

A Neural Screener–Reasoner Framework for Evidence-Grounded Log-Based Anomaly Detection and Explanation

Chi-Ming Chou

s1136107@mail.yzu.edu.tw

Yuan Ze University <https://orcid.org/0009-0003-7604-3840>

Shang-Kuan Chen

Yuan Ze University

Research Article

Keywords: Log anomaly detection, Evidence-grounded explanation, Retrieval-augmented generation, Automated evidence verification

Posted Date: September 2nd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10885441/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.
[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Abstract

Log-based anomaly detection is essential for maintaining the reliability of large-scale computing systems. While recent deep learning models have achieved strong detection performance, their limited interpretability restricts operators from understanding why a session is anomalous and which log evidence supports the decision. This work proposes a Screener–Reasoner framework for evidence-grounded log anomaly detection and explanation. The Screener is a parser-free anomaly detector based on linear self-attention. It directly processes raw log sessions without input truncation and efficiently handles long log sequences. For sessions predicted as anomalous, the Reasoner uses retrieval-augmented large language model (LLM) reasoning to generate structured trace explanations, in which each claim must cite explicit evidence identifiers and line-level spans. An automated verifier then checks schema validity, citation integrity, span consistency, and grounding constraints. Confidence-based gating is further introduced to control LLM inference cost under limited budgets. Experiments on the Blue Gene/L (BGL) and Hadoop Distributed File System (HDFS) datasets show that the Screener achieves F1-scores of 0.999 and 0.996, respectively, while the Reasoner-generated explanations pass all rule-based verification checks. Human evaluation further reports mean scores above 4.6 out of 5 in correctness, completeness, and evidence grounding. These results demonstrate a practical path toward auditable, cost-aware, and deployable log anomaly analysis.

**A Neural Screener–Reasoner Framework for
Evidence-Grounded Log-Based Anomaly
Detection and Explanation**

1. Introduction

Interconnected large-scale systems are essential to modern infrastructure, but their growing complexity also increases the difficulty of detecting abnormal behavior. Failures, attacks, or misconfigurations in such systems can lead to service disruption, data loss, and security risks. System logs provide semi-structured records of runtime behavior, including timestamps, severity levels, components, and free-form messages, making them a key source for anomaly detection. However, manually inspecting large-scale, high-velocity, and heterogeneous log streams is impractical. Automated log analytics are therefore critical for reliable and scalable system operation.

Within automated log analytics, log-based anomaly detection has progressed from machine-learning methods [1–3] to deep-learning methods [4–12]. Recent methods have pushed benchmark detection performance close to saturation on widely used datasets. However, high classification accuracy alone does not meet the operational need for diagnosis and review. A binary anomaly label does not explain why a session was flagged, which log lines support the decision, or how the observed behavior relates to known failure patterns. Operators therefore need interpretable, auditable, and evidence-grounded explanations. This requirement highlights a gap in current research on log anomaly detection. The task is no longer only to accurately detect anomalous sessions but also to support evidence-grounded anomaly analysis.

A large portion of the literature still relies on log parsing [3,13–20] to convert raw messages into structured templates or event sequences before modeling. Although parsing simplifies downstream learning, it may remove dynamic runtime details, introduce semantic errors, and require ongoing maintenance as logging systems evolve. Prior studies [21–23] have shown that these parser-induced issues can negatively affect log anomaly detection performance. For robust anomaly analysis, preserving raw-log context is therefore not a convenience but a modeling requirement. Direct raw-log modeling also increases sequence length, making computational efficiency essential. To address this need, this work uses AllLinLog, a parser-free anomaly screener based on linear self-attention, as the screening foundation of the proposed evidence-grounded anomaly analysis framework.

Large language models (LLMs) can generate readable explanations for anomalous log sessions. Recent studies have begun using them for interpretable log analysis or post-detection anomaly analysis, such as LogPrompt [24] and LogLAA [12]. Readability does not guarantee reliability. Without explicit evidence constraints, generated explanations may contain unsupported claims, vague reasoning, or fabricated evidence references. This concern is consistent with recent research on LLM hallucination and retrieval-augmented generation evaluation, which emphasizes factuality, faithfulness, and relevant external evidence [25,26]. Therefore, explanation generation should be grounded in traceable evidence and followed by automatic verification. In the proposed framework, the Reasoner generates structured trace JSON in which every claim cites evidence identifiers and line-level spans. The generated output is then checked by an automated verifier for schema validity, citation integrity, span consistency, and grounding constraints.

Another practical challenge is the cost of LLM reasoning. In real-world log streams, invoking the Reasoner for every detected anomaly can be computationally expensive, especially when many

anomalous sessions exhibit repeated or similar failure patterns. A deployable framework therefore needs a mechanism to decide which anomalous sessions should receive full reasoning under a limited inference budget. To address this issue, this work introduces confidence-based gating, which prioritizes sessions with lower Screener confidence for explanation. This design enables the framework to balance explanation coverage against LLM inference cost.

To address this gap, this article proposes a unified Screener–Reasoner framework for evidence-grounded log anomaly analysis, as shown in **Fig. 1**. The framework combines a linear self-attention-based Screener with a retrieval-augmented LLM Reasoner that produces structured explanations for anomalous sessions. The Screener first performs efficient parser-free anomaly detection on raw log sessions. For sessions routed to the Reasoner, the Reasoner retrieves supporting evidence and generates structured explanations whose claims are linked to explicit evidence identifiers and line-level spans. An automated evidence verifier then checks the required schema, citation, span, and grounding constraints. Together with confidence-based gating, the framework supports accurate detection, evidence-grounded explanations, and budget-aware deployment.

This article shifts the focus from incremental gains in anomaly classification accuracy to the broader problem of explainable, auditable, and practically deployable log anomaly analysis. The main contributions of this article are fourfold. First, it formulates log anomaly analysis as a unified Screener–Reasoner problem that decouples efficient anomaly screening from evidence-grounded explanations. Second, it employs a parser-free Screener based on linear self-attention to process long raw-log sessions without log parsing or input truncation. Third, it introduces a structured trace schema and an automated verification pipeline for claim-level traceability and evidence grounding. Fourth, it evaluates confidence-based gating as a practical mechanism for controlling LLM inference cost while retaining meaningful anomaly coverage. These contributions are validated on the Blue Gene/L (BGL) and Hadoop Distributed File System (HDFS) datasets through detection performance, automated verification, retrieval-aware faithfulness metrics, gating analysis, and human evaluation.

The remainder of this article is organized as follows: Section 2 discusses the related work. Section 3 presents the proposed Screener–Reasoner methodology. Section 4 reports the experimental results on BGL and HDFS. Section 5 concludes this article and outlines directions for future research.

2. Related Work

2.1 Parser-Free Screening

Parser-free log anomaly detection aims to model log messages without first converting them into event templates. This direction is relevant to the proposed framework because preserving raw log context is necessary not only for anomaly classification but also for later evidence-grounded explanations. NeuralLog [21] directly processes raw log messages using pretrained language representations. LogLLM [27] applies regular-expression-based preprocessing and combines BERT with Llama for log sequence classification. ContextLog [28] further preserves both alphabetic and non-alphabetic units from raw logs using Byte Pair Encoding (BPE) [29] tokenization. These studies show that reducing parser dependence can preserve richer contextual information and improve robustness against unstable log patterns.

A practical challenge of parser-free screening is that raw log sessions can become much longer than template sequences. When alphabetic tokens, non-alphabetic symbols, identifiers, and dynamic values are preserved, the sequence length may grow to thousands of tokens. Standard self-attention [30] becomes computationally expensive for raw-log modeling. Efficient attention mechanisms, such as Sparse Transformer [31], Reformer [32], and Linformer [33], reduce the cost of long-sequence modeling. Following this direction, the Screener adopts linear self-attention to process long raw-log sessions efficiently without log parsing or input truncation. However, existing parser-free methods mainly treat anomaly detection as the final task. They focus on whether a log session is normal or anomalous, but provide limited support for explaining which concrete log evidence supports the decision. In this work, parser-free screening is used as the first stage of a broader evidence-grounded anomaly analysis framework.

