, which weights Recall more strongly than Precision, is maximized. All thresholds are frozen before evaluation on the corresponding held-out test subsets.
4) For Alert sessions, we construct a locally sanitized evidence summary comprising selected session statistics, Top-K SHAP contributions, and minimized HTTP request evidence for optional LLM-based interpretation. The LLM remains separated from anomaly scoring and Alert classification. We quantitatively evaluate this interpretation path using conditional attack-type classification accuracy on CNN P99 true-positive Alerts, repeated-generation consistency, and a two-rater human assessment of explanation faithfulness and unsupported factual statements.
5) We broaden the empirical evaluation through a seven-configuration feature ablation, external validation of the CNN-based pipeline on an independent operational web server dataset, sensitivity analyses of inactivity-timeout and client-key settings, and paired statistical tests. Computational performance is evaluated using a central processing unit (CPU)-only benchmark of the local CNN-based pipeline, with Alert-only SHAP processing, privacy sanitization, and external-LLM latency measured separately. These analyses examine feature dependence, cross-service transferability, sessionization sensitivity, and the computational characteristics of the evaluated implementation.

The remainder of this paper is organized as follows. Section II reviews previous studies related to web log-based anomaly detection, sequence-based log anomaly detection, and the interpretability of detection results. Section III describes in detail the sessionization procedure, feature extraction, URL sequence rarity calculation, anomaly-score-based decision process, and SHAP–LLM-based interpretation module of TraceLog. Section IV presents the offline evaluation results and statistical analysis of the proposed framework. Section V discusses the interpretation of the results, study limitations, and directions for future research. Finally, Section VI summarizes the main findings of this study.

II. RELATED WORK

Log-based anomaly detection has been widely studied for system fault diagnosis and security monitoring, with web access logs and system logs serving as common data sources. Early web security monitoring primarily relied on predefined attack signatures and handcrafted request rules to identify suspicious requests. Subsequently, statistical and machine learning-based approaches were proposed by exploiting request-level features such as HTTP request length, character distribution, URL structure, and query parameters [2], [9], [10]. These studies mainly focused on structural characteristics of individual HTTP requests, while some approaches also employed sequential models. However, they have limited capability to jointly represent the overall request order, temporal intervals between requests, and the rarity of navigation paths within an entire session.

To address this limitation, session-based approaches were proposed to model user behavior by grouping consecutive requests into sessions. Previous studies modeled page occurrence probabilities and session-level behavioral patterns to detect sessions deviating from normal user profiles [11]. In addition, visualization frameworks were introduced to explore user behavior by comparing session attributes and event sequences [12]. Although these studies adopted session-level analysis as a richer unit of analysis than individual requests, they did not jointly utilize multidimensional session-level statistical features and the rarity of URL access sequences as inputs to anomaly detection models. Meanwhile, parsing unstructured log messages into structured log events or log keys has become an essential preprocessing step in system log anomaly detection, and the fixed-depth tree-based online log parser Drain has become one of the most widely adopted approaches [13].

Recently, deep learning-based approaches have been widely studied for learning temporal dependencies among log

events. Recurrent neural network (RNN) models, particularly LSTM networks, detect anomalous sequences by predicting the next log event from previously observed events [3]. CNN-based methods transform log keys into embedding vectors and apply one-dimensional convolutional filters of different sizes to capture local dependencies within log sequences [14]. Compared with conventional statistical feature engineering, these methods directly learn sequential and contextual patterns from log sequences.

Transformer-based approaches further improve contextual modeling by employing masked language modeling to capture bidirectional contextual information and identify deviations from normal log patterns [4], [15]. Unlike LSTM-based methods, which model sequential dependencies through next-event prediction, BERT-based methods learn bidirectional contextual representations, and some studies eliminate the need for explicit log parsing by directly processing raw log messages. Furthermore, empirical studies have systematically compared the effectiveness and practical limitations of existing log anomaly detection techniques in real-world online service environments [16]. Nevertheless, most existing sequence-based methods employ sequence models as the final anomaly detector and mainly target system logs or event logs collected from the Hadoop Distributed File System (HDFS), Blue Gene/L (BGL), and online services. Studies that transform URL navigation behavior into rarity features and combine them with session-level statistical features for downstream unsupervised anomaly detection remain relatively limited.

In practical web services, obtaining sufficient labeled data covering diverse attack types is often infeasible. Consequently, unsupervised and one-class anomaly detection methods have attracted considerable attention. IF isolates anomalous samples through recursive random partitioning and exploits the fact that anomalies tend to be isolated with fewer partitioning steps than normal instances [5]. This approach estimates anomaly scores without requiring predefined attack classes by measuring deviations from the distribution of normal sessions. However, the resulting anomaly score only reflects the degree of abnormality and does not explicitly explain which input features contributed to the prediction or how the detected anomaly relates to a specific attack scenario.

To improve the interpretability of anomaly detection models, feature attribution techniques have been widely adopted. SHAP quantifies the contribution of each input feature to model predictions based on Shapley values from cooperative game theory and has become one of the most widely adopted explanation methods [6]. More recently, SHAP has also been applied to intrusion detection systems to identify influential features and enhance model interpretability [17]. However, existing studies mainly focus on visualizing feature importance, while relatively little attention has been paid to integrating SHAP feature attributions with raw request sequences to generate natural-language attack interpretations using large language models.

Recently, LLMs have also been introduced into log analysis and anomaly detection. Existing studies employ LLMs either as anomaly detection models themselves or as semantic feature extractors combined with domain knowledge to improve detection performance [18], [19]. Although these approaches demonstrate the effectiveness of LLMs in log understanding and automated anomaly detection, LLMs are directly involved in anomaly prediction or feature representation learning. In contrast, the proposed framework does not use the LLM to determine whether a session is anomalous. Instead, the LLM is solely responsible for generating auxiliary natural-language interpretations based on the quantitative evidence produced by the anomaly detection model and privacy-filtered request evidence, including selected session statistics and Top-K SHAP feature attributions.

Motivated by the above limitations, this study proposes a session-level anomaly detection framework for web access logs by combining session-level statistical features with URL sequence rarity features. Rather than directly employing sequence models as anomaly detectors, rarity features generated by different sequence modeling approaches are evaluated under the same IF framework. Furthermore, Top-K SHAP feature attributions, selected statistical evidence, and privacy-filtered request evidence are jointly provided to the optional LLM module, thereby separating anomaly detection from explanation generation while limiting the disclosure of raw log data.

III. MATERIALS AND METHODS

A. OVERVIEW OF THE PROPOSED FRAMEWORK

This study proposes TraceLog, a framework that reconstructs web server HTTP access logs into sessions and detects anomalous sessions according to the extent to which they deviate from the feature distribution of normal sessions. Because an individual HTTP request captures only an access event at a specific point in time, it provides insufficient context for determining whether the request belongs to a legitimate user flow or forms part of an attack sequence. Accordingly, the proposed framework groups consecutive HTTP requests associated with a client key, defined as the combination of a client IP address and User-Agent string, into sessions and uses both session-level behavioral characteristics and URL access order as inputs to anomaly detection.

TraceLog organizes normalized HTTP requests into sessions and derives session-level behavioral features and URL sequences. Alternative URL sequence rarity representations are then evaluated independently within a common IF-based anomaly detection pipeline, with operating thresholds determined from calibration data before held-out evaluation. For sessions classified as Alerts, SHAP-based feature attributions are computed locally, and an optional external LLM interprets a locally privacy-filtered and minimized evidence summary. The detailed procedures for session construction, feature

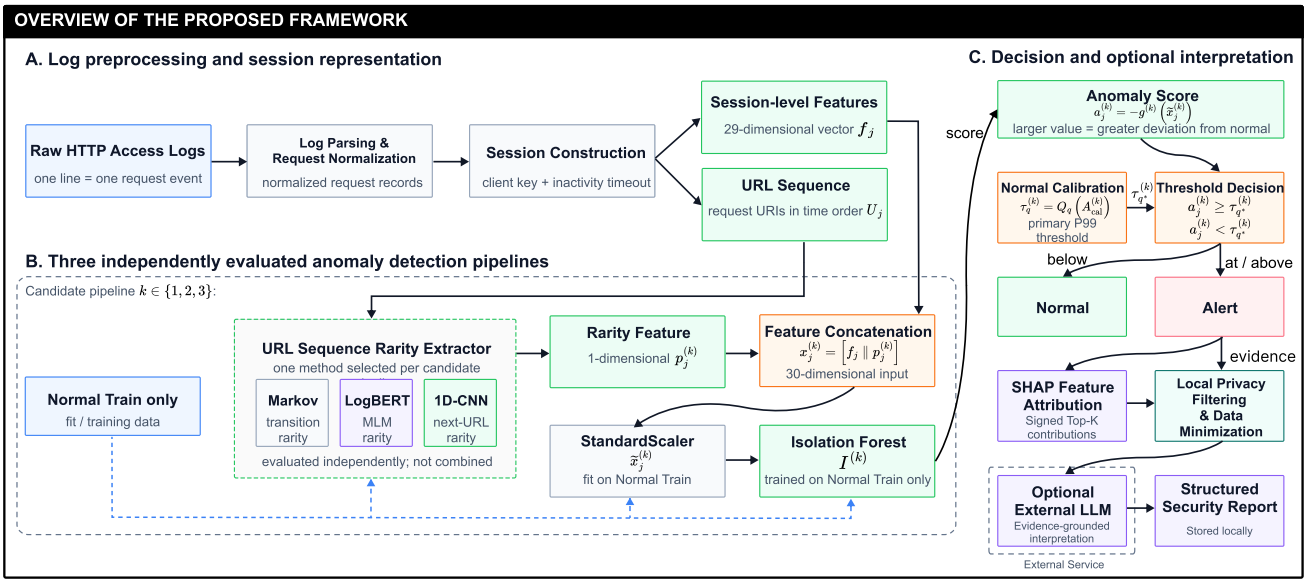


FIGURE 1. Overview of the proposed TraceLog framework.

extraction, rarity modeling, model training, operating-point selection, and LLM-based interpretation are described in the following subsections.

Fig. 1 summarizes the overall workflow of the proposed TraceLog framework.

B. DATASET

The internal data used in this study comprise a public web server access log for constructing the refined normal baseline and LLM-generated synthetic attack sessions for controlled attack evaluation. According to their roles in model development, operating-point determination, and held-out evaluation, the data were organized into five mutually separated subsets: Normal Train, Normal Calibration, Normal Test, LLM Synthetic Attack Calibration, and LLM Synthetic Attack Test.

Normal Train was reserved for model fitting and sequence-model hyperparameter selection, Normal Calibration for primary operating-point determination, and the two held-out test subsets for final evaluation. LLM Synthetic Attack Calibration was used only for the auxiliary attack-label-dependent operating points. The detailed threshold definitions are given in Section III-C6.

1) PUBLIC WEB LOG DATASET

To construct a candidate pool for the normal baseline, we used the *Web Server Access Logs* dataset publicly available on Kaggle [20]. The dataset contains approximately 3.3 GB of web server access logs collected from zanbil.ir, an Iranian e-commerce website, and records HTTP requests generated during real-world service operation. Each raw log entry contains fields such as the client IP address, request time, HTTP request line, HTTP status code, response size, referer, and User-Agent.

Each row of the raw log is treated as a single HTTP request event and converted into a normalized request record; the record definition and the URI-derived attributes are described in Section III-C1.

The public web log dataset provides no explicit normal or attack labels at either the request or session level. Moreover, because the logs were collected in an operational environment, they may contain automated scans or other anomalous access flows. We therefore applied a security-flag-based refinement procedure, as described in the following subsection.

Fig. 2 illustrates how a raw access log line is transformed into structured request fields.

2) NORMAL DATA REFINEMENT

Normal-data refinement was used to construct the internal normal baseline by excluding potentially suspicious sessions from the public web logs. After sessionization, a session was excluded if any request triggered a security flag for suspicious keywords, sensitive or scanner-related paths, login- or admin-related suspicious patterns, or malformed requests. The retained sessions are referred to as *refined normal sessions*; this procedure conservatively reduces suspicious contamination but does not establish complete ground-truth normal traffic.

Several request-level indicators used for this refinement are also aggregated into TraceLog's security-rule-derived feature group. Consequently, the internal normal-baseline construction and the security-rule-derived features are structurally related, which may increase the apparent contribution of this feature group on the internal synthetic benchmark. This

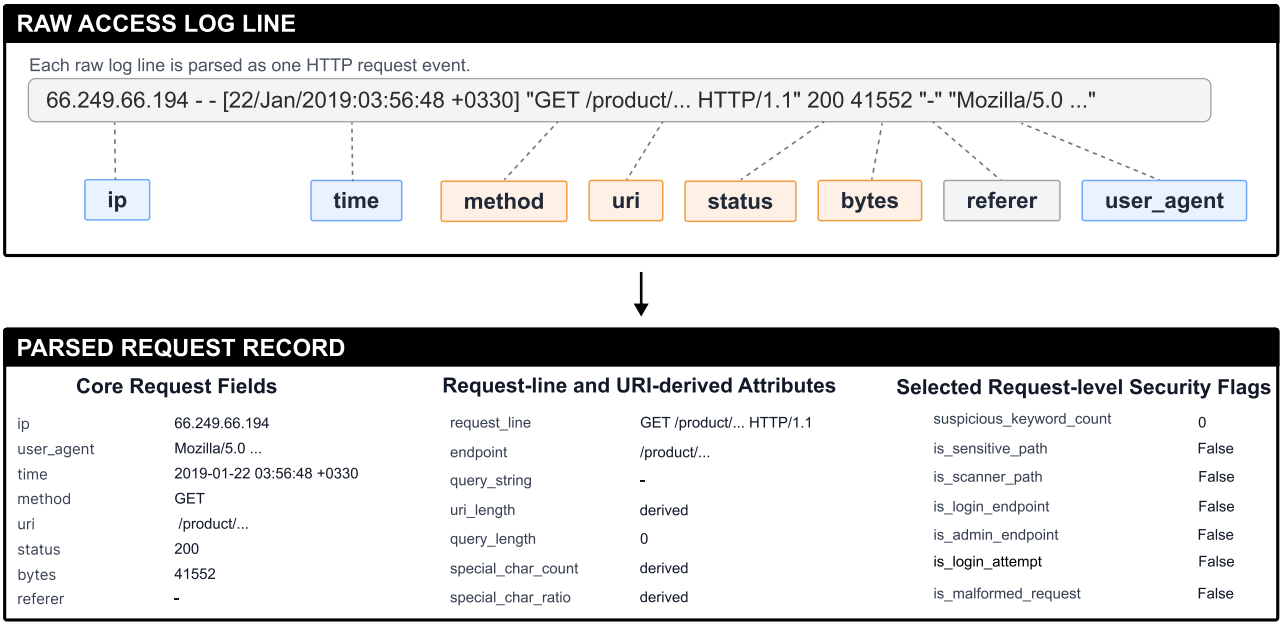


FIGURE 2. Conversion of a raw web server access-log line into a normalized request record.

preprocessing dependence is therefore considered explicitly in the feature-ablation analysis and its interpretation.

The refinement produced 331,234 normal sessions, which were shuffled at the session level using a random seed of 42 and partitioned into Normal Train (231,863; 70%), Normal Calibration (49,685; 15%), and Normal Test (49,686; 15%).