2.2 LLM-assisted Log Analysis

Large language models have recently been used in log anomaly detection and log analysis because they can process natural-language-like log messages and generate human-readable outputs. Existing studies use LLMs in different ways: prompt-based methods use general LLMs for zero-shot, few-shot, or interpretable log analysis [7,24], while LogLLM [27] integrates pretrained language models into an anomaly detection pipeline to classify log sequences. LogLAA [12] is more closely related to this work because it also applies an LLM after anomaly detection to support anomaly analysis. Its analysis quality is evaluated by comparing the generated analysis with human solutions using a weighted matching score, whereas this work focuses on claim-level evidence traceability and machine-verifiable grounding. These studies show that LLMs can improve semantic modeling and generate readable analyses, but their focus remains on detection, interpretation, or solution-level similarity analysis, rather than claim-level evidence traceability.

The proposed Reasoner is related to these LLM-assisted approaches but uses the LLM in a different role. Instead of relying on the query session or the model’s internal knowledge alone, it retrieves

historical anomaly examples and normal contrast sessions from an evidence store constructed only from the training set. BM25 [34] is used as a lightweight retrieval method because related log sessions often share recurring lexical cues. The retrieved evidence supports pattern-matching and contrast claims, enabling the generated explanation to relate the current anomalous session to historical log behavior. Therefore, the Reasoner is positioned not as a standalone LLM classifier but as a retrieval-augmented explanation module for evidence-grounded log anomaly analysis.

2.3 Evidence-Grounded Explanation

LLM-generated explanations can be fluent and readable, but readability alone does not guarantee reliability. A recent survey on LLM hallucination reports that generated text may contain unsupported or fabricated information, even when it appears coherent [25]. This issue is particularly important in log anomaly analysis, where operators need to verify which log lines support an explanation before using it for diagnosis or escalation. Therefore, explanation quality should be evaluated not only by natural-language readability but also by the extent to which it is grounded in concrete evidence.

Retrieval-augmented generation (RAG) addresses part of this problem by conditioning generation on external evidence rather than relying only on the model’s internal knowledge [35]. RAGAs [26] further formalizes the evaluation of retrieval-augmented generation through dimensions such as faithfulness, context relevance, and answer relevance. These ideas are relevant to this work because the proposed Reasoner also uses retrieved evidence to support the generation of explanations. However, general RAG evaluation is typically designed for open-domain question answering, where the output is often assessed against the retrieved passages as a whole. Unlike typical RAG evaluations, log anomaly explanation requires a finer level of traceability: each explanatory claim must be linked to specific evidence identifiers and line-level spans.

The proposed framework extends this evidence-grounding perspective to log anomaly explanation. In this setting, retrieved evidence should not merely provide background context for generation; it should define the boundary within which each explanatory claim can be justified. This reframes explanation quality from a matter of textual readability to a matter of traceability, specifically whether the generated analysis can be linked back to concrete log evidence in a verifiable form.

3. Methodology

The proposed framework, shown in **Fig. 1**, includes three closely connected components: a parser-free Screener for anomaly detection, a Reasoner for retrieval-based explanation generation, and a Gating layer for budget-conscious deployment. The Screener first classifies each raw log session as normal or anomalous. However, a binary label alone does not indicate which log lines triggered the alarm, why the observed behavior is anomalous, or how the incident relates to previously observed failure modes. To address this limitation, the framework routes selected anomalous sessions to the Reasoner, which generates structured, evidence-grounded explanations. This design preserves the efficiency of the lightweight Screener while applying the more expensive reasoning process only to sessions that require further inspection.

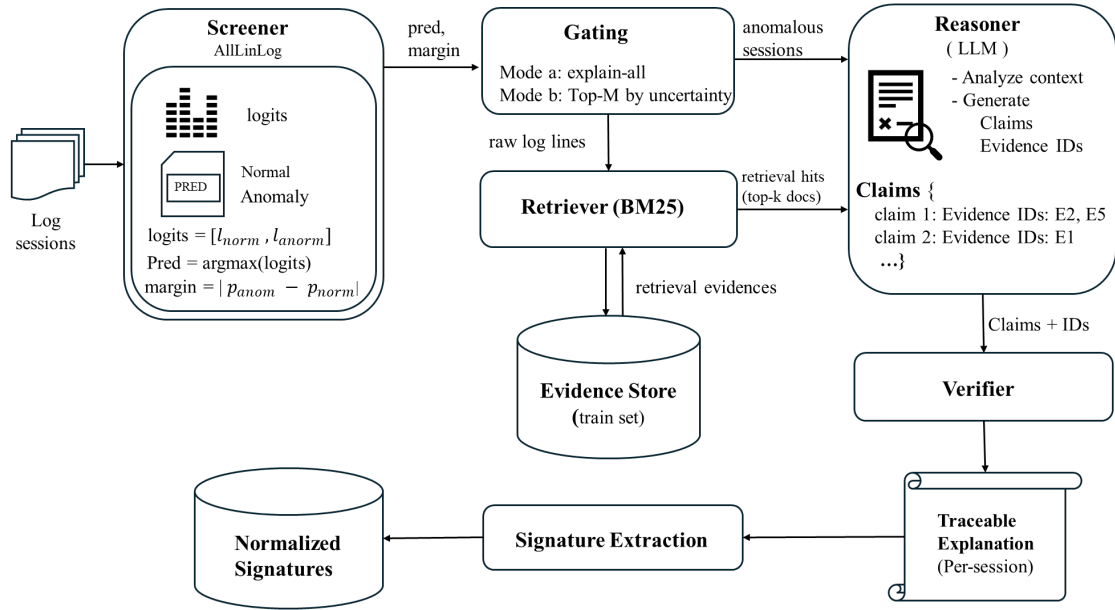


Fig. 1 Overview of the Screener-Reasoner Framework

3.1 Screener

The proposed Screener is designed as a lightweight parser-free detector. It consists of three major components. First, the data processing layer constructs log sessions and directly tokenizes raw log messages using the GPT-4 BPE tokenizer, preserving raw lexical and contextual information. Second, the linear self-attention (LSA) encoder reduces the computational complexity of self-attention and enables efficient modeling of long log sequences. Third, the classifier aggregates the encoder outputs and predicts whether each session is normal or anomalous. **Fig. 2** illustrates the Screener workflow, including data processing, the encoder, and classification stages. The following subsections describe each component in detail.

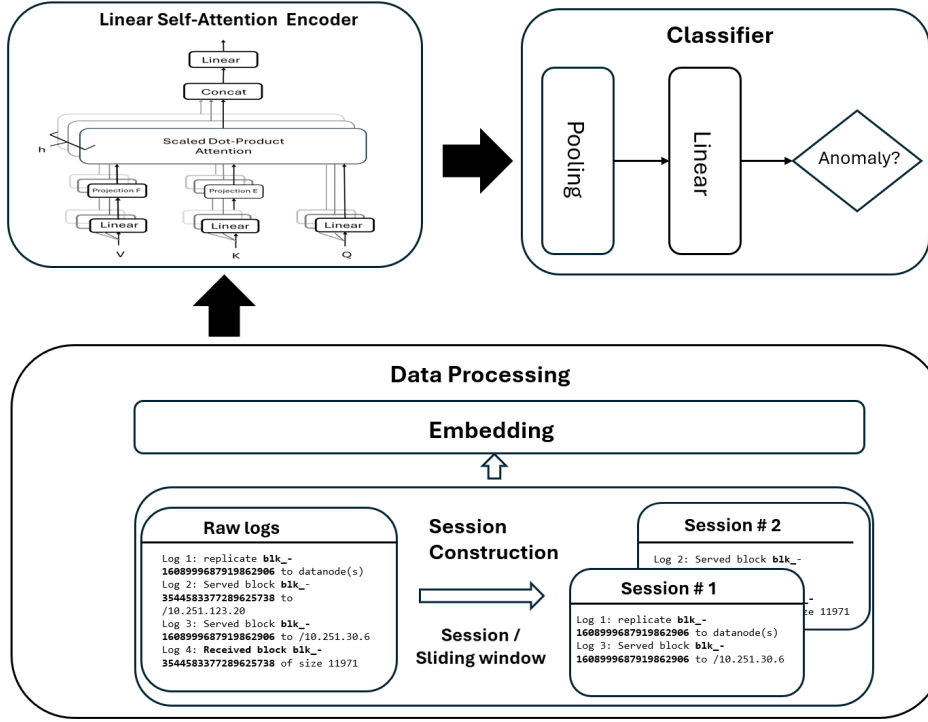


Fig. 2 Workflow of the proposed Screener

3.1.1 Data Processing

Session Construction

Research typically uses open-source system log datasets, such as BGL and HDFS, to evaluate the performance of log-based anomaly detection models [36]. The Screener first converts continuous raw log streams into session-level inputs. A session is constructed either by grouping log messages with the same session identifier or by applying a fixed-size sliding window, depending on the dataset characteristics.