3) SYNTHETIC ATTACK GENERATION

Because the public web log dataset does not provide explicit attack labels, an LLM was used to construct a controlled synthetic attack benchmark for internal evaluation. The generated sessions followed the Apache access-log schema and the general service context of the public normal log source while representing six predefined web-attack scenarios: SQL injection, cross-site scripting (XSS), path traversal, administrator/login brute-force, scanner-like probing, and mixed low-and-slow. These synthetic sessions were constructed for controlled evaluation and are not treated as substitutes for attack logs collected from operational environments.

The synthetic sessions were generated using ChatGPT (OpenAI) under a common generation protocol. Benign access-log samples from the target service were provided only as references for log format, URL structure, HTTP methods, response-code patterns, User-Agent style, and general service context. The generation instructions explicitly prohibited optimization of the generated sessions for any anomaly detector, feature set, security rule, operating threshold, or evaluation metric. No detector implementation, feature-extraction code, trained model parameters, anomaly scores, or operating thresholds were provided during generation.

Each synthetic IP address was assigned to a single generated session, and a consistent User-Agent was maintained within each session. Requests were generated in chronological order and were constructed to remain within the 30-min inactivity timeout used for sessionization. The generation protocol introduced variation in request paths, query strings, HTTP methods, response status codes, response sizes, timing patterns, referers, User-Agent values, and scenario-specific attack patterns.

Two predefined difficulty levels, *Clear* and *Moderate*, were used to vary behavioral intensity and attack-evidence density without reference to the anomaly detector. Clear sessions contained more concentrated attack evidence, whereas Moderate sessions contained less concentrated attack activity together with a greater proportion of benign-looking context while retaining identifiable scenario-specific attack behavior. At generation time, the attack/context request ranges were predefined by scenario group: for SQL injection, XSS, and path traversal, Clear sessions contained 8–12 attack requests and 1–2 benign/context requests, whereas Moderate sessions contained 5–8 and 1–3, respectively; for administrator/login brute-force and scanner-like probing, the corresponding ranges were 20–30 and 1–3 for Clear and 12–20 and 1–3 for Moderate; for mixed low-and-slow, they were 10–15 and 2–4 for Clear and 7–10 and 2–4 for Moderate. The realized characteristics of the generated sessions were also examined after generation using descriptive outputs of the session-feature extraction procedure, including request count and security-evidence density. These outputs are used only

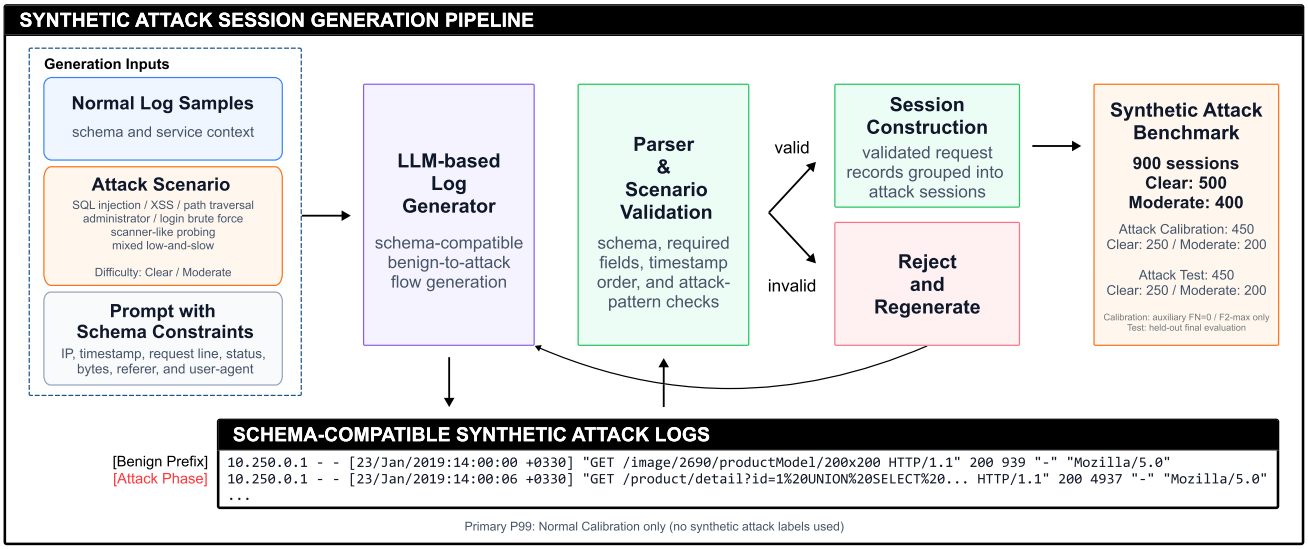


FIGURE 3. Generation, validation, and partitioning of the LLM-generated synthetic attack benchmark.

to characterize the realized Clear/Moderate difference and should not be interpreted as request-level attack ground-truth proportions. Their results are reported with the detection results in Section IV-D.

The resulting benchmark comprised 900 synthetic attack sessions, including 500 Clear sessions and 400 Moderate sessions. The generated logs were processed using the same Apache-log parser, session-construction procedure, and session-feature extraction pipeline used for the normal data. The benchmark was separated into LLM Synthetic Attack Calibration and a held-out LLM Synthetic Attack Test, each containing 450 sessions. The held-out test set comprised 250 Clear and 200 Moderate sessions. The predefined attack-scenario and difficulty labels were retained for subsequent performance analysis.

LLM Synthetic Attack Calibration and Test were kept separate from model fitting and hyperparameter selection, with the former used only for auxiliary operating-point analysis and the latter reserved for held-out internal evaluation. Accordingly, this benchmark provides controlled evidence for the predefined synthetic attack scenarios rather than evidence of generalization to arbitrary operational attack traffic.

Fig. 3 summarizes the synthetic-attack generation, validation, and benchmark-partitioning workflow.

4) EXTERNAL VALIDATION DATASET

To evaluate cross-service generalization beyond the internal synthetic-attack benchmark, we used Organization Y (Org Y) from the publicly available *Web Server Log Dataset* [21], which is released under the MIT License. The dataset was collected from an anonymized public-sector organization and contains OpenLiteSpeed/CyberPanel web server logs together with dataset-provider-supplied YAML-formatted labeling rules. In the present study, the provider-defined labels were used as an independent external reference and were not constructed using the TraceLog security rules employed to refine the internal normal baseline.

For request-level labeling, the dataset-provided YAML-formatted rules were applied to the original access-log records. Within each rule, all specified filter patterns were required to match, matching was case-insensitive, and the "%" symbol was interpreted as a wildcard. A successfully parsed HTTP request that matched at least one provider rule was assigned the corresponding provider-defined attack label, whereas a valid request that matched no rule was treated as normal. A session was labeled as an attack session if it contained at least one request matched by a provider rule. TraceLog's own security-rule-based normal-data refinement was not used to assign Org Y labels.

Because Org Y contains logs from multiple web server sources, the final preprocessing pipeline preserved log-source boundaries during session construction. We processed 163 access-log files associated with 21 log-source boundaries. Sessions were constructed using the tuple (log-source boundary, IP address, User-Agent) as the client key together with the same 30-min inactivity timeout used in the internal pipeline. Rotated log files belonging to the same source were treated as a common source, whereas requests from different log-source boundaries were not allowed to form the same session. After parsing, labeling, service-aware session construction, and application of the TraceLog session-feature extractor, the resulting external dataset contained 73,415 sessions: 67,487 normal sessions and 5,928 provider-defined attack sessions.

The 67,487 normal sessions were randomly partitioned at the session level using seed 42 into external Normal Train ($n = 47{,}240$), Normal Calibration ($n = 10{,}123$), and Normal Test ($n = 10{,}124$), corresponding approximately

to a 70%/15%/15% split. The 5,928 provider-defined attack sessions were divided into external Attack Calibration ($n = 2{,}964$) and held-out Attack Test ($n = 2{,}964$). The held-out Normal Test and Attack Test subsets were not used for model fitting or operating-point determination.

External validation was conducted under two complementary protocols. In the strict zero-shot protocol, the CNN-based TraceLog pipeline trained on the internal normal data was transferred to Org Y without fitting any model component or recalibrating the transferred threshold on Org Y. The frozen auxiliary FN=0 threshold determined from the internal calibration data was used in this strict transfer experiment to assess the portability of an operating point fixed in the original service environment.

In the rebuilt protocol, a service-specific CNN pipeline was reconstructed using only Org Y Normal Train: the CNN URL-sequence rarity model, StandardScaler, and IF were refitted on the external normal training data. The three operating points were then determined from the external calibration subsets according to the same definitions used for the internal evaluation. P99, the primary operating point, was determined exclusively from external Normal Calibration, whereas FN=0 and F_2-max were retained as auxiliary operating points using the labeled calibration subsets. Once determined, the thresholds were frozen and evaluated on external Normal Test and held-out Attack Test. This protocol assesses the behavior of TraceLog when a normal baseline is rebuilt for the target service rather than directly transferred across services.

C. SESSION-BASED ANOMALY DETECTION

This study formulates anomaly detection in web server HTTP request logs as a session-level outlier detection problem. Because a session contains a consecutive access flow generated by the same requesting entity over a certain period, it can jointly represent request order, inter-request intervals, URL access diversity, error-response distributions, and repeated access patterns. Accordingly, we sort HTTP requests from the same requesting entity chronologically and divide them using an inactivity timeout to construct sessions. The detailed sessionization procedure is defined in Section III-C2.

Each session is represented by a session-level statistical feature vector $\mathbf{f}_j$ and a URL-sequence-based rarity feature $p_j^{(k)}$. The final input vector $\mathbf{x}_j^{(k)}$, obtained using the k-th rarity calculation method, is constructed as follows:

$$\mathbf{x}_j^{(k)} = [\mathbf{f}_j \parallel p_j^{(k)}]. \tag{1}$$

Here, $\mathbf{f}_j \in \mathbb{R}^{29}$ is the session-level statistical feature vector defined in Section III-C3, and $p_j^{(k)} \in \mathbb{R}$ is the one-dimensional rarity feature calculated using the k-th method. Therefore, the final input vector is represented as $[\mathbf{f}_j \parallel p_j^{(k)}] \in \mathbb{R}^{30}$.

This study compares three URL sequence rarity calculation methods, $k \in \{1, 2, 3\}$: a Markov-based method, a LogBERT-based MLM method, and a self-supervised 1D-CNN-based method. The resulting $\mathbf{x}_j^{(k)}$ is standardized using StandardScaler and then provided to IF, which calculates the anomaly score of each session based on the feature distribution of normal sessions. Finally, each session is classified as Alert or Normal by comparison with a calibration-based threshold.

1) LOG PARSING AND REQUEST NORMALIZATION

Because raw HTTP access logs are recorded as unstructured text, they are converted into structured request records before sessionization and feature extraction. Each log line is regarded as a single HTTP request event and parsed according to a common schema.

The i-th request record $r_{j,i}$ included in the j-th session S_j is defined as follows:

$$r_{j,i} = (\text{ip}, \text{ua}, t, m, u, s, b). \tag{2}$$

Here, ip denotes the client IP address, ua the User-Agent, t the request timestamp, m the HTTP method, u the request URI, s the HTTP status code, and b the response size. The variables ip and ua are combined into a client key for identifying the same requesting entity, while t is used to chronologically order requests within a session and calculate inter-request intervals. The variables m and u represent access behavior, whereas s and b reflect server-response characteristics.

The request URI u is subsequently divided into an endpoint and a query string during feature extraction, and the URI length, query-string length, and the number and ratio of special characters are calculated. These URI-related values are not independent raw fields included in $r_{j,i}$ but are used as derived attributes calculated from the request URI u. In addition, each record is annotated with whether it contains suspicious keywords, accesses sensitive or scanner-like paths, accesses login or admin endpoints, or constitutes a malformed request. These attributes are used as criteria for filtering suspicious sessions during normal-data refinement and are aggregated into session-level security-derived features. The primary fields generated by parsing are presented in Table 1.

2) SESSION CONSTRUCTION

The normalized HTTP request records are sorted in ascending order by request timestamp t, after which consecutive access flows from the same requesting entity are organized into sessions. Because web server access logs do not always include login-based user identifiers or server-side session identifiers, the combination of the client IP address and User-Agent is defined as a client key that approximates the requesting entity. However, the accuracy of distinguishing requesting entities may decrease when multiple users share the same IP address in a network address translation (NAT) environment or when a User-Agent is spoofed.

The client key c_i of the i-th normalized request record is defined as follows:

$$c_i = (\text{ip}_i, \text{ua}_i). \tag{3}$$

TABLE 1. Normalized request record schema.

Category	Primary Fields	Description
Basic request information	ip, time, method, uri	Requesting entity, request timestamp, HTTP request method, and URI
Response information	status, bytes	Server response status and response-size information
Client information	user_agent	Information about the client browser or automated tool
URI-derived information	endpoint, query_string, uri_length, query_length, special_char_count, special_char_ratio	Separation of the request URI into a path and query string and calculation of length and special-character information
Suspicious-request identification information	suspicious_keyword_count, is_sensitive_path, is_scanner_path, is_malformed_request	Auxiliary information indicating suspicious keywords, sensitive paths, scanner-like access, and malformed request formats
Login and administrator access information	is_login_endpoint, is_admin_endpoint, is_login_attempt	Access to login/admin-related endpoints and whether a login attempt occurred

Here, ip_i denotes the client IP address of the i-th request, and ua_i denotes the User-Agent of that request. Requests with the same c_i are regarded as belonging to the access flow of the same requesting entity, and the chronological request sequence R_c associated with client key c is defined as follows:

$$R_c = (r_{c,1}, r_{c,2}, \ldots, r_{c,L_c}). \tag{4}$$

Here, $r_{c,l}$ denotes the l-th request record associated with client key c, and L_c denotes the number of requests associated with that client key.

Session boundaries are determined using an inactivity-timeout method. The inactivity-time threshold is set to $\Delta_{\mathrm{sess}} = 1800$ s, and the session boundary is determined according to the time interval between two adjacent requests, $r_{c,l}$ and $r_{c,l+1}$, as follows:

$$\begin{aligned} t_{c,l+1} - t_{c,l} > \Delta_{\mathrm{sess}} &\Rightarrow \text{new session}, \\ t_{c,l+1} - t_{c,l} \le \Delta_{\mathrm{sess}} &\Rightarrow \text{same session}. \end{aligned} \tag{5}$$

Fig. 4 illustrates how normalized request records are divided into sessions based on the client key and $\Delta_{\mathrm{sess}} = 1800$ s. Requests classified as belonging to the same session within R_c are reindexed using session index j, and the j-th session S_j is defined as follows:

$$S_j = (r_{j,1}, r_{j,2}, \ldots, r_{j,n_j}). \tag{6}$$

Here, $r_{j,i}$ denotes the i-th request record included in session S_j, and n_j denotes the number of requests included in that session. The request timestamps within a session satisfy the following condition:

$$t_{j,1} \le t_{j,2} \le \cdots \le t_{j,n_j}. \tag{7}$$

The sensitivity of this sessionization rule to alternative inactivity-timeout values is evaluated in Section IV-G, where each timeout condition is assessed with condition-specific session reconstruction, model refitting, and P99 recalibration.

3) SESSION-LEVEL FEATURE EXTRACTION

Session S_j is converted into a session-level statistical feature vector $\mathbf{f}_j$ for input to the anomaly detection model. The vector $\mathbf{f}_j$ is defined as the concatenation of multiple feature groups that summarize request volume, temporal patterns, URL access diversity, response-status distribution, URI structure, and security-related access characteristics:

$$\mathbf{f}_j = [\mathbf{f}_j^{\mathrm{scale}} \parallel \mathbf{f}_j^{\mathrm{div}} \parallel \mathbf{f}_j^{\mathrm{status}} \parallel \mathbf{f}_j^{\mathrm{bytes}} \parallel \mathbf{f}_j^{\mathrm{uri}} \parallel \mathbf{f}_j^{\mathrm{sec}}]. \tag{8}$$

The composition of each feature group is presented in Table 2. The vector $\mathbf{f}_j^{\mathrm{scale}}$ is calculated based on the number of requests n_j included in the session and the time intervals between requests.

The session duration d_j is defined as follows:

$$d_j = t_{j,n_j} - t_{j,1}. \tag{9}$$

The time interval $\delta_{j,i}$ between two adjacent requests is calculated as follows:

$$\delta_{j,i} = t_{j,i+1} - t_{j,i}, \quad i = 1, \ldots, n_j - 1. \tag{10}$$

The mean, maximum, and minimum inter-request intervals are calculated from these values. The vector $\mathbf{f}_j^{\mathrm{sec}}$ is obtained by aggregating, at the session level, the request-level security attributes calculated in Section III-C1.

For the feature-ablation analysis, the 30-dimensional model input was further partitioned into three non-overlapping feature groups. We denote by G_S the 19 general session/statistical features comprising $\mathbf{f}_j^{\mathrm{scale}}$, $\mathbf{f}_j^{\mathrm{div}}$, $\mathbf{f}_j^{\mathrm{status}}$, $\mathbf{f}_j^{\mathrm{bytes}}$, and $\mathbf{f}_j^{\mathrm{uri}}$; by G_R the 10 security-rule-derived features comprising $\mathbf{f}_j^{\mathrm{sec}}$; and by G_U the one-dimensional URL sequence rarity feature. In the feature-ablation experiments, G_U was instantiated using the CNN-based rarity feature. Accordingly, the seven evaluated configurations were G_S, G_R, G_U, $G_S + G_R$, $G_S + G_U$, $G_R + G_U$, and $G_S + G_R + G_U$, where $G_S + G_R + G_U$ corresponds to the full 30-dimensional input.

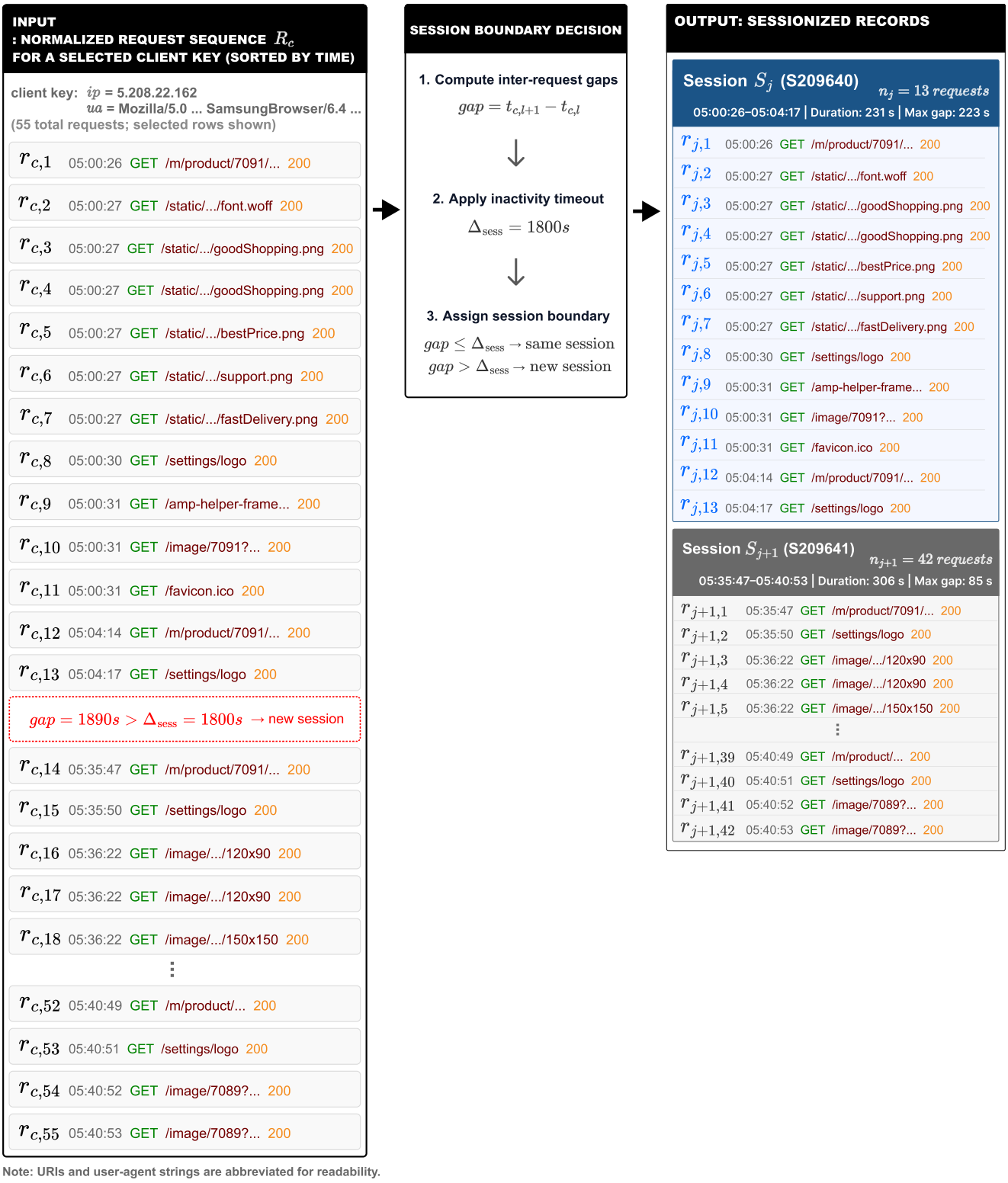


FIGURE 4. Session construction via client key and inactivity timeout.

To preserve the order of URL access within a session, the URL sequence U_j is also defined as follows:

$$U_j = (u_{j,1}, u_{j,2}, \ldots, u_{j,L_j}). \tag{11}$$

Here, $u_{j,i}$ denotes the i-th request URI included in S_j, and L_j is defined as follows:

$$L_j = \min(n_j, 50). \tag{12}$$

TABLE 2. Session-level feature groups.

Category	Primary Fields	Description
Session-scale and temporal features ($\mathbf{f}_j^{\text{scale}}$)	duration_sec, request_count, avg_request_interval_sec, max_request_interval_sec, min_request_interval_sec	Session duration, number of requests, and inter-request intervals
Request-diversity features ($\mathbf{f}_j^{\text{div}}$)	unique_url_count, unique_method_count	Numbers of unique URLs and HTTP methods within the session
Status-code features ($\mathbf{f}_j^{\text{status}}$)	error_4xx_ratio, error_5xx_ratio, status_200_count	Distribution of successful responses, client errors, and server errors
Response-size features ($\mathbf{f}_j^{\text{bytes}}$)	avg_bytes, max_bytes, std_bytes	Mean, maximum, and standard deviation of response sizes within the session
URI-structure features ($\mathbf{f}_j^{\text{uri}}$)	avg_uri_length, max_uri_length, avg_query_length, max_query_length, special_char_count_sum, special_char_ratio_avg	URI and query-string lengths and the degree of special-character inclusion
Security-related derived features ($\mathbf{f}_j^{\text{sec}}$)	suspicious_keyword_count_sum, login_count, admin_count, login_attempt_count, sensitive_path_count_sum, scanner_path_count_sum, distinct_error_url_count, notfound_404_count, forbidden_403_count, server_error_5xx_count	Suspicious keywords, login and administrator access, sensitive paths, scanner-like access, and error-URL characteristics

In the implementation, the request records of each session were sorted by request timestamp, and up to the first 50 chronological requests were used for the URL, status, and method sequences, while the session-level statistical features were calculated using all requests in the session. Accordingly, up to 50 URL tokens were provided as input to the Markov- and CNN-based rarity calculations.

For sessions consisting of a single request, the mean, maximum, and minimum inter-request intervals were all set to 0 because an interval between adjacent requests could not be defined. Empty URIs were replaced with `[EMPTY_URI]`, and empty HTTP methods were replaced with `UNKNOWN`.

4) URL SEQUENCE REPRESENTATION

Although the session-level statistical feature vector $\mathbf{f}_j$ can represent quantitative attributes such as the number of requests, status-code distribution, response size, and URI structure, it has limited ability to directly reflect how much the order of URL access within a session differs from normal user flows. Therefore, the URL sequence U_j of each session is used as input to calculate a rarity feature $p_j^{(k)}$ that quantifies URL flows rarely observed in normal sessions. The sequence U_j consists of the request URIs within a session arranged chronologically, with each URI treated as a single URL token. This study compares three sequence modeling methods, $k \in \{1, 2, 3\}$. The comparative structure is illustrated in Fig. 5.

a: MARKOV-BASED RARITY

The Markov-based method calculates the rarity of a URL flow using the probabilities of adjacent URL transitions observed in normal training sessions. The number of transitions $n(a, b)$ from URL token a to URL token b is counted from the U_j sequences of Normal Train, and the conditional transition probability is defined as follows:

$$P(b|a) = \frac{n(a, b)}{\sum_{b' \in \mathcal{V}} n(a, b')}. \tag{13}$$

Here, $\mathcal{V}$ is the URL vocabulary constructed from the Normal Train data. For unobserved transitions, a minimum probability of $\epsilon = 10^{-4}$ was applied. This value was selected through the limited systematic hyperparameter search described in Section III-C5. The transition-probability table was selected through 5-fold cross-validation (CV) using the Normal Train sessions. In each fold, a transition-probability table was constructed using the sessions in the training split, and the mean rarity was calculated for the normal sessions in the validation split using the corresponding table. Among the five folds, the transition-probability table constructed from the training split of the fold with the lowest mean validation rarity was selected as the final Markov rarity model. The Markov-based rarity $p_j^{(1)}$ of session S_j is defined as follows:

$$p_j^{(1)} = -\frac{1}{L_j - 1} \sum_{l=1}^{L_j - 1} \log P(u_{j,l+1}|u_{j,l}). \tag{14}$$

For sessions containing fewer than two URL tokens, adjacent URL transitions cannot be defined; therefore, the Markov-based rarity value is set to 0.

b: LogBERT-BASED RARITY

The LogBERT-based method treats a URL sequence as a token sequence and calculates rarity using the loss incurred when a trained masked language model predicts selected URL tokens [4]. A vocabulary is constructed from the URL tokens appearing in Normal Train sessions, and the special tokens `[CLS]`, `[SEP]`, `[MASK]`, `[PAD]`, and `[UNK]` are defined to construct BERT-based input sequences.

During session-feature generation, up to 50 URLs sorted by request timestamp were stored. The Markov- and CNN-based methods used up to 50 URL tokens from the stored sequence. The maximum input length of the LogBERT extractor used in the final experiment was set to 64. Each input consisted of a `[CLS]` token, up to 50 URL tokens, and a `[SEP]` token. Thus, up to 52 valid tokens were provided per session, and the remaining positions were filled with `[PAD]` tokens.

During training, 15% of the URL tokens, excluding special tokens, were selected as MLM prediction targets. Of the selected tokens, 80% were replaced with the `[MASK]` token, 10% were replaced with URL tokens randomly selected from the vocabulary, and the remaining 10% were left unchanged.

The LogBERT model was trained and internally selected through 5-fold cross-validation using Normal Train sessions. In each fold, the model was trained for 5 epochs, and the model state with the lowest validation MLM loss across epochs was retained as the candidate for that fold. The model state with the lowest validation MLM loss among the five folds was then selected as the final LogBERT rarity model.

During inference, deterministic fixed-stride masking was applied to reduce variation in rarity scores caused by random masking and to ensure that the same rarity score was always produced for the same session. The mask stride was set to $m_{\text{mask}} = 4$. Starting from the first URL token, positions 1, 5, 9, ... were selected for masking using a fixed stride. The URL tokens at the selected positions were simultaneously replaced with the `[MASK]` token within a single input sequence. Rather than repeatedly masking every URL position individually, this method was designed to evaluate the URL flow throughout the sequence at regular intervals.

Accordingly, the set of mask positions $\mathcal{M}_j$ for session S_j is defined as follows:

$$\mathcal{M}_j = \left\{ m \in \{1, \ldots, L_j\} \;\middle|\; m \equiv 1 \pmod{m_{\text{mask}}} \right\}. \quad (15)$$

Here, $\widetilde{U}_j^{(\mathcal{M}_j)}$ denotes the input sequence obtained by replacing the URL tokens at all positions included in $\mathcal{M}_j$ in URL sequence U_j with the `[MASK]` token. The LogBERT-based rarity $p_j^{(2)}$ of session S_j is defined as the mean negative log-likelihood of the masked URL tokens:

$$p_j^{(2)} = -\frac{1}{|\mathcal{M}_j|} \sum_{m \in \mathcal{M}_j} \log P_\theta \left(u_{j,m} \;\middle|\; \widetilde{U}_j^{(\mathcal{M}_j)} \right). \quad (16)$$

Here, $u_{j,m}$ is the actual URL token at mask-target position m, and $\widetilde{U}_j^{(\mathcal{M}_j)}$ is the input sequence in which the URL tokens at all selected mask positions have been replaced with the `[MASK]` token. $P_\theta \left(u_{j,m} \mid \widetilde{U}_j^{(\mathcal{M}_j)} \right)$ denotes the probability assigned by the LogBERT model to the actual URL token $u_{j,m}$, based on the unmasked URL context in the input sequence.

c: SELF-SUPERVISED CNN-BASED RARITY

The 1D-CNN-based method is formulated as a self-supervised learning task that predicts the next URL from a prefix of the URL sequence. For session S_j, the prefix up to time step l, $(u_{j,1}, \ldots, u_{j,l})$, is used as input, and the actual next URL $u_{j,l+1}$ is used as the prediction target. The model is trained using prefix–next URL pairs generated from normal training sessions.

The 1D-CNN model takes URL-token embeddings as input and generates a sequence representation through convolution, batch normalization, a rectified linear unit (ReLU), dropout, and adaptive max pooling. A classification layer then produces a probability distribution over the next URL. The CNN-based rarity $p_j^{(3)}$ of session S_j is defined as the mean negative log probability assigned to the actual next URL:

$$p_j^{(3)} = -\frac{1}{L_j - 1} \sum_{l=1}^{L_j - 1} \log \left[P_\phi \left(u_{j,l+1} \mid u_{j,1:l} \right) + \epsilon \right]. \quad (17)$$

Here, $P_\phi(u_{j,l+1} \mid u_{j,1:l})$ is the probability assigned by the CNN model to the actual next URL token $u_{j,l+1}$, based on prefix $u_{j,1:l}$. For numerical stability, $\epsilon = 10^{-8}$ was applied. For sessions containing fewer than two URL tokens, a next-URL prediction pair cannot be constructed; therefore, the rarity value was set to 0.

For CNN training, prefix–next URL pairs were generated from each URL sequence in Normal Train. For a sequence of length L_j, $(u_{j,1}, \ldots, u_{j,l})$ was used as the input prefix, and $u_{j,l+1}$ was used as the prediction target for each $l = 1, \ldots, L_j - 1$.

After combining all prefix–next URL samples generated from all Normal Train sessions into a single training sample set, 5-fold cross-validation was performed using seed 42. The model in each fold was trained for up to 10 epochs, and training was terminated early if the validation cross-entropy loss failed to improve for two consecutive epochs. The model state with the lowest validation loss across all folds was selected as the final CNN rarity model.