For the HDFS dataset, log messages can be associated with block identifiers. Therefore, messages referring to the same block ID are grouped into the same session, as illustrated in **Fig. 3**. For the BGL dataset, which lacks explicit session identifiers, sessions are generated using a fixed-size sliding window. As shown in **Fig. 4**, each session contains a fixed number of consecutive log messages. For example, with a window size of three and a step size of three, non-overlapping sessions are created from the raw log stream.

Let $L = \{l_1, l_2, \dots, l_N\}$ be the log messages, and $S = \{s_1, s_2, \dots, s_M\}$ be a collection of sessions created from set L . A session is segmented as:

$$S_m = \{l_{m1}, l_{m2}, \dots, l_{mw}\} \text{ where } w = \text{window size} \quad (1)$$

The proposed Screener is trained as a supervised classifier, and each session is assigned a binary label. For BGL, a session is labeled anomalous if any log message in the session contains an anomaly marker. Otherwise, it is labeled normal. For HDFS, the session label follows the ground-truth label associated with the corresponding block ID. This session-level input construction enables the Screener to model semantic and temporal relationships across log messages within each session.

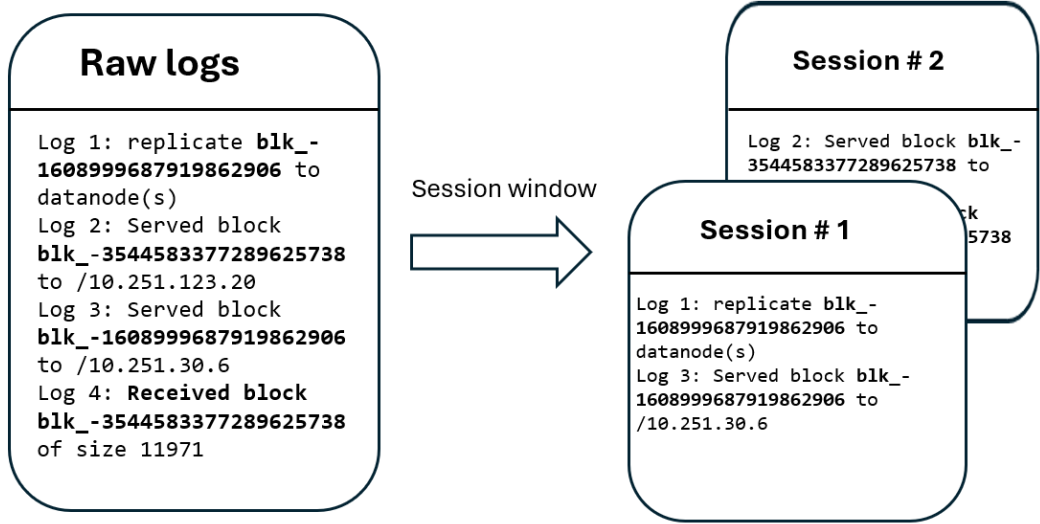


Fig. 3 Grouping sessions by session identifiers, such as block ID

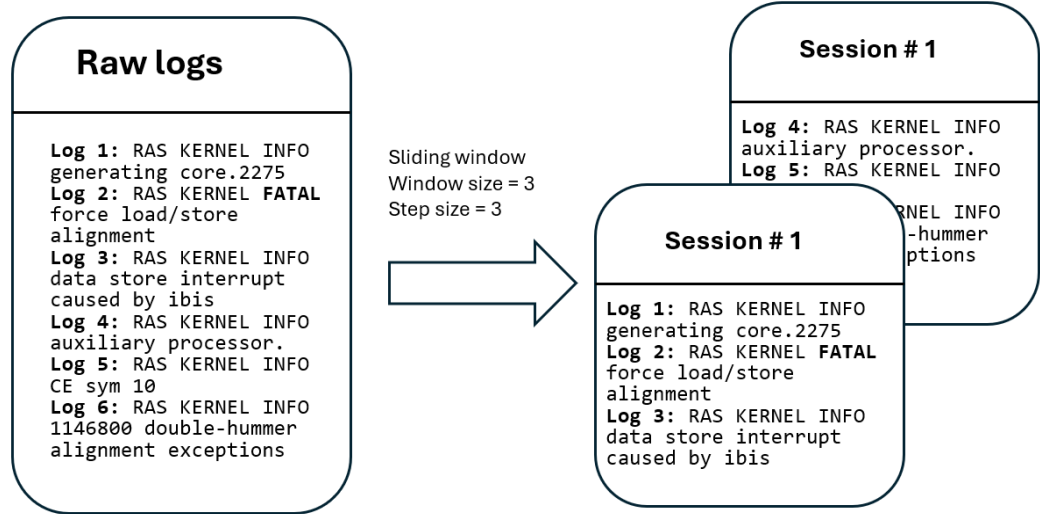


Fig. 4 Constructing sessions uses a fixed-size sliding window over the raw system log

Log Tokenization and Vectorization

After session construction, each raw log session must be converted into a numerical representation before being processed by the encoder. The proposed Screener uses GPT-4 BPE to tokenize raw log messages directly, without first converting them into log templates. This parser-free tokenization preserves lexical units, symbols, identifiers, and dynamic values that may be removed or distorted by log parsing.

BPE [29] tokenization is suitable for raw log modeling because it decomposes unseen words into subword units, reducing the out-of-vocabulary problem. For example, as shown in **Fig. 5**, the unseen word “crond” is split into “cr” and “ond”. Let T be the pretrained GPT-4 tokenizer. The session s_m is tokenized by T to yield a sequence of token IDs.

$$x_m = T(s_m) = [t_1^{(m)}, t_2^{(m)}, \dots, t_{n_m}^{(m)}], t_i^{(m)} \in \mathbb{Z}_+ \quad (2)$$

```

Raw Log      : crond pam_unix 16448 : session closed for user root
Split to Tokens : ['cr', 'ond', ' pam', ' _unix', ' ', '164', '48', ' ', ':', ' session', ' closed', ' for', ' user', ' root']
Encode to IDs  : [5192, 2159, 42290, 81672, 220, 10513, 2166, 551, 3882, 8036, 369, 1217, 3789]
Decode to Log  : crond pam_unix 16448 : session closed for user root

Raw Log      : ERROR: Disk failure at sector 4096.
Split to Tokens : ['ERROR', ':', ' ', ' Disk', ' failure', ' at', ' sector', ' ', '409', '6', ' ']
Encode to IDs  : [3767, 25, 39968, 8060, 520, 10706, 220, 12378, 21, 13]
Decode to Log  : ERROR: Disk failure at sector 4096.

```

Fig. 5 System log tokenized by GPT-4 BPE

Table 1: Average and maximum token counts in dataset sessions

Dataset	# of log messages	Grouping method	# Avg of tokens of a session	# Max of tokens of a session
BGL	4,747,963	Sliding	756	2756
		window=10		
		Sliding	1453	4841
		window=20		
HDFS	11,175,629	Session window	1006	15983
		(same blk-id)		

The proposed Screener segments tokenized log messages into log sessions based on dataset characteristics. Table 1 reports the session lengths of BGL and HDFS. In the BGL dataset, there are 4,747,963 log messages. When a window size of 20 is used to form a session, the average token length of a session is 1,453, and the maximum token length is 4,841. For HDFS, the average session token length is 1,006, and the maximum length is 15,983. This long input sequence length will be a challenge for the traditional Transformer encoder. The proposed Screener addresses this issue with an efficient encoder, as stated in Section 3.1.2. These tokens are then mapped to the encoder embedding dimension. The final input matrix $E_m \in \mathbb{R}^{n_m \times d}$ for each session m is computed as follows:

$$E_m = Embed_{token}(x_m) \quad (3)$$

where $Embed_{token} : \mathbb{Z}_+^{n_m} \rightarrow \mathbb{R}^{n_m \times d}$

n_m : the sequence length of session m

d : the embedding dimension, we set $d = 128$ in the proposed model

Handling Data Imbalance with Oversampling

A significant class imbalance is present in the datasets, as shown in Table 2. With a window size of 20, the anomalous-session ratio is 9% for BGL. For HDFS, where sessions are grouped by block ID, the anomalous-session ratio is 3%. Because such an imbalance may bias a supervised detector toward the normal class, class resampling is required during training.

Table 2: The ratio of anomalous sessions of BGL and HDFS datasets

Dataset	# of Log messages	# of Log sessions	# Anomalous sessions	Anomaly ratio
BGL	4,747,963	237,389	20,711	0.09
HDFS	11,175,629	575,061	16,838	0.03

The proposed Screener applies Random Oversampling (ROS) to the training data before tokenization and feature embedding. Common resampling strategies include oversampling, undersampling, and hybrid sampling [37]. ROS is adopted because it directly increases the representation of anomalous sessions without removing normal sessions from the training set.

Let β be the target proportion of the anomalous class for oversampling in a dataset, α be the current proportion of the anomalous class with $\alpha < \beta$, and the total number of log sessions to classify is N . To reach the proportion of β , the training data will need to be randomly sampled from the training set as an additional anomalous session quantity X , which satisfies $\alpha \times N + X = \beta(N + X)$. The following formula can be used to calculate X .

$$X = \frac{N(\beta - \alpha)}{1 - \beta} \quad (4)$$

For BGL, the original anomalous-session ratio is 0.09, and the target percentage for the anomalous class β is set to 0.4 in the proposed screener implementation.

The random oversampling function O applies to session set S to rebalance the minority class S' . Let α be the original minority class ratio, β the expected minority ratio, and S_1 be the set of anomalous sessions.

$$S' = O(S; \alpha, \beta), \quad \text{where } \alpha = \frac{|S_1|}{|S|} \quad (5)$$

After incorporating the oversampled anomalous samples, the number of anomalous sessions can be calculated as follows:

$$|S_1^{new}| = \left\lceil \frac{\beta(1 - \alpha)}{1 - \beta} \bullet |S| \right\rceil \quad (6)$$

3.1.2 Linear Self-Attention Encoder

Reducing Self-Attention Complexity

The self-attention mechanism computes contextual representations by comparing each query token with all key tokens and using the resulting attention weights to aggregate value vectors. Given query, key, and value matrices Q , K , and V , scaled dot-product attention is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (7)$$

where d is the dimensionality of the key vectors.

In multi-head attention, each input vector of dimension d_{model} is projected by learned parameter matrices into query, key, and value vectors with dimensions d_k , d_k , and d_v , respectively. The attention operation is applied in parallel across multiple heads, and the resulting outputs are concatenated:

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (8)$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$

Although multi-head self-attention is effective for capturing contextual dependencies, it requires each token to attend to every other token in the sequence. For a session with n tokens, this results in $O(n^2)$ time and memory complexity. This becomes a bottleneck for raw-log modeling because parser-free tokenization preserves lexical units, symbols, identifiers, and dynamic values, resulting in log sessions with thousands of tokens. To reduce this cost, the Screener adopts linear self-attention [33], which approximates full self-attention through low-rank projection. As shown in **Fig. 6**, instead of computing attention over the full $n \times n$ token space, two learnable projection matrices E and F compress the key and value sequences from length n to a smaller projected length k where $k \ll n$. The resulting attention is computed as:

$$\text{Linear - Attn}(Q, K, V) = \text{softmax}\left(\frac{Q(EK)^T}{\sqrt{d}}\right)(FV) \quad (9)$$

where $E \in \mathbb{R}^{k \times n}$ is used to compress the key sequence, and $F \in \mathbb{R}^{k \times n}$ is used to compress the value sequence. This reduces the complexity from $O(n^2)$ to $O(nk)$, which is linear in n when k is fixed. Table 3 summarizes the computational complexity of the standard Transformer and the proposed linear self-attention encoder.

Table 3: The computational complexity of the Transformer and linear self-attention

operation	Transformer complexity	Linear self-attention complexity
Computing QK^T	$O(n^2d)$	$O(nkd)$
Softmax	$O(n^2)$	$O(nk)$
Multiplying by V	$O(n^2d)$	$O(nkd)$
Total Complexity	$O(n^2)$	$O(n)$

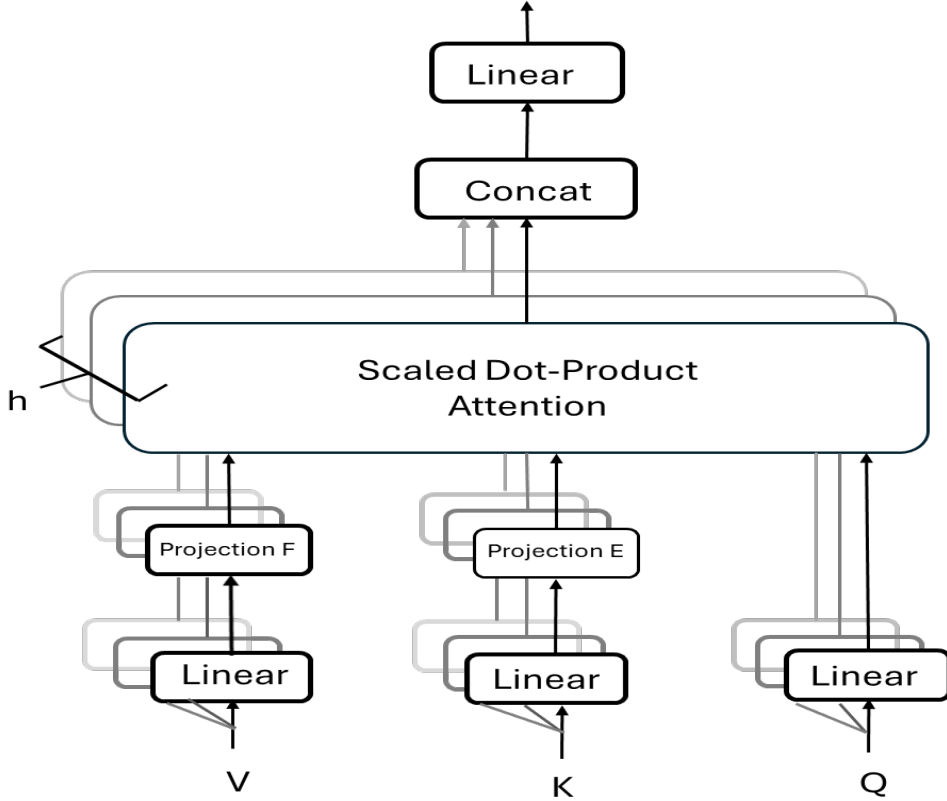


Fig. 6 Multi-head self-attention with projection E and F

Efficient Encoder

The proposed Screener reduces the quadratic computational complexity of standard self-attention by using a linear self-attention encoder. As illustrated in Fig. 6, the encoder compresses the key and value sequences before computing attention, enabling linear time and memory complexity in the sequence length. The encoder projects the original key (KW_i^K) and value (VW_i^V) layers, which are $n \times d$ in dimension, into $k \times d$ dimensional projected key and value layers using learnable matrices E and F . Then, the encoder applies the scaled dot-product and the softmax function to Q ($n \times d$) and K ($k \times d$) to compute the attention score. With this attention score and V ($k \times d$), it obtains the output of each head as follows:

$$\begin{aligned}
 head_i &= \text{Attention}(QW_i^Q, E_iKW_i^K, F_iVW_i^V) \\
 &= \text{softmax} \left(\frac{QW_i^Q (E_iKW_i^K)^T}{\sqrt{d_k}} \right) \cdot F_iVW_i^V
 \end{aligned} \tag{10}$$

To make the encoder lighter, the proposed Screener adopts a layer-wise parameter-sharing technique utilizing a single projection matrix E , with $E_i = F_i = E$ across all layers and heads. This reduces the number of additional parameters in the proposed model, thereby decreasing its memory footprint.

3.1.3 Classification Layer

The linear self-attention encoder outputs a contextual representation $H \in \mathbb{R}^{n \times d}$ for each input session, where n is the sequence length and d is the embedding dimension. To obtain a session-level representation, the classifier applies average pooling over the sequence dimension:

$$h_{avg} = \frac{1}{n} \sum_{i=1}^n H_i \quad (11)$$

where $H_i \in \mathbb{R}^d$ is the encoded representation of the i -th token. The pooled representation $h_{avg} \in \mathbb{R}^d$ is then passed through a linear layer to produce the logits for binary classification:

$$z = W \cdot h_{avg} + b \quad (12)$$

where $W \in \mathbb{R}^{2 \times d}$ and $b \in \mathbb{R}^2$ are all learnable parameters, and $z \in \mathbb{R}^2$ represents the unnormalized logits for the normal and anomalous classes.