5) MODEL TRAINING AND SELECTION

Each candidate pipeline shares the same 29-dimensional session-level statistical feature vector $\mathbf{f}_j$ and constructs a 30-dimensional final input vector $\mathbf{x}_j^{(k)}$ by combining it with a different URL sequence rarity feature $p_j^{(k)}$. The five mutually separated data subsets and their roles in model fitting, operating-point determination, and held-out evaluation are defined in Section III-B.

For each pipeline, the StandardScaler was fitted exclusively on the input vectors from Normal Train and subsequently applied, without refitting, to the calibration and test

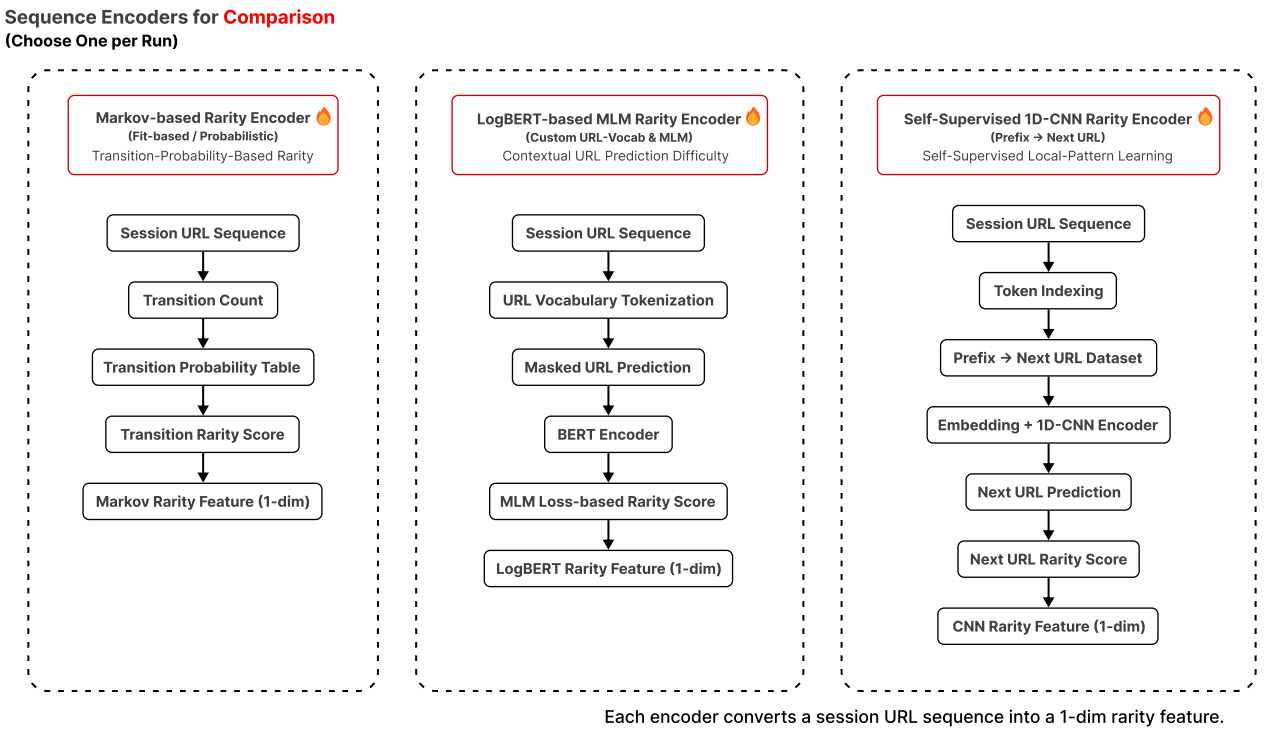


FIGURE 5. Comparative framework for sequence feature extraction.

subsets. The standardized Normal Train inputs were used to train the corresponding IF $I^{(k)}$. Labeled attack sessions were not used to train the URL sequence rarity models, fit the StandardScalers or IF models, or select sequence-model hyperparameters.

The hyperparameters of the three URL sequence rarity models were selected using a limited systematic grid search conducted exclusively on Normal Train. The search used all 231,863 Normal Train sessions and a fixed random seed of 42. For hyperparameter selection, the same session-level 5-fold partitions were used for all three rarity methods. For each candidate configuration, a model-specific validation objective was calculated in each fold, and the configuration with the best mean validation objective across the five folds was selected. Normal Calibration and all labeled attack and evaluation data were excluded from hyperparameter selection.

The method-specific search spaces, validation objectives, selected hyperparameters, and fixed model settings are summarized in Table 3.

After hyperparameter selection, final model-state selection followed the method-specific procedures summarized in Table 3. Markov and LogBERT used session-level 5-fold splits, whereas the final CNN used 5-fold splits over pooled prefix–next URL samples. No separate retraining on the entire Normal Train set was performed after final model-state selection.

The common downstream anomaly detector was an IF with 200 estimators, automatic contamination, a random seed of 42, and parallel execution over all available CPU cores (`n_jobs=-1`). All remaining parameters were left at the default values of scikit-learn 1.7.2.

6) ANOMALY SCORE ESTIMATION AND THRESHOLD CALIBRATION

The $I^{(k)}$ trained in Section III-C5 calculates an anomaly score for each standardized session input vector $\tilde{\mathbf{x}}_j^{(k)}$. Let the decision function of $I^{(k)}$ be denoted by $g^{(k)}(\cdot)$. To ensure that a larger anomaly score indicates a greater degree of anomalousness, the sign is inverted, and $a_j^{(k)}$ is defined as

$$a_j^{(k)} = -g^{(k)}\left(\tilde{\mathbf{x}}_j^{(k)}\right). \tag{18}$$

Here, $a_j^{(k)}$ denotes the anomaly score assigned to session S_j by the k-th candidate pipeline. Because the three pipelines use different URL sequence rarity features, their anomaly-score distributions and corresponding operating thresholds are determined independently.

Let D_{cal}^N and D_{cal}^A denote the Normal Calibration and LLM Synthetic Attack Calibration sets, respectively. Their anomaly-score sets for the k-th pipeline are defined as

$$A_{\text{cal},N}^{(k)} = \left\{ a_j^{(k)} \mid S_j \in D_{\text{cal}}^N \right\}, \tag{19}$$

and

$$A_{\text{cal},A}^{(k)} = \left\{ a_j^{(k)} \mid S_j \in D_{\text{cal}}^A \right\}. \tag{20}$$

The primary operating point was defined independently of attack labels using only the Normal Calibration score

TABLE 3. Model configurations and limited hyperparameter-search settings.

Configuration	Markov	LogBERT	Self-supervised 1D-CNN
Input sequence	Up to 50 URL tokens	Up to 50 URL tokens with `[CLS]` and `[SEP]`; padded to a maximum input length of 64	Up to 50 URL tokens
Vocabulary	URL transition states constructed from Normal Train	138,674 tokens including five special tokens	138,671 tokens including `<PAD>` and `<UNK>`
Model structure	First-order URL transition-probability table	Hidden size 64, 3 Transformer layers, 4 attention heads, intermediate size 128, and maximum position embeddings 128	Embedding size 32, 64 convolution filters, kernel size 7, padding 3, batch normalization, ReLU, dropout 0.2, and adaptive max pooling
Learning objective	Empirical estimation of adjacent-URL transition probabilities	Masked language modeling	Next-URL cross-entropy prediction
Optimizer	–	AdamW	Adam
Learning rate	–	1×10^{-3}	1×10^{-3}
Batch size	–	32	64
Training epochs	–	5	Up to 10
Hyperparameter search space	Unobserved transition probability: $\{10^{-4}, 10^{-5}, 10^{-6}\}$	Transformer layers: $\{1, 2, 3\}$; learning rate: $\{5 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}\}$	Kernel size: $\{3, 5, 7\}$; convolution filters: $\{16, 32, 64\}$
Hyperparameter-selection CV	5-fold, session-level	5-fold, session-level	5-fold, session-level
Hyperparameter-selection objective	Minimum 5-fold mean normal-validation rarity	Minimum 5-fold mean deterministic MLM validation loss	Minimum 5-fold mean next-URL validation cross-entropy loss
Selected searched values	Unobserved transition probability 10^{-4}	3 Transformer layers; learning rate 1×10^{-3}	Kernel size 7; 64 convolution filters
Final model training / selection	5-fold, session-level; retain the transition table with the lowest validation rarity	5-fold, session-level; retain the model state with the lowest validation MLM loss	5-fold, prefix–next URL sample-level; retain the model state with the lowest validation cross-entropy loss; early-stopping patience of 2
Rarity calculation setting	Unobserved transition probability of 10^{-4}	Training mask probability of 0.15 with an 80/10/10 replacement policy; evaluation mask stride of 4	Probability stabilization constant $\epsilon = 10^{-8}$
Random seed	42	42	42

distribution. For the k-th candidate pipeline, the primary threshold was set to the empirical 99th percentile of $A^{(k)}_{\mathrm{cal},N}$:

$$\tau_k^{\mathrm{P99}} = Q_{0.99}\left(A^{(k)}_{\mathrm{cal},N}\right), \tag{21}$$

where $Q_{0.99}(\cdot)$ denotes the empirical 99th-percentile operator. This threshold corresponds to a nominal 1% upper-tail false-positive budget on Normal Calibration and was used as the primary operating point throughout the main evaluation.

For complementary analysis, two attack-label-dependent operating points were also determined using the calibration subsets. First, the auxiliary FN=0 threshold was defined as the largest threshold that detected every session in LLM Synthetic Attack Calibration. Under the decision rule that sessions with anomaly scores greater than or equal to the threshold are classified as Alerts, this threshold is

$$\tau_k^{\mathrm{FN0}} = \min A^{(k)}_{\mathrm{cal},A}. \tag{22}$$

Second, the auxiliary F_2-max operating point was selected by maximizing the F_2-score over the thresholds evaluated on the calibration subsets:

$$\tau_k^{\mathrm{F2}} \in \arg\max_{\tau \in \mathcal{T}^{(k)}_{\mathrm{cal}}} F^{(k)}_{2,\mathrm{cal}}(\tau), \tag{23}$$

where $\mathcal{T}^{(k)}_{\mathrm{cal}}$ denotes the set of candidate thresholds evaluated using Normal Calibration and LLM Synthetic Attack Calibration, with synthetic attack sessions treated as the positive class.

For operating point $p \in \{\mathrm{P99}, \mathrm{FN0}, \mathrm{F2}\}$, the session-level decision rule is

$$\hat{y}_j^{(k,p)} = \begin{cases} \text{Alert}, & \text{if } a_j^{(k)} \geq \tau_k^p, \\ \text{Normal}, & \text{if } a_j^{(k)} < \tau_k^p. \end{cases} \tag{24}$$

The three operating points were determined independently for each candidate pipeline because the corresponding anomaly-score distributions differ across the three URL sequence rarity representations. The P99 threshold was designated as the primary operating point because it is determined entirely from normal calibration data and therefore does not require attack labels. FN=0 and F_2-max were retained as auxiliary operating points to show the trade-off obtained when labeled attack-calibration information is incorporated into threshold selection. After determination, all thresholds were frozen before evaluation on Normal Test and LLM Synthetic Attack Test; neither held-out test subset was used for operating-point selection or subsequent threshold adjustment. Fig. 6 summarizes the primary P99 calibration and session-level decision procedure.

D. LLM-BASED ATTACK INTERPRETATION

The Alert sessions identified in the stage are associated with anomaly scores and session-level feature values. However, these numerical outputs alone do not provide sufficient information for security analysts to interpret why a session was classified as anomalous. Therefore, this section presents an optional LLM-based attack interpretation module that combines feature attribution with locally sanitized request

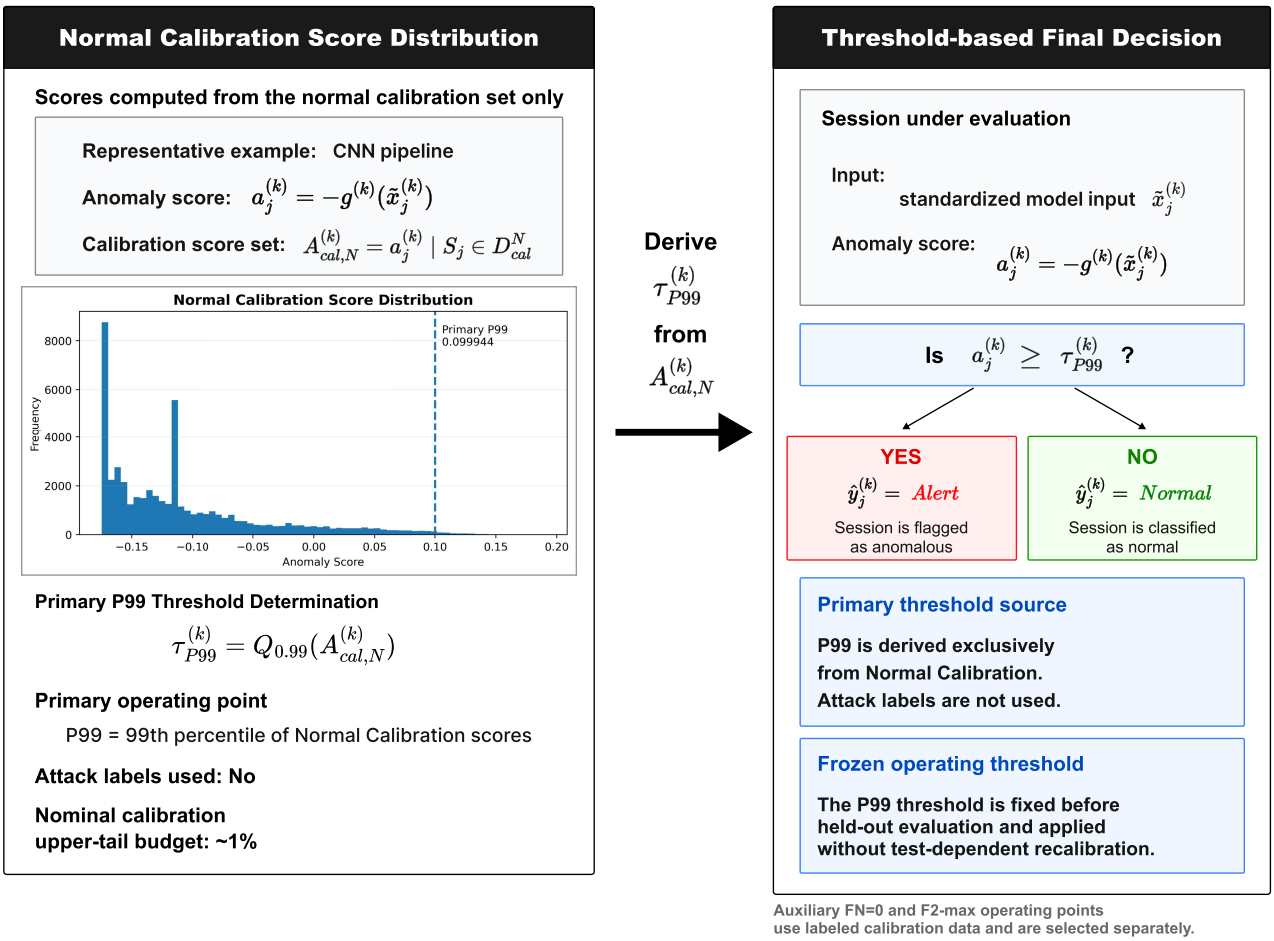


FIGURE 6. Primary P99 threshold calibration and session-level decision process.

evidence and natural-language explanation. Before any external application programming interface (API) invocation, the evidence is processed by a local privacy-filtering and data-minimization stage. The proposed module does not replace the anomaly detection model and does not participate in the Alert decision; it transforms only the sanitized quantitative and request evidence into a human-interpretable security explanation. The core anomaly detection pipeline remains operational without the LLM module.

Specifically, feature attribution is first computed for each Alert session to identify the features with the largest contributions to the anomaly score. The resulting evidence is processed by the local privacy-filtering and data-minimization stage, organized into a sanitized evidence summary, and optionally provided as input to an LLM. Based only on the sanitized evidence summary, the LLM generates a structured attack interpretation. As a result, security analysts can examine the anomaly score, the most influential features, the locally retained request flow, the privacy-filtered evidence supplied to the LLM, the potential attack type, and recommended follow-up actions.

For quantitative evaluation, the LLM interpretation module was applied to all sessions classified as Alerts by the CNN-based pipeline at the frozen primary P99 operating point. The resulting evaluation pool comprised 869 Alerts: 382 true-positive attack sessions from LLM Synthetic Attack Test and 487 false-positive sessions from Normal Test. The external LLM received only the locally sanitized evidence summary described below. The detector decision, anomaly score, and operating threshold were included only as contextual metadata, whereas cohort membership and ground-truth attack labels were not supplied to the LLM. The quantitative experiments used `gemma-4-26b-a4b-it` through the Gemini API with a temperature of 0.2 and the same fixed prompt and output schema for all evaluated cases.

For the 382 true-positive attack Alerts, the primary attack-type output was compared with the six predefined synthetic-attack labels: scanner-like probing, SQL injection, administrator/login brute-force, mixed low-and-slow, path traversal, and XSS. The permitted output ontology additionally included `Normal` and `Unknown` so that the LLM was not forced to assign one of the six attack categories when the

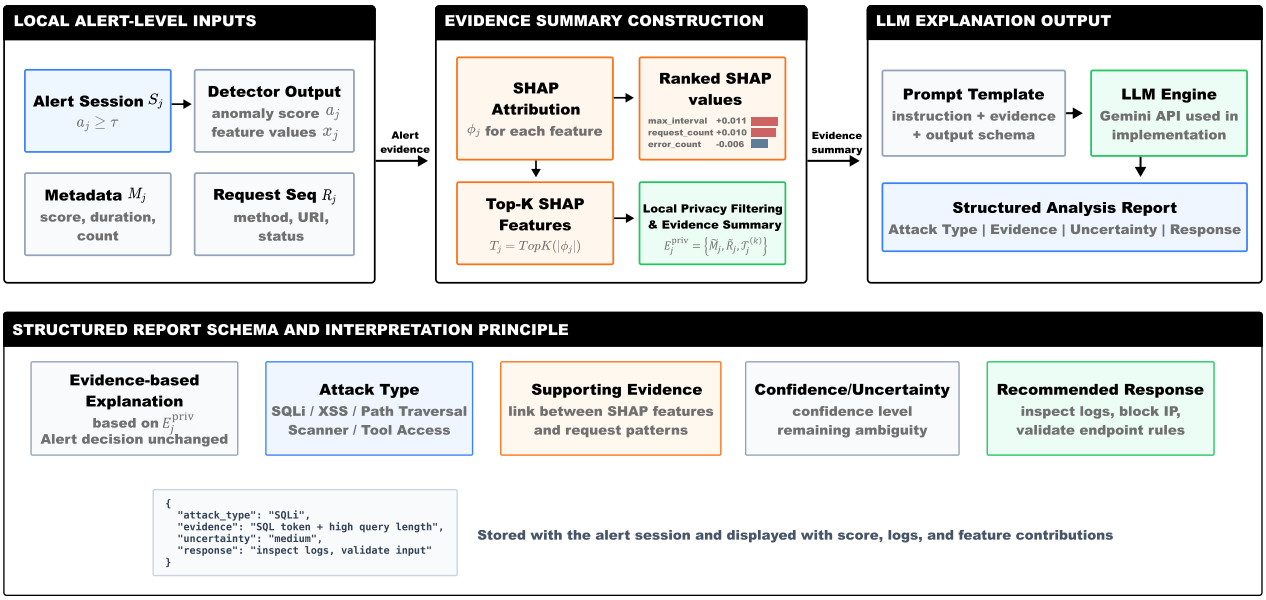


FIGURE 7. Architecture of the optional LLM-based attack interpretation module.

supplied evidence was insufficient. Conditional attack-type classification accuracy was calculated over all 382 true-positive Alerts, and a confusion matrix was constructed between the six ground-truth attack categories and the eight permitted output categories. For the 487 normal-session false-positive Alerts, the proportion for which the LLM returned `Unknown` was evaluated separately as an abstention-related measure; these sessions were not assigned synthetic attack ground-truth labels.

Explanation faithfulness was evaluated on a fixed 100-case subset of the 869 Alert pool. The subset comprised 48 attack true positives and 52 normal false positives. The 48 attack cases were balanced across the six attack categories, with eight cases per category and four Clear and four Moderate cases within each category. Two human raters independently reviewed the same 100 generated explanations against the sanitized evidence supplied to the LLM. Each rater recorded a binary judgment of whether the explanation was fully faithful to the supplied evidence and separately counted the total number of factual statements and the number of factual statements not supported by that evidence. The unsupported-statement rate was calculated for each rater as the number of unsupported factual statements divided by the total number of factual statements. Agreement between the two binary faithfulness judgments was summarized using raw agreement and Cohen's κ.

Run-to-run consistency was evaluated on a separate fixed subset of 30 Alerts comprising 18 attack true positives and 12 normal false positives; the attack subset contained three cases from each of the six attack categories. Each case was generated five times using the same sanitized input, model, temperature, prompt, and output schema. Consistency was summarized using the proportion of cases with identical attack-type outputs in all five generations, the proportion with agreement in at least four of five generations, pooled pairwise agreement among the five outputs, and the agreement of generations 2–5 with the first generation. Fig. 7 summarizes the architecture and evidence flow of the optional LLM-based interpretation module.

1) FEATURE ATTRIBUTION

To interpret anomaly detection results at the feature level, local feature attribution is performed using SHAP. In this study, SHAP is applied to explain the anomaly score defined in Eq. (18). Let $g^{(k)}(\cdot)$ denote the `decision_function` of the IF in the k-th pipeline. The target function explained by SHAP is defined as

$$h^{(k)}(\tilde{\mathbf{x}}) = -g^{(k)}(\tilde{\mathbf{x}}). \tag{25}$$

Thus, $h^{(k)}(\tilde{\mathbf{x}}_j^{(k)}) = a_j^{(k)}$, where a larger value indicates that the corresponding session deviates further from the normal baseline.

Let $\mathcal{F} = \{1, \ldots, D\}$ denote the set of all input features, where $D = 30$, and let $\phi_{j,d}^{(k)}$ represent the SHAP value of the d-th feature for session S_j. Based on the Shapley value formulation in cooperative game theory, the SHAP value is defined as

$$\phi_{j,d}^{(k)} = \sum_{\mathcal{S} \subseteq \mathcal{F} \setminus \{d\}} \frac{|\mathcal{S}|!\,(D - |\mathcal{S}| - 1)!}{D!} \times \left[v_{j,\mathcal{B}}^{(k)}(\mathcal{S} \cup \{d\}) - v_{j,\mathcal{B}}^{(k)}(\mathcal{S}) \right]. \tag{26}$$

Here, $\mathcal{S}$ denotes a subset of features excluding feature d, $\mathcal{B}$ is the background set used for SHAP estimation, and $v_{j,\mathcal{B}}^{(k)}(\mathcal{S})$ represents the coalition value, i.e., the expected anomaly score obtained by fixing the feature values in $\mathcal{S}$ while integrating out the remaining features with respect to the background set.

The SHAP values computed by Kernel SHAP approximately satisfy the following additive property:

$$a_j^{(k)} \approx \phi_{0,\mathcal{B}}^{(k)} + \sum_{d=1}^{D} \phi_{j,d}^{(k)}, \qquad D = 30, \tag{27}$$

where $\phi_{0,\mathcal{B}}^{(k)}$ is the expected anomaly score over the background set, and $\phi_{j,d}^{(k)}$ quantifies the contribution of the d-th feature of session S_j to the anomaly score.

Since the anomaly score in this study is defined such that larger values indicate higher abnormality, a positive SHAP value $\phi_{j,d}^{(k)} > 0$ indicates that the corresponding feature increases the anomaly score, whereas $\phi_{j,d}^{(k)} < 0$ indicates that the feature decreases the anomaly score. The absolute SHAP value, $\left|\phi_{j,d}^{(k)}\right|$, represents the magnitude of the feature contribution regardless of its direction.

Kernel SHAP was applied to the standardized 30-dimensional model input. For the final held-out quantitative evaluation, a fixed background set of 20 instances was randomly sampled from the standardized Normal Train matrix of 231,863 sessions using a fixed random seed of 42. The same background set was used to explain all 869 CNN P99 Alerts, and Kernel SHAP estimation was performed using `nsamples`=100.

For dashboard visualization and LLM-based interpretation, only the top $K = 8$ features with the largest absolute SHAP values are selected. The top feature set is defined as

$$\mathcal{T}_j^{(k)} = \mathrm{TopK}_{d\in\mathcal{F}}\left(\left|\phi_{j,d}^{(k)}\right|\right), \qquad K = 8. \tag{28}$$

The selected top SHAP features are not limited to features that contribute positively to the anomaly score; they include both positive and negative contributions with the largest absolute magnitudes. For interpretation, both the signed SHAP value and its absolute magnitude are retained, while feature values are reported in their original scale before standardization to improve interpretability. The SHAP computation itself is performed in the standardized feature space used by the anomaly detection model.

2) ATTACK EXPLANATION USING LLM

Although SHAP provides quantitative feature-level evidence, it does not explicitly describe how these features relate to the request sequence or how they can be interpreted in terms of potential web attacks. To address this limitation while limiting the disclosure of log data, TraceLog constructs a privacy-filtered evidence summary only for sessions already classified as Alerts. The complete access-log corpus, the normal-baseline data, and the model-training data are not transmitted to the LLM.

For an Alert session S_j, the locally retained session metadata M_j and chronological request sequence R_j are transformed before prompt construction as follows:

$$\begin{aligned} \widetilde{M}_j &= P_M(M_j), \\ \widetilde{R}_j &= P_R(R_j), \\ E_j^{\mathrm{priv}} &= \left\{\widetilde{M}_j, \widetilde{R}_j, \mathcal{T}_j^{(k)}\right\}, \end{aligned} \tag{29}$$

where $P_M(\cdot)$ and $P_R(\cdot)$ denote the locally executed metadata- and request-sanitization functions, respectively. The sanitized metadata $\widetilde{M}_j$ contains the anomaly score, applied threshold, number of requests, selected aggregate session statistics, and a pseudonymous Alert identifier. Raw IP addresses and original session identifiers are excluded. Exact timestamps are replaced with request order or relative elapsed time, and detailed User-Agent strings are omitted or generalized to a browser or automated-tool family.

The sanitized request evidence $\widetilde{R}_j$ contains only the HTTP method, normalized URL path, privacy-filtered query pattern, status code, and request order required to interpret the Alert. Query-parameter values that contain or may reveal personal identifiers, credentials, session tokens, API keys, or other authentication information are removed or replaced with `[REDACTED]`. Attack-relevant structural patterns are retained in masked form only when they are required to interpret the Alert. Cookies, authorization values, and other authentication fields are not included in the external-LLM payload.

The Top-K SHAP set $\mathcal{T}_j^{(k)}$ contains only the selected feature names, aggregate feature values, signed SHAP values, and contribution directions. Consequently, the external LLM receives the minimized per-Alert evidence summary E_j^{priv}, rather than raw log records. Raw logs and direct identifiers remain within the local service boundary.

The sanitized evidence summary is inserted into a predefined prompt template. The prompt instructs the LLM to use only the supplied evidence, not to infer user identities or reconstruct redacted values, and to output `Unknown` when the evidence is insufficient to support an attack category. The prompt also states that the Alert decision has already been made by the anomaly detection model and must not be modified or reclassified by the LLM.

The LLM output is restricted to a structured analysis report containing a possible attack type, the evidence supporting the interpretation, the uncertainty of the interpretation, and recommended follow-up actions. The generated output is auxiliary information for operator review and is not used for automated blocking or anomaly classification.

IV. EXPERIMENTS AND RESULTS

Offline experiments evaluated the three candidate TraceLog pipelines on the internal benchmark and further examined feature ablation, external validation, sessionization sensitivity, statistical significance, computational performance, and the optional LLM interpretation module. Unless otherwise stated, the primary internal evaluation used 49,686 Normal Test sessions and 450 held-out LLM Synthetic Attack Test sessions at pipeline-specific P99 thresholds determined from

Normal Calibration and frozen before evaluation; FN=0 and F_2-max were reported only as auxiliary operating points.

A. EXPERIMENTAL SETUP

The data partitioning, model training, hyperparameter selection, threshold calibration, and held-out evaluation followed the procedures described in Section III.

Model training and performance evaluation were conducted on an HPE DL380 Gen11 compute node in the high-performance computing (HPC) environment of Kumoh National Institute of Technology. The compute node was equipped with an Intel Xeon Gold 6430 central processing unit (CPU) and an NVIDIA H100 NVL graphics processing unit (GPU). Rocky Linux 8.9 was used as the operating system, and the software environment comprised Python 3.10.3, PyTorch 2.5.1, the NVIDIA Compute Unified Device Architecture (CUDA) runtime 12.1, NumPy 2.2.6, pandas 2.3.3, scikit-learn 1.7.2, and Transformers 5.4.0.

The end-to-end performance benchmark was conducted as a CPU-only job in the institutional HPC environment with eight CPU threads and CUDA disabled. It evaluated the final CNN-based pipeline at the frozen primary P99 operating point, with the local detection core and optional post-Alert processing measured separately as detailed in Section IV-K.

B. EVALUATION METRICS

Anomaly detection performance was evaluated using Recall, the false negative (FN) count, Precision, the false-positive rate (FPR), F_1-score, F_2-score, the area under the receiver operating characteristic (ROC) curve (AUC-ROC), and average precision (AP). Recall, Precision, FPR, F_1, and F_2 were used to characterize threshold-dependent performance, while AUC-ROC and AP summarized threshold-independent score separation.

Precision denotes the proportion of sessions classified as Alerts that are actual attack sessions, whereas Recall denotes the proportion of all attack sessions that are correctly detected as Alerts. The F_1-score is the harmonic mean of Precision and Recall, whereas the F_2-score assigns greater weight to Recall. FPR denotes the proportion of normal sessions incorrectly classified as Alerts. Because excessive false positives can increase the analytical burden on operators in operational security-monitoring environments, FPR is an important operational metric. AUC-ROC and AP were used as auxiliary metrics for examining the overall separation performance of anomaly scores without depending on a single threshold. Computational performance was characterized using local-core latency and throughput, sampled peak resident memory, Alert-only SHAP latency, local sanitization and prompt-construction latency, and external-LLM API round-trip time, as described in Section IV-K.