3.2 Reasoner

The Reasoner is built on a retrieval-augmented generation (RAG) architecture in which a large language model (LLM) is guided by retrieved evidence and a machine-verifiable output schema. The Reasoner consists of six sub-components, including the evidence store, retriever, prompt builder, LLM inference, verifier, and signature generator. The evidence store provides a searchable corpus of normalized historical sessions. The prompt builder then assembles the target session and the retrieved evidence into a fixed prompt format that guides the LLM to generate a structured explanation. After generation, the verifier checks whether the output satisfies the required schema and whether the cited evidence meets the grounding constraints. Finally, the generated anomaly signature is normalized to support aggregation and downstream analysis.

The Reasoner is designed for forensic explanations rather than root-cause diagnosis. Its goal is to summarize the abnormal features of a session, connect them to previously observed anomaly patterns, and, when appropriate, compare them with normal evidence. Every generated claim must be supported by explicit evidence identifiers and line-level evidence spans, enabling automated verification and human review to ensure traceability.

3.2.1 Evidence Store

The evidence store is a precomputed corpus of normalized log sessions constructed from the training split. It provides the grounding context for LLM-generated explanatory claims. The evidence store is constructed under two requirements. First, only training-split sessions are indexed to prevent data leakage from the test set. Second, both normal and anomalous sessions are included, enabling the LLM to

generate contrast claims that highlight behavioral differences between normal and anomalous log sessions.

Raw log messages contain dynamic parameters, including IP addresses, UUIDs, hexadecimal memory addresses, file system paths, and numerical values. These dynamic portions may vary across sessions without changing the underlying behavior pattern. If left unnormalized, lexical retrieval may match incidental parameter values rather than structural log patterns. Therefore, the proposed framework uses a dataset-specific log normalizer that replaces dynamic parameters with typed placeholders. Let ℓ denote a raw log line and $N_D(\cdot)$ denote the normalizer for dataset D . The normalized line is computed as:

$$\ell' = N_D(\ell) = \text{RegChain}(\ell, P_D) \quad (13)$$

where P_D is an ordered list of regular expression substitution rules specific to the dataset D . For BGL, the rule set includes node identifiers, hexadecimal addresses, file system paths, and numeric values. For HDFS, an analogous rule set normalizes block identifiers, IP addresses, and port numbers. All rules are applied in a fixed order to avoid interaction effects.

For datasets like HDFS, where anomalous and normal sessions mostly share the same vocabulary and differ mainly in event sequences, lexical normalization alone is not enough for effective retrieval. The proposed framework, therefore, augments each evidence document with a structural summary consisting of a set of operation-count tags and anomaly-indicative structural flags. For each HDFS session, the normalizer counts the occurrence of key operations and appends a structural anomaly tag (e.g., INCOMPLETE_PIPELINE, EXCESS_REPLICATION) when characteristic event patterns are detected. These injected tokens serve as discriminative features in BM25 [34], enabling the retriever to distinguish structurally anomalous sessions from normal ones that otherwise share the same vocabulary.

The evidence store produces one evidence document per session. Given a set of training sessions $S_{train} = \{s_1, s_2, \dots, s_m\}$. For each session s_i , the document text is computed by applying the dataset-specific normalizer $N_D(\cdot)$ to all lines in the session, and concatenating their structural summary when available:

$$d_i = N_D(s_i) \parallel \text{Struct}_D(s_i) \quad (14)$$

where $\parallel$ denotes text concatenation and $\text{Struct}_D(s_i)$ denotes the dataset-specific structural summary. If no structural summary is used for a dataset, $\text{Struct}_D(s_i)$ is set to empty string.

Each evidence document is stored as a tuple (e_i, d_i, m_i) , where e_i is a unique evidence identifier, d_i is the normalized text, and m_i is a metadata record containing the session label, dataset name, line count, and normalization statistics. The label in the metadata is used only for retrieval routing and prompt construction, allowing the Reasoner to distinguish anomalous evidence examples from normal contrast evidence. This does not constitute label leakage because the labels belong to reference evidence from the training split, not to the query session under analysis. The schema of an evidence document in the evidence store is shown in **Fig. 7**.

EvidenceDoc	
evidence_id	"E_BGL_03700010"
session_id	"BGL_03700010"
evidence_type	"session"
text (BM25 indexed)	10 normalized lines, joined by \n <i>Example:</i> <pre>"<IPV6> <NUM>.<NUM> <NODE> RAS KERNEL INFO 1 torus receiver x- input pipe error(s) (dcr <HEX>) detected\n<IPV6> <NUM>..."</pre> Replacements: IP address -> <IPV6> numbers -> <NUM> node id -> <NODE> hex value -> <HEX>
metadata	label : 0 (0=normal, 1=anomaly) dataset : "BGL" num_lines : 10 original_length : 1,757 (chars) normalized_length: 1,677 (chars) param_stats : { NODE: 20, IPV6: 10, HEX: 9, NUM: 103 }

Fig. 7 Schema of an evidence document in the evidence store

At inference time, the evidence store provides up to five pieces of evidence per query session. Following this framework's naming convention, the query session itself is designated as E_0 . The top-4 retrieval anomaly examples are labeled E_1 through E_4 , and the highest-scoring normal session is labeled E_5 . This fixed six-slot evidence layout enables the trace schema described in Section 3.2.3 to enforce that every claim references a specific evidence identifier and its corresponding line-level span. With this, the reasoner establishes a verifiable chain of evidence between the explanation and the raw log data.

3.2.2 Retriever

The retriever selects relevant documents from the evidence store to serve as grounding context for LLM-based explanation generation after the Screener flags a query session. The retrieval step is important because, without external evidence, the LLM can only refer to the query session itself, limiting its ability to generate evidence-supported pattern-match and contrast claims. Retrieval therefore provides the channel through which historical failure patterns and normal-behavior baselines are introduced into each explanation prompt.

The retriever uses BM25 [34] to rank evidence documents with respect to the normalized query session:

$$\text{BM25}(q, d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t, d)(k_1 + 1)}{f(t, d) + k_1(1 - b + b |d| / \text{avgdl})} \quad (15)$$

where q is the query session, $f(t, d)$ is the term frequency of token t in document d , $IDF(t)$ is the inverse document frequency, $|d|$ is the document length in tokens, and $avgdl$ is the average document length over the corpus. $k_1 \in \mathbb{R}^+$ controls term-frequency saturation, and $b \in [0,1]$ controls document-length normalization. In this work k_1 is set to 1.5 and b is set to 0.75. BM25 is suitable for log retrieval for two reasons. First, log messages contain repetitive structured vocabulary, including component names, error codes, and kernel identifiers, which creates strong term overlap between related failure events. Second, BM25 does not require supervised training beyond the normalization performed during evidence creation. At index construction time, each evidence document is lowercased and split on whitespace. The BM25 index is then constructed over the resulting corpus of document token lists.

The retriever constructs each query representation by joining the log lines of the query session with newline delimiters, appending the structural summary when available for HDFS, and applying the same dataset-specific normalizer used during evidence construction. The normalizer replaces instance-specific identifiers, such as node IDs, IP addresses, hexadecimal values, and integers, with typed placeholders. This normalization allows sessions that record similar failures on different nodes to produce comparable normalized query tokens.

The retriever uses a mixed-evidence strategy that returns two types of evidence for the prompt slots. The first type consists of anomaly examples E_1 through E_4 , which are the four highest-scoring documents among training sessions labeled as anomalous. These examples provide previously observed failure patterns similar to the current query and support pattern-match claims. The second type is the normal contrast evidence E_5 , which is the highest-scoring document among training sessions labeled as normal. This evidence provides a representative baseline of normal behavior for the same subsystem or workload type.

The query session is designated E_0 and is not retrieved from the evidence store. The identifiers E_0 through E_5 are propagated directly into the prompt, where each cited line is addressed by the fine-grained notation E_i-L_j representing evidence slot i , and line j . This allocation of four anomalous evidence slots and one normal contrast slot provides sufficient diversity for pattern-match grounding while reserving explicit evidence for contrast-based explanation.