The confusion matrix consists of true positives (TPs), false positives (FPs), true negatives (TNs), and false negatives (FNs). In this study, TP denotes an attack session correctly classified as Alert, FP denotes a normal session incorrectly classified as Alert, TN denotes a normal session correctly classified as Normal, and FN denotes an attack session incorrectly classified as Normal. The primary evaluation metrics are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{30}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{31}$$

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{32}$$

$$F_2 = \frac{5 \times \text{Precision} \times \text{Recall}}{4 \times \text{Precision} + \text{Recall}}, \tag{33}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{34}$$

The ROC curve consists of the pairs $(\text{FPR}(\tau), \text{TPR}(\tau))$ obtained by varying all possible thresholds for the anomaly score, where the true-positive rate (TPR) at threshold τ, $\text{TPR}(\tau)$, is equivalent to Recall. AUC-ROC is defined as the area under the ROC curve and was used to evaluate the separation performance of anomaly scores between normal and attack sessions without depending on a single threshold [22].

AP is calculated by weighting each increase in recall obtained as the anomaly-score threshold is varied by the precision at that point, thereby summarizing the precision–recall relationship in a single value [23]. In this study, AP was calculated using the `average_precision_score` function in scikit-learn. AP was used together with AUC-ROC to evaluate separation performance for the positive class in a class-imbalanced environment in which the proportion of attack sessions was low [24].

C. THRESHOLD CALIBRATION AND OPERATING-POINT SELECTION

Following the procedure described in Section III-C6, the pipeline-specific P99, FN=0, and F_2-max thresholds were determined from their corresponding calibration subsets and frozen before held-out evaluation. The resulting P99 thresholds were 0.109681, 0.096927, and 0.099944 for the Markov-, LogBERT-, and CNN-based pipelines, respectively. P99 was used for the primary comparison, whereas FN=0 and F_2-max were retained as auxiliary attack-label-dependent operating points. The corresponding held-out anomaly-score distributions and P99 thresholds are shown in Fig. 8.

D. OVERALL DETECTION PERFORMANCE

At P99, the CNN-based pipeline achieved the highest Recall (0.8489) and F_2-score (0.7156), with an FPR of 0.0098; the corresponding Recall and FPR values were 0.8400 and 0.0101 for Markov and 0.8022 and 0.0101 for LogBERT, respectively. CNN also achieved the highest AUC-ROC (0.9960) and AP (0.7415). Complete P99 results, including FN, FP, Precision, and F_1, are reported in Table 4.

Panel B of Table 4 confirms that the predefined Clear and Moderate conditions produced measurably different evidence densities in the realized benchmark. Clear sessions

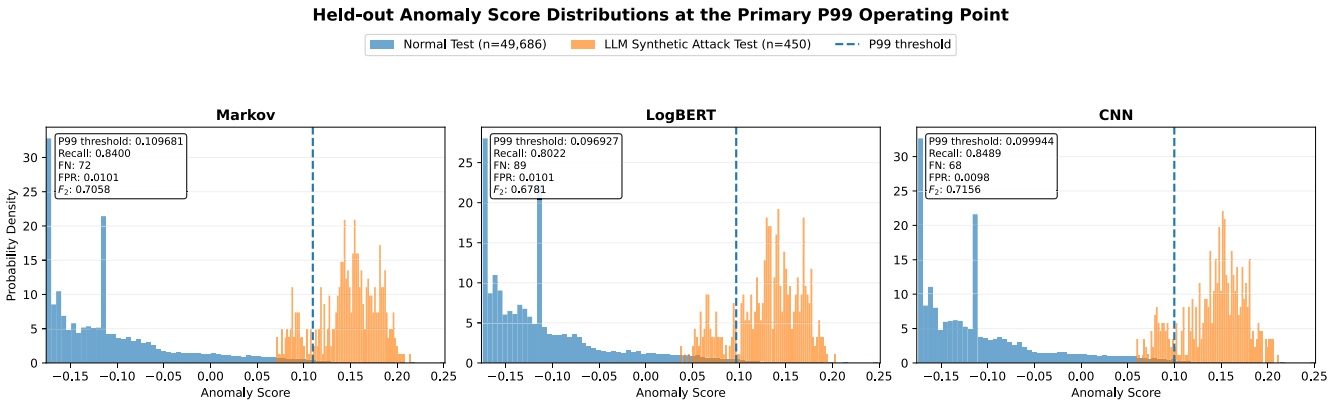


FIGURE 8. Anomaly-score distributions of the three candidate pipelines at the primary P99 operating point.

TABLE 4. Internal synthetic benchmark characterization and held-out detection performance.

Panel A. Held-out detection performance across operating points

Pipeline	Operating point	Threshold	AUC-ROC	AP	Recall	FN	FP	Precision	FPR	F_1	F_2
Markov + IF	FN=0	0.063987	0.9954	0.7020	1.0000	0	1,893	0.1921	0.0381	0.3222	0.5431
Markov + IF	P99	0.109681	0.9954	0.7020	0.8400	72	500	0.4305	0.0101	0.5693	0.7058
Markov + IF	F_2-max	0.122898	0.9954	0.7020	0.7800	99	259	0.5754	0.0052	0.6623	0.7282
LogBERT + IF	FN=0	0.027028	0.9937	0.6506	1.0000	0	3,096	0.1269	0.0623	0.2253	0.4210
LogBERT + IF	P99	0.096927	0.9937	0.6506	0.8022	89	501	0.4188	0.0101	0.5503	0.6781
LogBERT + IF	F_2-max	0.106115	0.9937	0.6506	0.7644	106	316	0.5212	0.0064	0.6198	0.6992
CNN + IF	FN=0	0.053102	0.9960	0.7415	1.0000	0	1,999	0.1837	0.0402	0.3105	0.5295
CNN + IF	P99	0.099944	0.9960	0.7415	0.8489	68	487	0.4396	0.0098	0.5792	0.7156
CNN + IF	F_2-max	0.118177	0.9960	0.7415	0.7822	98	206	0.6308	0.0041	0.6984	0.7464

Panel B. Realized Clear and Moderate benchmark characteristics, median [Q1–Q3]

Metric	Clear ($n = 500$)	Moderate ($n = 400$)
Requests per session	14.000 [11.000–25.000]	11.000 [9.000–17.000]
Suspicious keyword count	13.000 [0.000–18.000]	7.000 [1.000–10.000]
Suspicious-keyword evidence per request	1.172 [0.000–1.455]	0.809 [0.062–1.129]
Security evidence sum	18.500 [14.000–35.250]	12.000 [8.000–24.000]
Security evidence per request	1.500 [1.153–1.833]	1.222 [0.797–1.656]

Values in Panel B are median [Q1–Q3], where Q1 and Q3 denote the 25th and 75th percentiles, respectively. "Evidence per request" denotes the density of security-related evidence captured by the feature extractor and should not be interpreted as the fraction of requests with attack ground-truth labels.

generally contained more requests and denser security-related evidence, whereas Moderate sessions contained fewer requests and lower evidence density. Nevertheless, as shown below, this descriptive difficulty distinction had only a small effect on aggregate P99 Recall; the observed variation in P99 Recall was larger across attack categories, particularly for path traversal and XSS, than between the Clear and Moderate subsets. The auxiliary operating points further illustrate the recall–false-positive trade-off. FN=0 achieved a Recall of 1.0000 for all three pipelines but increased held-out FPR to 0.0381–0.0623, whereas F_2-max reduced FPR to 0.0041–0.0064 with a Recall of 0.7644–0.7822. CNN achieved the highest F_2-score at this auxiliary operating point (0.7464); the complete operating-point results are reported in Table 4.

Performance at P99 also varied across attack categories. For the CNN-based pipeline, Recall remained 1.0000 for administrator/login brute-force, mixed low-and-slow, and scanner-like probing, and was 0.9620 for SQL injection. Lower Recall was observed for path traversal (0.5606) and XSS (0.4783). Markov showed a similar pattern, with Recall values of 1.0000 for administrator/login brute-force, mixed low-and-slow, and scanner-like probing, 0.9620 for SQL injection, 0.5303 for path traversal, and 0.4493 for XSS. LogBERT achieved 1.0000 for administrator/login brute-force, mixed low-and-slow, and scanner-like probing, 0.9494 for SQL injection, 0.3333 for path traversal, and 0.4058 for XSS.

The Clear and Moderate subsets showed similar aggregate behavior at P99. CNN achieved Recall values of approximately 0.848 and 0.850 on Clear and Moderate sessions, respectively; the corresponding values were 0.840 and 0.840 for Markov and 0.792 and 0.815 for LogBERT. Thus, the principal variation in held-out detection at the primary operating point was associated more strongly with attack category than with the predefined Clear/Moderate distinction, with path-traversal and XSS sessions accounting for most of the missed attacks.

E. FEATURE ABLATION ANALYSIS

To examine the contributions of the three feature groups defined in Section III-C3, we conducted a

TABLE 5. Feature-ablation results on the internal benchmark and Org Y.

Dataset	Configuration	AUC-ROC	AP	P99 Recall	P99 F_2	F_2-max F_2
Internal	G_S	0.9592	0.1296	0.2511	0.2356	0.3659
Internal	G_R	0.9997	0.9640	1.0000	0.8007	0.9481
Internal	G_U	0.9336	0.1990	0.3444	0.3209	0.3233
Internal	$G_S + G_R$	0.9938	0.6171	0.7911	0.6730	0.6710
Internal	$G_S + G_U$	0.9565	0.1024	0.0667	0.0654	0.3555
Internal	$G_R + G_U$	0.9998	0.9803	1.0000	0.7965	0.9453
Internal	$G_S + G_R + G_U$ (Full)	0.9960	0.7415	0.8489	0.7156	0.7464
Org Y	G_S	0.8009	0.5932	0.1549	0.1848	0.6024
Org Y	G_R	0.8945	0.7289	0.3816	0.4320	0.6917
Org Y	G_U	0.7484	0.3575	0.0263	0.0321	0.5941
Org Y	$G_S + G_R$	0.8330	0.6950	0.3650	0.4146	0.7145
Org Y	$G_S + G_U$	0.8064	0.6058	0.1680	0.1999	0.6035
Org Y	$G_R + G_U$	0.8725	0.6937	0.3505	0.3997	0.6659
Org Y	$G_S + G_R + G_U$ (Full)	0.8361	0.6938	0.3526	0.4016	0.7098

TABLE 6. CNN-based external validation on Org Y.

Protocol	Operating point	Threshold	Precision	Recall	FPR	F_1	F_2	AUC-ROC	AP
Zero-shot	Internal FN=0	0.053102	0.5755	0.7955	0.1718	0.6679	0.7390	0.8637	0.6496
Rebuilt	FN=0	-0.184174	0.2266	1.0000	0.9991	0.3695	0.5943	0.8361	0.6938
Rebuilt	P99	0.121528	0.9040	0.3526	0.0110	0.5073	0.4016	0.8361	0.6938
Rebuilt	F_2-max	-0.125701	0.4663	0.8165	0.2736	0.5936	0.7098	0.8361	0.6938

seven-configuration ablation analysis using G_S, G_R, G_U, $G_S + G_R$, $G_S + G_U$, $G_R + G_U$, and $G_S + G_R + G_U$. Table 5 summarizes the results on both the internal benchmark and Org Y. The CNN-based URL sequence rarity feature was used for G_U. For each configuration, the StandardScaler and IF were fitted using the corresponding normal-training features, while the primary P99 operating threshold was determined exclusively from the corresponding Normal Calibration anomaly-score distribution and frozen before held-out evaluation. The auxiliary F_2-max operating point was also evaluated to characterize the feature-group behavior under an attack-label-dependent calibration criterion.

On the internal benchmark, the security-rule-derived configurations showed particularly strong performance; however, this result must be interpreted in light of the structural relationship between the internal normal-data refinement rules and the security-rule-derived features described in Section III-B2. Adding CNN-based URL sequence rarity to the $G_S + G_R$ baseline provided a modest improvement, increasing AP from 0.6171 to 0.7415, P99 Recall from 0.7911 to 0.8489, and the P99 F_2-score from 0.6730 to 0.7156.

This improvement was not reproduced on Org Y. Adding URL rarity to $G_S + G_R$ changed AP from 0.6950 to 0.6938, P99 Recall from 0.3650 to 0.3526, and the P99 F_2-score from 0.4146 to 0.4016. The general session/statistical feature group also behaved differently across the two datasets: adding G_S to $G_R + G_U$ reduced the auxiliary F_2-max performance internally but improved it on Org Y. Thus, no single configuration dominated all reported metrics across both datasets, and the incremental contribution of URL sequence rarity is best characterized as modest and dataset-dependent.

F. EXTERNAL VALIDATION

To evaluate cross-service behavior beyond the controlled internal synthetic benchmark, we applied the CNN-based TraceLog pipeline to the independent Org Y dataset described in Section III-B4. The held-out evaluation comprised 10,124 Normal Test sessions and 2,964 dataset-provider-defined Attack Test sessions, whose labels were derived independently of the TraceLog rules used for internal normal-data refinement.

Using the two protocols defined in Section III-B4, Table 6 shows that strict zero-shot transfer produced substantially higher false-positive behavior despite retaining considerable attack sensitivity, whereas service-specific rebuilding restored a low-FPR regime at the P99 threshold calibrated on Org Y but produced substantially lower Recall than on the internal benchmark.

The auxiliary operating points further exposed the recall–false-positive trade-off: forcing FN=0 caused nearly all held-out normal sessions to be flagged, whereas the F_2-max operating point increased Recall at the cost of a much higher FPR than P99. Thus, neither direct transfer nor service-specific rebuilding reproduced the internal benchmark behavior unchanged. Although zero-shot AUC-ROC was numerically higher than rebuilt AUC-ROC, possible effects of heterogeneous Org Y log sources were not evaluated separately and are not treated as a confirmed explanation. Overall, the results demonstrate substantial cross-service distribution shift.

G. SENSITIVITY TO SESSION INACTIVITY TIMEOUT

To evaluate the sensitivity of TraceLog to the session inactivity timeout, we re-sessionized the complete public access log of 10,365,152 requests using inactivity timeouts

TABLE 7. CNN-based sensitivity to session inactivity timeout.

Timeout	Reconstructed sessions	Normal Test	FP	FPR	Source-session Recall
10 min	399,252	55,567	177	0.0032	0.8489
15 min	382,814	53,220	215	0.0040	0.8489
30 min	357,825	49,686	487	0.0098	0.8489
60 min	336,465	46,652	1,085	0.0233	0.8489

TABLE 8. CNN-based sensitivity to client-key grouping.

Client key	Reconstructed sessions	Normal Test	FP	FPR	Derived attack sessions	Source-session Recall
IP+UA	357,825	49,686	487	0.0098	450	0.8489
IP-only	337,979	46,823	448	0.0096	450	0.8489
UA-only	141,717	19,722	448	0.0227	438	0.8556

of 10, 15, 30, and 60 min. The analysis was restricted to the CNN-based pipeline and the primary P99 operating policy, following the final evaluation protocol. The resulting numbers of reconstructed sessions were 399,252, 382,814, 357,825, and 336,465 for the 10-, 15-, 30-, and 60-min settings, respectively.

For each timeout condition, normal-data refinement and session-level partitioning were performed separately. The CNN rarity extractor was trained using the corresponding Normal Train subset, and a condition-specific Standard-Scaler and IF were subsequently fitted using the resulting 30-dimensional training inputs. The operating threshold was then determined independently as the 99th percentile of the corresponding Normal Calibration anomaly-score distribution. Thus, this experiment evaluates the complete CNN-based TraceLog configuration adapted to each sessionization policy rather than applying a single model and numerical threshold unchanged across different session definitions.