3.2.3 Prompt Builder

The prompt builder assembles the query session and the retrieved evidence into a two-part prompt that the LLM uses to generate a structured, machine-verifiable explanation. Let E_0 denote a query session, $\{E_1, \dots, E_5\}$ the retrieved evidence, $\hat{p}$ the anomaly probability, and m be the confidence margin produced by the Screener. The prompt P is constructed as:

$$P = (P_{sys}, P_{user}) \quad (16)$$

$$P_{sys} = \text{Template}(D, S) \quad (17)$$

$$P_{user} = \text{Format}(E_0, \{E_1, \dots, E_5\}, \hat{p}, m) \quad (18)$$

where D is the dataset configuration, and S is the output specification defined below. The system prompt P_{sys} defines the analytic context: dataset identity, component vocabulary (e.g., KERNEL and APP for

BGL; DATANODE and NAMENODE for HDFS), the E_i-L_j span notation introduced in Section 3.2.2, the three required claim types (observation, pattern_match, and contrast), the signature naming convention, and an annotated output example that specifies expected fields and constraints. The user prompt P_{user} instantiates session-specific content: the session identifier, $\hat{p}$ and m , the line-numbered E_0 log block, and the E_1 to E_5 evidence blocks. The structural constraints embedded in P are designed to be directly enforceable by the verifier described in Section 3.2.4.

The E_0 log lines are passed through the same dataset normalizer used during evidence construction. The normalizer replaces instance-specific identifiers, such as block IDs, with typed placeholders, so that E_0 and E_1 through E_5 share a consistent lexical representation, allowing the LLM to focus on structural log patterns rather than incidental parameter differences. All log lines in E_0 and E_1 through E_5 are prefixed with their span identifiers, and a header declares the total line count and valid span range for each evidence slot, which bounds the line references the LLM may cite. For HDFS, the structural summary described in Section 3.2.1 is appended to E_0 so that the LLM observes the same operation-count tag present in the retrieved evidence. Each E_1 through E_5 block additionally carries a metadata header that includes the evidence label (anomaly or normal). Exposing the label to the LLM is intentional. It enables the model to distinguish anomalous examples from the normal contrast session, which is necessary to generate correctly typed claims. This does not constitute label leakage because the labels refer to reference evidence sourced from the training split, not to the query session E_0 under analysis.

The prompt specifies a structured output format, which is referred to as the trace schema. The LLM must produce a structured explanation T , defined as follows:

$$T = (\hat{y}, \text{summary}, \sigma, C) \quad (19)$$

where $\hat{y} \in \{\text{anomaly}, \text{normal}\}$ is the predicted label, *summary* is a concise forensic description with quantified observations. $\sigma = (\text{name}, \{E_i\}_{\text{matched}})$ is the anomaly signature with matched evidence identifiers. $C = \{c_1, c_2, c_3\}$ is a set of exactly three claims. Each claim is a tuple:

$$c_k = (\tau_k, \text{text}_k, \mathcal{E}_k, \mathcal{R}_k) \quad (20)$$

where $\tau_k \in \{\text{observation}, \text{pattern_match}, \text{contrast}\}$ is the claim type, text_k is the claim text, $\mathcal{E}_k \subseteq \{E_0, \dots, E_5\}$ is the set of referenced evidence identifiers, and $\mathcal{R}_k$ is a non-empty set of evidence spans in E_i-L_j format. The prompt enforces a claim-type completeness constraint requiring exactly one claim of each type:

$$\{\tau_1, \tau_2, \tau_3\} = \{\text{observation}, \text{pattern_match}, \text{contrast}\} \quad (21)$$

This constraint ensures that every explanation integrates direct observation from E_0 , pattern matches against anomaly examples E_1 through E_4 , and behavioral contrasts with the normal session E_5 .

The system prompt further specifies formatting rules that make it machine-verifiable. Each element in $\mathcal{R}_k$ must conform to either the single-line format E_i-L_j or the range format E_i-L_j to E_i-L_k . Bare evidence IDs, alternative separators, and placeholder line numbers are not allowed. Every claim must reference at least one span within the specified valid range, and the claim text must mention terms from the cited lines to ensure span-level verifiability. The assembled prompt pair (P_{sys}, P_{user}) is submitted

to the LLM with JSON-mode output enabled. The returned JSON response is parsed into the explanation T structure and then sent to the verifier for schema and evidence grounding validation.

3.2.4 Verifier

The structural constraints in the prompt builder guide the LLM toward well-formed output, but prompt-time instructions alone cannot guarantee structurally valid and well-grounded explanations. The LLM may fabricate evidence identifiers that do not exist in the provided context, cite line spans outside the valid range, or attach claims to irrelevant log lines. To address these failure modes, the verifier performs rule-based post-hoc validation of each generated explanation against its corresponding evidence without requiring an additional LLM call.

Let T denote a generated explanation. The verification function is defined as follows.

$$V(T, E_0, \{E_1, \dots, E_5\}, \mathcal{M}) \rightarrow (b, \mathcal{F}) \quad (22)$$

where $\mathcal{M}$ maps slot identifiers to evidence store identifiers, $b \in \{PASS, FAIL\}$ is the overall verdict, and $\mathcal{F} = \{f_1, \dots, f_n\}$ is a set of individual check outcomes, each carrying a status $s_i \in \{PASS, FAIL, WARNING\}$. The overall verdict follows strict failure semantics:

$$b = PASS \Leftrightarrow \forall f_i \in \mathcal{F}, s_i \neq FAIL \quad (23)$$

The WARNING outcomes are recorded as soft quality signals but do not block acceptance. This prevents the pipeline from discarding explanations that satisfy the required structural checks but exhibit minor stylistic deviations.

The rule-based verification checks fall into three categories. First, structural checks ensure that the generated output conforms to the trace schema. These checks verify that all required fields are present, every referenced evidence identifier belongs to the provided evidence set, each claim cites at least one evidence source, and the anomaly signature follows the canonical COMPONENT__ERROR_TYPE format. Second, grounding checks test whether each claim has lexical support in the evidence it references. The verifier performs this check at two levels. At the document level, it verifies whether claim keywords appear somewhere in the cited evidence document. At the span level, it checks whether the keywords appear specifically within the cited Ei-Lj line spans. The span-level check is more restrictive and is intended to detect claims that cite a valid evidence identifier but point to irrelevant lines. Third, severity checks address a common hallucination pattern in which the LLM cites ordinary INFO-level log lines as indicators of anomalies. This check verifies whether the cited query-session lines contain valid severity or dataset-specific anomaly cues, such as ERROR, WARN, FATAL, exception terms, or configured structural anomaly tags. If no such cue is found, the verifier flags the explanation as a probable hallucination according to the dataset-specific verification configuration.

3.2.5 Signature Generator

Aggregating LLM-generated explanations into an interpretable anomaly landscape requires a consistent signature naming scheme. The signature generator assigns each explanation a standardized label in the COMPONENT__ERROR_TYPE format, such as KERNEL__DATA_TLB_ERROR or

DATANODE__BLOCK_WRITE_FAILURE. These signatures support deduplication, frequency counting, and cross-session pattern analysis. The Reasoner first produces a raw signature name as part of its structured output. Rather than selecting from a fixed, predefined taxonomy, the framework allows the LLM to generate open-ended signature names, guided by dataset-specific component vocabularies and examples in the system prompt. A deterministic normalization step then maps the raw signature to a canonical form:

$$\sigma(T) = \text{Normalize}(\sigma_{\text{raw}}(T)) \quad (24)$$

where $\sigma_{\text{raw}}(T)$ is the raw signature generated by the LLM for explanation T , and $\sigma(T)$ is normalized canonical signature. The normalization process standardizes separators, removes unnecessary severity terms, fixes component prefixes, and consolidates lexical variants into standard error-type names.

This step is necessary because the same failure pattern may otherwise be represented by multiple surface forms. For example, raw outputs such as `KERNEL__FATAL__DATA_TLB_ERROR` or `KERNEL__FATAL__data TLB error interrupt` are normalized to `KERNEL__DATA_TLB_ERROR`. This canonicalization prevents signature fragmentation and makes exact-match aggregation more reliable. Signature format compliance is also checked by the verifier as a soft warning, as described in Section 3.2.4.

3.3 Gating

The Reasoner pipeline invokes an LLM to generate explanations for anomalous sessions flagged by the Screener. In large-scale log streams, explaining every flagged anomaly can be costly, especially when many sessions share similar failure patterns. The Gating module is placed between the Screener and the Reasoner to control which anomalous sessions receive full LLM-based explanations under a specified budget.

Let $A = \{x_1, \dots, x_N\}$ denote the set of sessions predicted as anomalous by the Screener, where $N = |A|$. Given a budget ratio $B \in (0, 1]$, the gating module selects $K = \lfloor B \cdot N \rfloor$ sessions for explanation. The selected subset $\mathcal{G}_B \subseteq A$ is defined as:

$$\mathcal{G}_B = \text{TopK}_{x_i \in A}(u(x_i), K) \quad (25)$$

where $u(x) = 1 - m(x)$ is the Screener uncertainty score and $m(x) = |p_{\text{anomalous}}(x) - p_{\text{normal}}(x)|$ is the classification margin. Sessions with low margin indicate lower Screener confidence and are therefore prioritized for explanation. When $B = 1$, the gating module reduces to the explain-all setting, in which every Screener-flagged anomaly is routed to the Reasoner. This setting serves as the reference point for evaluating the cost-coverage trade-off in Section 4.

4. Experiments

A series of experiments is conducted to evaluate the proposed Screener–Reasoner framework across two dimensions: (1) detection accuracy of the Screener, and (2) explanation quality, faithfulness, and cost efficiency of the Reasoner pipeline. The evaluation is structured around three research questions. RQ1 determines whether the framework generates traceable, evidence-grounded explanations that meet both automated verification and human judgment. RQ2 measures the contribution of retrieval-augmented evidence grounding by comparing it to a no-retrieval baseline. RQ3 analyzes the cost-quality trade-off under confidence-based gating at different budget levels. Sections 4.1 and 4.2 present the Screener’s detection performance, including comparisons with baseline models and an ablation study of architectural options. Sections 4.3 to 4.6 sequentially address the three research questions. All experiments are performed on two public log datasets, BGL and HDFS, using an Intel Core i9-13900HX CPU at 2.20 GHz, 32 GB of RAM, and an NVIDIA GeForce RTX 4080 GPU with 12 GB of memory, running Ubuntu 24.04 LTS with PyTorch 2.5.1.

4.1 Benchmark Datasets

4.1.1 Public Datasets and Splitting Strategy

This work evaluates the proposed framework on two publicly available log datasets: Blue Gene/L (BGL) and Hadoop Distributed File System (HDFS) [36]. These datasets are widely used in log anomaly detection and represent different types of large-scale systems. BGL contains logs collected from a Blue Gene/L supercomputer system at Lawrence Livermore National Laboratory (LLNL) in Livermore, California. The HDFS dataset contains logs generated by Hadoop-based MapReduce jobs running on Amazon EC2 nodes.

After session construction, each dataset is split into training (70%), validation (15%), and test (15%) sets. The training set is used to train the Screener and to construct the train-only evidence store. The validation set is used for hyperparameter tuning and early stopping, while the test set is held out for final evaluation. This split ensures that detection performance, retrieval-based explanation, and evidence grounding are evaluated on unseen sessions.

4.2 Screener

For the proposed Screener configuration, the minority class oversampling ratio is set to 0.4, the low-rank projection factor k in linear self-attention is set to 32, and the embedding dimension is set to 128. To preserve the full raw-log context, the input sequence is not truncated. The maximum sequence length is set to the length of the longest session observed during input construction. These settings allow the Screener to process long parser-free log sessions while keeping the model lightweight and computationally efficient.

4.2.1 Loss Function and Class Weighting

The proposed Screener adopts a weighted cross-entropy loss function to address the class imbalance commonly observed in log datasets. During training, a higher penalty is assigned to misclassified anomalous sessions, reducing the tendency of the model to favor the majority normal class. Let $\mathbf{z} = [z_0, z_1] \in \mathbb{R}^2$ represent the logit vector output by the classification layer, $y \in \{0,1\}$ be the ground truth label, and the class weight vector $\mathbf{w} = [w_0, w_1]$. The loss for a single training sample is defined as:

$$L(\mathbf{z}, y) = -w_y \cdot \log \left(\frac{e^{z_y}}{\sum_{j=0}^1 e^{z_j}} \right) = -w_y \cdot \left(z_y - \log \sum_{j=0}^1 e^{z_j} \right) \quad (26)$$

where w_y denotes the weight assigned to the class. In this work, the weight $w_0 = 1.0$ is assigned to the normal class, and the weight $w_1 = 4.0$ is assigned to the anomalous class. This weighting encourages the Screener to pay more attention to anomalous sessions and reduces the bias caused by the skewed class distribution in the training set.

4.2.2 Optimizer and Learning Rate Schedule

The proposed Screener employs the AdamW optimizer [38], which decouples the weight decay term from gradient updates to improve regularization during training. The optimizer is configured with a weight decay of 10^{-2} and an epsilon value of 10^{-8} . The initial learning rate is 5×10^{-4} . A ReduceLROnPlateau scheduler is used to reduce the learning rate when validation performance stops improving, with the reduction factor set to 0.1.

4.2.3 Baseline Comparison

Evaluation Metrics

To evaluate Screener performance, this work uses widely adopted metrics: Precision, Recall, and F1-score. True Positive (TP) denotes anomalous sessions correctly identified as anomalous. False Positive (FP) denotes normal sessions incorrectly identified as anomalous. True Negative (TN) denotes normal sessions correctly identified as normal. False Negative (FN) denotes anomalous sessions incorrectly identified as normal.

Precision measures the proportion of predicted anomalous sessions that are truly anomalous. Recall measures the proportion of real anomalous sessions that are effectively detected. F1-Score is the harmonic mean of precision and recall, providing a balanced measure when both false alarms and missed anomalies are important.

$$Precision = \frac{TP}{TP + FP} \quad (27)$$

$$Recall = \frac{TP}{TP + FN} \quad (28)$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (29)$$

Comparison with Baseline Detection Models

The performance metrics of the proposed Screener are compared with the reported results of existing methods on the HDFS and BGL datasets, as shown in Table 4. The results of DeepLog [4], LogCluster [3], LogAnomaly [39], SwissLog [40], PLELog [41], and FSMFLog [8] are taken from Niu et al. [8], while the LogLAA results are reported by Gao et al. [12]. The results of the proposed Screener, implemented as AllLinLog, are obtained in this work. Most compared methods rely on log parsing to transform raw logs into structured templates or event sequences, whereas AllLinLog performs anomaly detection directly on raw logs without log parsing. The proposed Screener achieves the highest F1-score on HDFS and matches LogLAA’s near-ceiling F1-score on BGL. Since the reported results are obtained under different preprocessing and data-splitting settings, the comparison should be interpreted as a contextual benchmark rather than a strictly controlled reproduction.

Table 4: Comparison of detection performance on public datasets. The best results are highlighted in bold.

	HDFS			BGL		
Methods	Precision	Recall	F1-score	Precision	Recall	F1-score
DeepLog	0.879	0.812	0.844	0.823	0.807	0.815
LogCluster	0.851	0.745	0.794	0.578	0.829	0.681
LogAnomaly	0.965	0.834	0.895	0.859	0.912	0.885
SwissLog	0.960	0.885	0.921	0.941	0.852	0.894
PLELog	0.923	0.734	0.818	0.911	0.811	0.858
FSMFLog	0.981	0.952	0.966	0.972	0.951	0.961
LogLAA	0.916	0.960	0.937	0.999	0.999	0.999
AllLinLog	0.997	0.995	0.996	0.999	0.999	0.999

4.2.4 Screener Ablation Study

To evaluate the effectiveness and efficiency of the Screener architecture, an ablation study is conducted on the BGL dataset. Since the purpose of this ablation is to analyze architectural sensitivity rather than dataset-specific performance, BGL is used as the representative dataset. The study examines

three architectural factors: the low-rank approximation factor k , the number of linear self-attention encoder layers, and the trade-off between the linear self-attention encoder and a standard Transformer encoder. All evaluations use the same preprocessing pipeline, loss function, optimizer, and training settings described above. Overall, this ablation assesses whether the proposed Screener can maintain strong detection performance while reducing the computational cost of modeling long raw-log sessions.

(A) Effect of Low Rank Approximation Factor k

The linear self-attention encoder replaces full self-attention with a low-rank approximation to reduce the quadratic time and memory complexity of standard self-attention. To assess the effect of k , the Screener is evaluated on the BGL dataset with $k \in \{4, 8, 16, 32, 64, 128\}$. Other settings are kept fixed, including an embedding dimension of 128, one linear self-attention encoder layer, and four attention heads. As shown in **Fig. 8**, $k=32$ achieves the best F1-score of 0.99888.