The held-out normal-test sizes were 55,567, 53,220, 49,686, and 46,652 sessions for inactivity timeouts of 10, 15, 30, and 60 min, respectively. The corresponding numbers of false positives were 177, 215, 487, and 1,085, yielding held-out FPRs of 0.0032, 0.0040, 0.0098, and 0.0233. The 30-min configuration reproduced the primary CNN result reported in Section IV-D, with 487 false positives among 49,686 Normal Test sessions.

Attack-side evaluation used the same 450 source synthetic attack sessions for all four timeout settings. The source-session-weighted Recall remained 0.8489 (382/450) for the 10-, 15-, 30-, and 60-min conditions. Therefore, within the present controlled attack benchmark, varying the inactivity timeout did not change the number of source attack sessions detected by the condition-specific CNN pipelines. This result should not be interpreted as evidence that attack detection is generally invariant to session boundaries, because the synthetic benchmark does not cover arbitrary attack flows whose semantic boundaries may change substantially under alternative timeout settings. The timeout-sensitivity results are summarized in Table 7.

Normal-session behavior was more sensitive to the timeout setting than source-session Recall: longer timeouts produced fewer reconstructed sessions and higher held-out FPR. Thus, the inactivity timeout materially affects both the evaluated session population and false-positive behavior.

H. CLIENT-KEY GROUPING STRESS TEST

To evaluate sensitivity to the client-key definition independently of the inactivity timeout, we fixed the inactivity timeout at 30 min and reconstructed the complete public access log using three grouping keys: IP address plus a User-Agent (UA) string (IP+UA), IP address only (IP-only), and UA only (UA-only). The resulting numbers of reconstructed sessions were 357,825, 337,979, and 141,717 for IP+UA, IP-only, and UA-only, respectively.

Following the same CNN-based, condition-specific rebuilding protocol and primary P99 operating policy used in Section IV-G, each client-key condition was independently refitted and assigned a P99 threshold from its own Normal Calibration subset before held-out evaluation.

Under the baseline IP+UA condition, the held-out Normal Test set contained 49,686 sessions and produced 487 false positives, corresponding to an FPR of 0.0098. The IP-only condition produced 46,823 held-out normal sessions, of which 448 were classified as Alerts, yielding an FPR of 0.0096. The UA-only condition produced 19,722 held-out normal sessions and also yielded 448 false positives; because of the substantially smaller denominator, its FPR increased to 0.0227. These results indicate that removing User-Agent information while retaining IP produced normal-traffic false-positive behavior close to the baseline, whereas grouping by User-Agent alone resulted in substantially greater aggregation and a higher false-positive rate. The results across the three client-key definitions are summarized in Table 8.

Attack-side behavior was evaluated using source-session-weighted Recall with the denominator preserved at the original 450 synthetic attack sessions. The baseline IP+UA and IP-only conditions both detected 382 of the 450 source attack sessions, corresponding to a Recall of 0.8489. Under UA-only grouping, the 450 source sessions were reconstructed into 438 derived sessions because requests sharing User-Agent values could be merged; source-session-weighted evaluation detected 385 of the 450 source sessions, corresponding to a Recall of 0.8556. The small numerical

TABLE 9. Pairwise McNemar test results with Holm correction.

Comparison	n_{10}	n_{01}	χ^2	Raw p-value	Holm-adjusted p-value
Markov–LogBERT	100	82	1.5879	0.207625	0.330654
Markov–CNN	58	75	1.9248	0.165327	0.330654
LogBERT–CNN	59	94	7.5556	0.005983	0.017948

TABLE 10. Pairwise DeLong test results with Holm correction.

Comparison	AUC Difference	z	Raw p-value	Holm-adjusted p-value
Markov–LogBERT	0.001768	10.3991	2.504×10^{-25}	5.007×10^{-25}
Markov–CNN	-0.000593	-9.7189	2.505×10^{-22}	2.505×10^{-22}
LogBERT–CNN	-0.002362	-12.3856	3.128×10^{-35}	9.384×10^{-35}

increase in Recall under UA-only should not be interpreted as evidence that UA-only grouping improves attack detection, because the grouping policy substantially changes the session boundaries and merges activity originating from distinct source sessions.

The UA-only result indicates substantial over-aggregation among requests sharing common User-Agent strings. Accordingly, IP+UA is retained as the baseline client-key definition because it was used in the main experiments, not because this stress test establishes it as universally optimal. This stress test isolates sensitivity to the grouping key available in the present logs; it does not reproduce controlled NAT collisions, recover ground-truth user identity, or simulate adversarial User-Agent spoofing or rotation. Such identity-grounded scenarios require additional identifiers or controlled experimental data and are therefore treated as limitations rather than inferred from the present grouping experiment.

I. COMPARISON OF URL SEQUENCE RARITY METHODS

Although the three candidate pipelines share the same 29-dimensional session-level feature vector, StandardScaler-based preprocessing, and IF-based downstream detector, they differ in their URL sequence rarity representations as well as in their model-specific training objectives, model-state selection procedures, and inference mechanisms. Accordingly, the observed differences are interpreted at the complete-pipeline level rather than as the isolated causal effect of the rarity representation alone.

Based on the predefined primary P99 comparison summarized in Table 4, the CNN-based pipeline was retained for the subsequent final-pipeline analyses. This selection is specific to the present data and evaluation protocol and does not establish universal superiority of the CNN rarity representation.

J. STATISTICAL SIGNIFICANCE ANALYSIS

To compare classification errors between pipelines at their primary P99 operating points, the McNemar test was applied to paired classification outcomes for the same 50,136 held-out sessions, comprising 49,686 sessions from Normal Test and 450 sessions from LLM Synthetic Attack Test [25]. The test statistic was calculated using the continuity-corrected chi-square approximation with one degree of freedom. Here, n_{10} denotes the number of sessions correctly classified by the first pipeline but misclassified by the second pipeline, whereas n_{01} denotes the number of sessions misclassified by the first pipeline but correctly classified by the second pipeline. The Holm procedure was applied to adjust for the three pairwise comparisons [26]. All tests were two-sided with a significance level of $\alpha = 0.05$. The pairwise McNemar results are summarized in Table 9.

At P99, the Markov–CNN and Markov–LogBERT comparisons were not statistically significant after Holm correction, whereas CNN–LogBERT remained significant (Holm-adjusted $p = 0.017948$; Table 9).

Because the P99 evaluation produces both FN and FP differences, the McNemar analysis may include discordant pairs arising from both the 49,686 held-out normal sessions and the 450 held-out attack sessions.

To compare threshold-independent anomaly-score separation, AUC-ROC was evaluated from the continuous anomaly scores of the same 50,136 held-out sessions. The AUC-ROC values were 0.9954 for Markov, 0.9937 for LogBERT, and 0.9960 for CNN. Pairwise DeLong tests were used to compare the correlated ROC curves produced by the three pipelines [27], and the Holm procedure was again applied to the three pairwise comparisons [26]. The AUC differences reported in Table 10 are defined as the unrounded AUC estimate of the first pipeline minus that of the second pipeline.

All three pairwise AUC-ROC differences remained statistically significant after Holm correction. CNN achieved the highest AUC-ROC, followed by Markov and LogBERT. However, the absolute differences were small: based on the reported rounded AUC values, CNN exceeded Markov by approximately 0.0006, Markov exceeded LogBERT by approximately 0.0017, and CNN exceeded LogBERT by approximately 0.0023. Thus, the statistical significance of the DeLong comparisons should not be interpreted as evidence of large practical separation between the pipelines. The DeLong results are therefore considered together with the P99 Recall, FPR, precision–recall performance, and the McNemar results when interpreting differences among the candidate pipelines.

K. LATENCY ANALYSIS

The latency analysis focused on the CNN-based pipeline at the primary P99 operating point, consistent with the final end-to-end evaluation protocol. The frozen held-out input contained 856,260 normal-log requests and 7,032 synthetic-attack requests, which were reconstructed into 49,686 Normal Test sessions and 450 Attack Test sessions, respectively. The resulting 50,136-session pipeline reproduced the final P99 classification outcome of 487 FPs, 49,199 TNs, 382 TPs, and 68 FNs in every timed repetition.

The local core measurement covered the processing path from raw HTTP access-log records through parsing, 30-min IP+UA sessionization, session-level feature extraction, CNN URL-rarity computation, standardization, IF scoring, and P99 threshold application. Model and scaler loading were excluded because these artifacts are loaded before inference. The core pipeline was measured five times under the CPU-only configuration described in Section IV-A, without model retraining, refitting, threshold recalibration, or test-dependent adjustment. SHAP computation, local privacy sanitization and external-LLM request construction, and external-LLM network communication were excluded from the core timing and measured separately because they occur only after a session has already been classified as an Alert. Memory is reported as resident-set size (RSS) in mebibytes (MiB), and external latency is reported as application programming interface round-trip time (RTT).

TABLE 11. CPU-only end-to-end performance of the final CNN-based TraceLog pipeline.

Metric	Value
Held-out sessions	50,136
Median local-core processing time	256.450 s
Latency per session	5.115 ms
Throughput	195.500 sessions/s
Parsed-request throughput	3,366.323 requests/s
Sampled peak RSS	2,853.191 MiB

1) END-TO-END PERFORMANCE EVALUATION

Table 11 summarizes the final CPU-only end-to-end benchmark of the CNN-based TraceLog pipeline on the complete 50,136-session held-out cohort. Across five repeated executions, the median local-core processing time was 256.450 s, corresponding to approximately 5.115 ms per session, 195.500 sessions/s, and 3,366.323 parsed log requests/s. The sampled peak resident-set size during the benchmark was 2,853.191 MiB. These measurements characterize the implemented CPU-only pipeline and should not be interpreted as hardware-independent latency estimates.

Table 12 summarizes the post-Alert processing costs. At the frozen P99 threshold, 869 sessions were classified as Alerts. Kernel SHAP computation for all 869 Alerts required 18.323 s in total, corresponding to approximately 21.085 ms per Alert. Local privacy sanitization and construction of the external-LLM request payloads required an additional 0.668 s for the same 869 Alerts, or approximately 0.768 ms per Alert. Neither measurement includes network communication with the external LLM.

External-LLM network latency was measured separately using the 869 successful Alert-level requests used in the quantitative interpretation experiment. The mean API round-trip time was 9.869 s, the median was 9.050 s, and the 95th percentile was 10.753 s. The observed standard deviation was 6.052 s, with successful-request latencies ranging from 6.912 to 86.885 s. These values characterize the external service observed during the experiment and are not intrinsic latency properties of TraceLog.

The measurements show a clear separation between the local anomaly detection path and the optional external interpretation path. The local CNN-based detector processes a session on the order of milliseconds once the relevant log records are available, whereas generation of an external natural-language interpretation introduces latency on the order of seconds. Because the LLM is invoked only after the local Alert decision and does not participate in anomaly scoring or classification, external API latency does not delay or alter the core detection decision. In deployment, however, the time required to finalize an active session under the inactivity-timeout rule and additional storage, messaging, and dashboard overhead must also be considered when estimating the complete time from log arrival to operator-visible interpretation.

TABLE 12. Post-alert processing latency of the final CNN-based TraceLog pipeline.

Metric	Value
P99 Alerts	869
Kernel SHAP, total	18.323 s
Kernel SHAP	21.085 ms/Alert
Privacy sanitization and request construction, total	0.668 s
Privacy sanitization and request construction	0.768 ms/Alert
External LLM API RTT, mean	9.869 s
External LLM API RTT, median	9.050 s
External LLM API RTT, 95th percentile	10.753 s
External LLM API RTT, standard deviation	6.052 s
External LLM API RTT, range	6.912–86.885 s

L. QUANTITATIVE EVALUATION OF LLM-BASED ATTACK INTERPRETATION

The LLM produced a valid structured response for all 869 CNN P99 Alerts. Among the 382 true-positive attack Alerts, the primary attack type matched the predefined synthetic-attack label for 348 sessions, corresponding to a conditional attack-type classification accuracy of 0.9110 (91.10%) among the detected attack Alerts. Conditional classification performance among detected attacks was perfect for scanner-like probing (108/108), administrator/login brute-force (74/74), path traversal (37/37), and XSS (33/33). SQL injection was correctly identified in 72 of 76 cases (94.74%), whereas mixed low-and-slow was correctly identified in 24 of 54 cases (44.44%). The

TABLE 13. LLM attack-type confusion matrix and per-class recall for true-positive alerts.

Ground Truth	Predicted Attack Type								Per-Class
	Scan.	SQLi	Admin BF	Mixed/L&S	Path Trav.	XSS	Normal	Unknown	Recall
Scanner-like probing	108	0	0	0	0	0	0	0	1.0000
SQL injection	4	72	0	0	0	0	0	0	0.9474
Administrator/login brute-force	0	0	74	0	0	0	0	0	1.0000
Mixed low-and-slow	30	0	0	24	0	0	0	0	0.4444
Path traversal	0	0	0	0	37	0	0	0	1.0000
XSS	0	0	0	0	0	33	0	0	1.0000
					Conditional attack-type accuracy on detected attacks				0.9110

Scan. denotes scanner-like probing; SQLi, SQL injection; Admin BF, administrator/login brute-force; Mixed/L&S, mixed low-and-slow; and Path Trav., path traversal.

TABLE 14. Illustrative LLM interpretation for a verified CNN P99 alert.

Alert ID	Decision / Operating point	Locally observed / sanitized request evidence	Top features ranked by absolute SHAP value	LLM interpretation
A000597	Alert; CNN + IF, P99, $\tau = 0.099944$, score $= 0.127888$	The session contained 13 GET requests over 168 s. The sanitized evidence contained multiple traversal-related URI paths and query parameters represented using [TRAVERSAL] or [REDACTED]. One request returned HTTP 200, six returned HTTP 403, and six returned HTTP 404, resulting in error_4xx_ratio=0.9231. The detailed User-Agent and direct identifiers were not included in the external-LLM payload.	distinct_error_url_count (+0.059260), error_4xx_ratio (+0.053220), notfound_404_count (+0.036942), cnn_rarity (+0.034739), forbidden_403_count (+0.027820), special_char_ratio_avg (+0.008211), request_count (+0.003297), max_query_length (+0.001795)	Based on the sanitized evidence, the LLM identified *Path Traversal* as the primary attack type and *Scanner-Like Probing* as a secondary attack type, with a self-assessed confidence of 0.95. It cited the traversal-marked URI patterns, high 4xx-error ratio, large number of distinct error URLs, and traversal-related query parameters as supporting evidence.

corresponding confusion matrix and per-class Recall are reported in Table 13.

The 34 attack-type errors were concentrated in two categories. Four SQL-injection sessions and 30 mixed low-and-slow sessions were interpreted as scanner-like probing. Thus, most explicit single-pattern categories were identified consistently in the present controlled benchmark, whereas the heterogeneous mixed low-and-slow category remained substantially more difficult for the LLM to distinguish from scanner-like probing. These results characterize agreement with the predefined synthetic scenario labels and should not be interpreted as validation against operational attack ground truth.

Among the 487 normal-session false-positive Alerts, the LLM returned Unknown for 20 sessions, corresponding to an Unknown ratio of 0.0411 (4.11%). The relatively low abstention rate indicates that the LLM frequently produced a more specific interpretation even for sessions that were false positives of the anomaly detector.

The five-generation consistency analysis showed that 29 of the 30 repeated cases produced the same primary attack-type output in all five generations, corresponding to a unanimous consistency rate of 96.67%; the pooled pairwise agreement across generations was 98.00%. Agreement of generations 2–5 with the first generation was 97.50%. Stratified by detector outcome, all 18 attack true-positive cases (100.00%) and 11 of 12 normal false-positive cases (91.67%) were unanimous across all five generations. Thus, the attack-type field remained highly repeatable under fixed inputs and generation settings, with one normal false-positive case producing different labels across the five generations.

In the 100-case human evaluation, Rater 1 judged 90 of 100 explanations as fully faithful to the supplied sanitized evidence, whereas Rater 2 judged 67 of 100 as fully faithful. Rater 1 identified 11 unsupported factual statements among 1,138 factual statements, corresponding to an unsupported-statement rate of 0.97%; Rater 2 identified 48 unsupported statements among 1,497 factual statements, corresponding to a rate of 3.21%. The two raters agreed on the binary faithfulness judgment for 77 of the 100 cases, yielding a raw agreement of 77.0% and Cohen's $\kappa = 0.368$. These results show that both case-level faithfulness judgments and statement-level unsupported-statement rates differed between the two raters.

M. QUALITATIVE ANALYSIS OF LLM-BASED ALERT INTERPRETATION

Table 14 presents one verified CNN P99 Alert to illustrate how a completed anomaly detection decision, locally sanitized request evidence, and signed SHAP contributions are connected to the optional LLM interpretation stage. The case belongs to the held-out LLM Synthetic Attack Test set, and its ground-truth attack label was not supplied to the LLM.

In this example, the CNN-based pipeline classified the session as an Alert before any LLM processing, after which only locally sanitized evidence and the Top-K signed SHAP contributions were provided to the external LLM. The positive SHAP values shown in Table 14 indicate features that increased the anomaly score relative to the SHAP background expectation, although the final decision was determined by the complete IF score rather than by any individual attribution. The resulting Path Traversal interpretation is presented only as an illustration of the evidence flow; quantitative reliability is evaluated separately in Section IV-L, and the LLM output does not modify the Alert decision.

V. DISCUSSION

Overall, the results show that TraceLog's behavior depends on operating-point selection, feature composition, sessionization, and service-specific traffic distributions, while the optional LLM module provides supplementary interpretation rather than independent confirmation of an attack. The following discussion examines these findings in terms of detection performance, URL-sequence rarity, operational implications, interpretability, and remaining limitations.

A. INTERPRETATION OF DETECTION PERFORMANCE

The internal results show that comparative performance depends on the operating policy rather than on threshold-independent separation alone. The normal-calibration-based P99 policy was therefore used as the primary comparison because it preserves an approximately 1% false-positive regime without using attack labels for threshold selection, whereas the auxiliary FN=0 and F_2-max operating points illustrate alternative recall–false-positive trade-offs.

The paired statistical analyses provide additional context for these differences. McNemar testing showed that not all P99 detection differences were statistically significant, whereas DeLong testing identified significant AUC-ROC differences among the three pipelines despite small absolute effect sizes. These findings indicate that statistical significance should be interpreted together with effect magnitude and operating-point behavior rather than as evidence of large practical differences by itself.

The external validation on Org Y (Section IV-F) further demonstrates that operating behavior is service-dependent and that performance metrics should not be assumed to transfer directly across environments. Accordingly, deployment requires target-service baseline construction and calibration. The negative threshold observed at the auxiliary FN=0 operating point on Org Y, together with the severe false-positive inflation, further supports treating FN=0 as a diagnostic auxiliary operating point rather than as the primary deployment policy.

B. INTERPRETATION OF URL SEQUENCE RARITY METHODS

The feature-ablation results provide a more direct assessment of the contribution of URL sequence rarity. Adding CNN-based URL rarity to the $G_S + G_R$ representation improved several metrics on the internal benchmark, but the corresponding addition provided no improvement on Org Y. Accordingly, the contribution of URL sequence rarity is characterized as modest and dataset-dependent rather than uniformly beneficial.

The full $G_S + G_R + G_U$ representation is retained because it is the predefined TraceLog representation used in the main experiments, not because the ablation establishes it as the best-performing feature subset. The ablation was introduced to quantify the contribution of the predefined feature groups rather than to perform feature selection. Replacing the main pipeline retrospectively with the best-performing reduced configuration based on held-out ablation results would constitute post hoc model selection; a dedicated feature-selection study would require subset selection using training or calibration data followed by evaluation on an independent test set. Broader conclusions regarding the contribution of sequence rarity therefore require repeated evaluation across additional services and attack populations.

C. OPERATIONAL IMPLICATIONS OF THE PROPOSED FRAMEWORK

The sensitivity analyses show that sessionization policy affects both the reconstructed session population and false-positive behavior, indicating that sessionization settings should be calibrated for the target service.

The latency results further support separating local anomaly detection from optional external interpretation in deployment. TraceLog should therefore be viewed as an auxiliary session-prioritization layer whose operational use must account for alert volume, privacy constraints, and interpretation latency.

D. INTERPRETABILITY AND THE ROLE OF THE LLM MODULE

The quantitative LLM evaluation should be interpreted as conditional classification agreement on detector-selected Alerts rather than as independent evidence that an attack occurred. Although attack-type agreement was high for several synthetic categories, performance was weaker for mixed low-and-slow, and the LLM frequently produced specific attack-oriented interpretations for detector false positives. Repeated-generation analysis showed high consistency, whereas the human faithfulness judgments varied between the two raters. These findings support treating the LLM output as advisory rather than as confirmation of an Alert. Because the sanitization procedure intentionally preserves coarse attack-relevant structural markers in masked form to support interpretation, the conditional attack-type classification result should not be interpreted as a blind attack-classification benchmark independent of the sanitization scheme.

Because the optional interpretation path relies on externally processed evidence, privacy and data-governance considerations remain relevant despite local sanitization and data minimization. These controls reduce disclosure risk but do not constitute a formal privacy guarantee; external-LLM outputs should therefore remain subject to operator review rather than being used as automated confirmation of an attack.

E. LIMITATIONS

First, the internal evaluation still relies on a refined normal baseline and LLM-generated synthetic attacks. Although Org Y provides independent operational validation with dataset-provider-defined labels, it represents only one additional organization and service environment. The present results therefore do not establish broad generalization to diverse

production services, attack populations, evasive behaviors, or multi-stage attacks. The structural relationship between the internal normal-data refinement rules and the security-rule-derived feature group may also increase the apparent contribution of those features on the internal benchmark.

Second, the model-development and feature analyses remain limited in scope. The hyperparameter study used a restricted search space rather than exhaustive optimization, and the final CNN model-state selection retained the existing prefix–next-URL sample-level cross-validation procedure. The seven-configuration ablation was designed to diagnose the contribution of predefined feature groups rather than to perform feature selection, and the incremental value of URL-sequence rarity was not consistent across the internal and external datasets. These findings should therefore not be interpreted as establishing a universally optimal feature subset or sequence model.

Third, session construction remains an approximation because true actor or server-side session identifiers were unavailable. The timeout and client-key sensitivity analyses quantify the effects of alternative grouping rules but do not constitute controlled simulations of NAT collisions or adversarial User-Agent spoofing. In addition, sequence rarity is based on service-specific URL representations and a fixed maximum sequence length, which may reduce transferability when legitimate paths, request structures, or navigation patterns differ across services.

Finally, the engineering and interpretation evaluations remain bounded by the present experimental setting. The end-to-end benchmark characterizes one CPU-only implementation and does not include sustained production workloads, resource contention, active-session finalization delay, storage or dashboard overhead, or long-term model maintenance. The LLM evaluation is based primarily on Alerts derived from the controlled internal benchmark, and the human faithfulness study involved only two raters; analyst workflow benefits were not measured with security practitioners. Local sanitization and data minimization reduce external disclosure risk, but sensitive-field redaction recall, residual information leakage, and the privacy–utility trade-off were not quantitatively evaluated. The privacy-filtering stage should therefore be regarded as a risk-reduction mechanism rather than as a formal privacy guarantee.

F. FUTURE WORK

Future work should extend validation across additional services, organizations, software stacks, attack populations, and time periods, with chronological and client-group-aware evaluation where possible. Repeated calibration splits should also be used to assess the stability of P99 and auxiliary operating points under temporal and service-level distribution shift.

Model-development studies should examine broader hyperparameter search, session-grouped cross-validation for CNN prefix–next-URL learning, repeated feature ablation across independent services, and alternative sequence representations such as normalized endpoints, query abstraction, hierarchical URI tokenization, and different maximum sequence lengths. Controlled session-identification experiments with ground-truth actor or session identity are also needed to evaluate NAT collisions, User-Agent spoofing or rotation, and alternative identifiers or temporal-continuity signals.

Operational studies should evaluate sustained production workloads, long-term stability, active-session finalization, model-update cost, and end-user analyst workflows. The LLM interpretation module should be assessed on real operational Alerts with security practitioners, including investigation time, decision quality, inter-analyst consistency, and quantitative privacy evaluation. External API-based and self-hosted/on-premises LLM deployments should also be compared in terms of explanation quality, latency, cost, hardware requirements, privacy, and data-governance risk.

VI. CONCLUSION

This study proposed TraceLog, a session-based anomaly detection framework for web server HTTP access logs that combines session-level statistical features with URL-sequence rarity and provides optional evidence-guided LLM interpretation for detected Alerts. Markov-, LogBERT-, and CNN-based rarity representations were evaluated within a common IF-based pipeline under a calibration-and-evaluation protocol that separates model development, threshold determination, and held-out testing.

On the internal held-out benchmark, the CNN-based pipeline achieved the highest Recall and F_2 among the three pipelines at the primary normal-only P99 operating point. Feature ablation showed that the contribution of URL-sequence rarity was modest and dataset-dependent, while external validation on Org Y demonstrated substantial cross-service generalization limitations despite service-specific rebuilding. The sessionization and computational analyses further showed that deployment settings affect false-positive behavior and that the local detection pipeline is substantially faster than the optional external-LLM interpretation path.

For detected attack Alerts, the LLM interpretation module achieved a conditional attack-type classification accuracy of 91.10% on 382 true-positive Alerts, while the human evaluation showed rater-dependent explanation faithfulness. Overall, TraceLog should be viewed as an auxiliary session-prioritization and interpretation framework rather than as an autonomous replacement for existing security controls or analyst judgment. Broader validation across additional services, attack populations, time periods, and operational environments remains necessary before stronger generalization or deployment claims can be made.

REFERENCES

[1] OWASP Foundation. (2025). *OWASP Top 10:2025*. Accessed: Jul. 15, 2026. [Online]. Available: https://owasp.org/Top10/2025/

[2] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proc. 10th ACM Conf. Comput. Commun. Secur.*, Oct. 2003, pp. 251–261.

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.
[4] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.
[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.
[6] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 4765–4774.
[7] K. Banstola, F. A. Faisal, and X. Ou, "Experiences of using agentic AI to fill tooling gaps in a security operations center," in *Proc. Workshop Secur. Operation Center Oper. Construct. (WOSOC)*, Feb. 2026, pp. 1–7, doi: 10.14722/wosoc.2026.23016.
[8] Z. Ji, D. Chen, E. Ishii, S. Cahyawijaya, Y. Bang, B. Wilie, and P. Fung, "LLM internal states reveal hallucination risk faced with a query," 2024, *arXiv:2407.03282*.
[9] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen, "Analysis of HTTP requests for anomaly detection of web attacks," in *Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput.*, Aug. 2014, pp. 406–411.
[10] Q. Cao, Y. Qiao, and Z. Lyu, "Machine learning to detect anomalies in web log analysis," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 519–523.
[11] S. Cho and S. Cha, "SAD: Web session anomaly detection based on parameter estimation," *Comput. Secur.*, vol. 23, no. 4, pp. 312–319, Jun. 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404804000264
[12] H. Lam, D. Russell, D. Tang, and T. Munzner, "Session viewer: Visual exploratory analysis of web session logs," in *Proc. IEEE Symp. Vis. Anal. Sci. Technol.*, Oct. 2007, pp. 147–154.
[13] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.
[14] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervasive Intell. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr.(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 151–158.
[15] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," *Appl. Soft Comput.*, vol. 146, Oct. 2023, Art. no. 110689. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156849462300707X
[16] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang, K. Sui, and D. Pei, "An empirical investigation of practical log anomaly detection for online service systems," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2021, pp. 1404–1415.
[17] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems," *IEEE Access*, vol. 8, pp. 73127–73141, 2020.
[18] W. Guan, J. Cao, S. Qian, J. Gao, and C. Ouyang, "LogLLM: Log-based anomaly detection using large language models," 2024, *arXiv:2411.08561*.
[19] A. Peng, A. K. Chathoth, and S. Lee, "Log anomaly detection with large language models via knowledge-enriched fusion," 2025, *arXiv:2512.11997*.
[20] E. Dabbas. (2019). *Web Server Access Logs*. [Dataset]. Kaggle. Based F. Zaker. Online Shopping Store–Web Server Logs, Harvard Dataverse. Accessed: Jul. 17, 2026. [Online]. Available: https://www.kaggle.com/datasets/eliasdabbas/web-server-access-logs
[21] *Web Server Log Dataset*, Zenodo, [Dataset]. Version v1, MIT License, Anonim, Istanbul, Turkey, May 2026, doi: 10.5281/zenodo.20001206. Accessed: Sep. 15, 2026.
[22] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
[23] Scikit-Learn Developers. *Sklearn.Metrics.Average_Precision_Score*. Accessed: Jul. 16, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.averageprecisionscore.html
[24] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLoS ONE*, vol. 10, no. 3, Mar. 2015, Art. no. e0118432.
[25] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, Jun. 1947.
[26] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandin. J. Statist.*, vol. 6, no. 2, pp. 65–70, 1979.
[27] E. R. DeLong, D. M. DeLong, and D. L. Clarke-Pearson, "Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach," *Biometrics*, vol. 44, no. 3, pp. 837–845, Sep. 1988.



SEYEON JANG is currently pursuing the B.S. degree with the Department of Computer Engineering, Kumoh National Institute of Technology (KIT), Gumi, South Korea. Her current research focuses on machine learning-based anomaly detection in web logs and the design of data-processing pipelines. Her research interests include software engineering, systems architecture, data engineering, and applied artificial intelligence.



JINHO PARK is currently pursuing the B.S. degree with the Department of Computer Engineering, Kumoh National Institute of Technology (KIT), Gumi, South Korea. His research interests include scalable backend systems, cloud-native computing, data-driven web services, and AI service integration.



NAYOUNG KIM is currently pursuing the B.S. degree with the Department of Computer Engineering, Kumoh National Institute of Technology (KIT), Gumi, South Korea. Her research interests include artificial intelligence, backend development, and systems and networking.



SOOHYUN PARK received the B.S. degree from the Department of Computer Engineering, Kumoh National Institute of Technology (KIT), Gumi, South Korea, in 2025. He is currently pursuing the integrated M.S. and Ph.D. degree with the Department of Software Engineering, KIT. His research interests include computer vision, medical image analysis, deep learning, and artificial intelligence.



YOUNGWOO KIM received the M.S. degree in information and communications engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2009, and the Ph.D. degree in biomedical engineering from Seoul National University, Seoul, South Korea, in 2014. He was a Research Instructor with the Department of Radiology, University of Pittsburgh Medical Center (UPMC), Pittsburgh, PA, USA, until 2018, and a Principal Researcher with the Smart Wellness Group, Korea Institute of Industrial Technology (KITECH), until 2023. Since 2024, he has been a Professor with the Department of Computer Engineering, Kumoh National Institute of Technology (KIT), Gumi, South Korea. His research interests include anomaly detection and AI-based medical image analysis, including noise reduction, precision medicine, and radiogenomics.