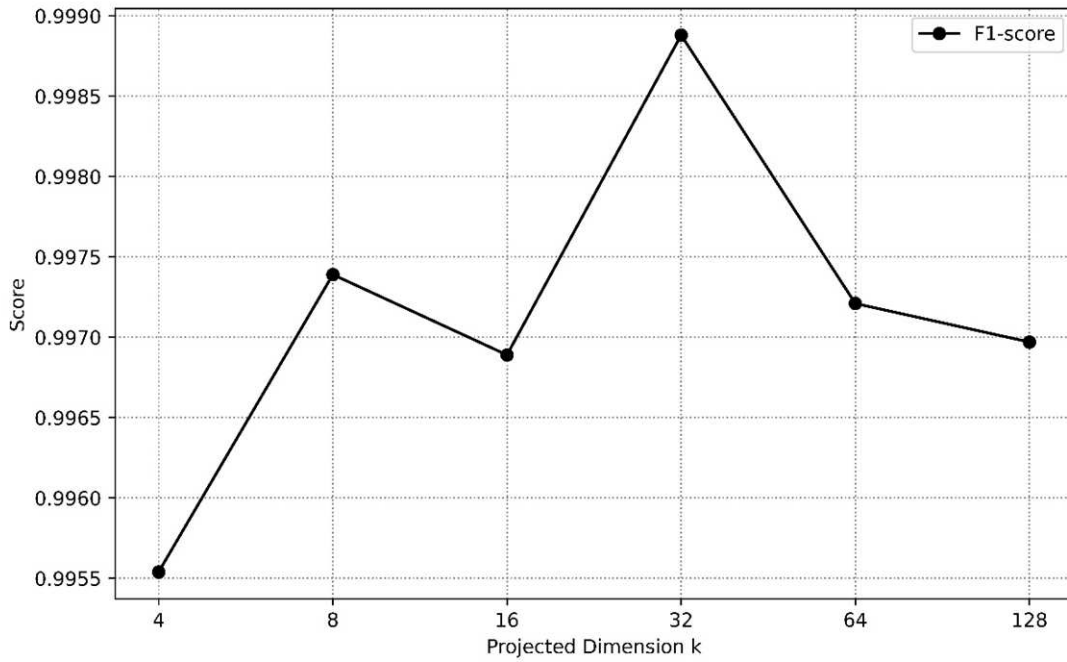


Fig. 8 The effect of different low-rank approximation factor k

(B) Effect of Encoder Capacity

The Screener can increase model capacity by stacking more linear self-attention encoder layers and using more attention heads. To examine whether a larger encoder improves anomaly detection performance, the Screener is evaluated on the BGL dataset under different encoder-capacity settings. The evaluation starts from a lightweight configuration with one encoder layer and 4 attention heads, and increases up to four encoder layers and 32 attention heads, while keeping other model parameters unchanged. Based on the results shown in **Fig. 9**, the lightweight configuration with one encoder layer achieves the best F1-score of 0.99888. This result suggests that increasing encoder capacity does not improve detection performance in this setting, and that a shallow linear self-attention encoder is sufficient for the Screener.

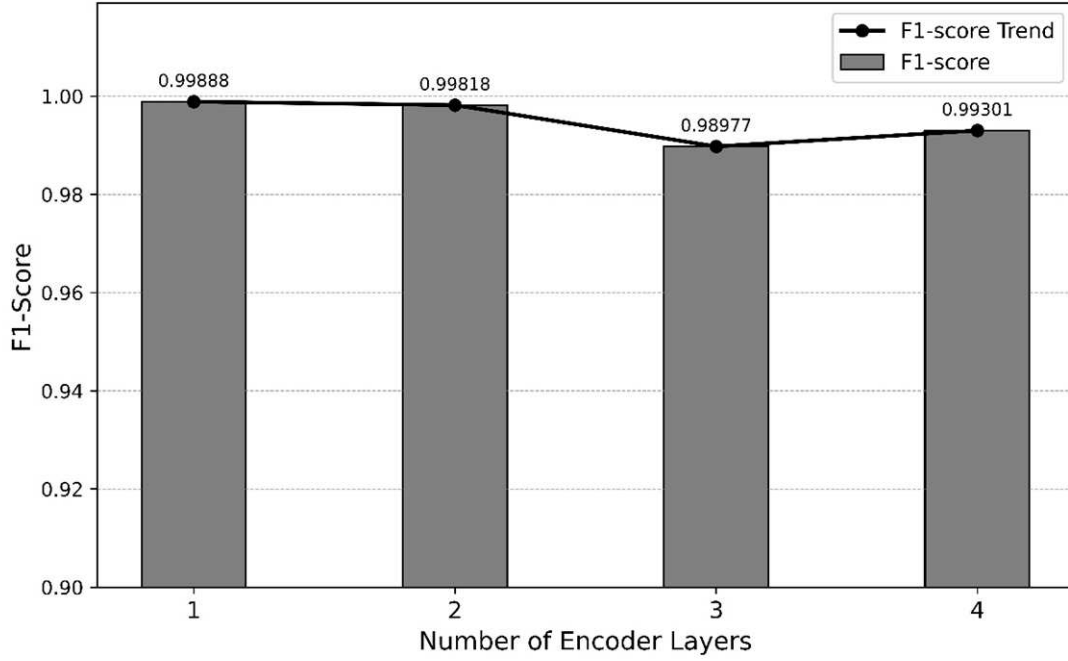


Fig. 9 Impact of encoder layer depth on F1-score

(C) Comparison with Standard Transformer Encoder

The proposed Screener uses a linear self-attention encoder to reduce the computational complexity of full self-attention from quadratic to linear complexity through low-rank approximation. To evaluate the performance cost of this approximation, the linear self-attention encoder is replaced with a standard Transformer encoder, and the detection performance of the two encoders is compared on the BGL dataset. From the results shown in **Fig. 10**, the linear self-attention encoder reduces computational complexity with only a 0.079% drop in F1-score compared with the standard Transformer encoder, from 0.99967 to 0.99888. This result indicates that the linear self-attention encoder preserves near-Transformer detection performance while providing a more efficient architecture for long raw-log sessions.

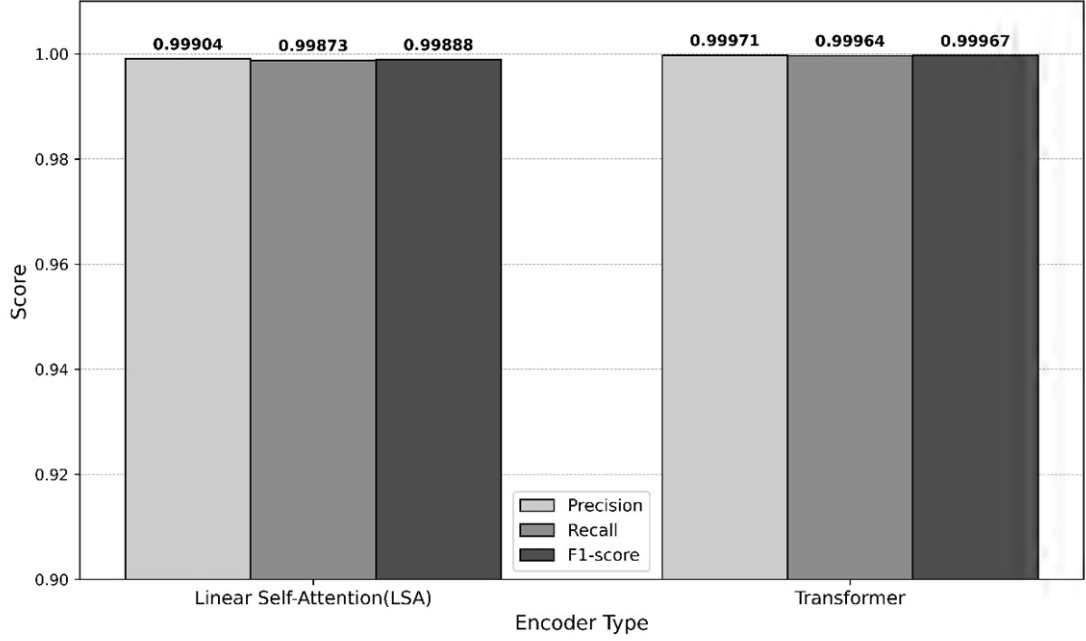


Fig. 10 Comparison of anomaly detection performance between the linear self-attention encoder and the transformer encoder

In addition to maintaining near-Transformer detection performance, the linear self-attention encoder provides substantial efficiency gains, as Table 5 illustrates. The comparison is conducted with batch sizes of 1 and 16. A batch size of 1 represents real-time anomaly detection, while a batch size of 16 simulates batched inference when multiple log sessions are processed together. Memory usage is measured using `torch.cuda.max_memory_allocated`, which records peak GPU memory allocation during inference. This measurement is used because PyTorch pre-allocates and caches GPU memory, making before-and-after memory differences less reliable. For inference time, each configuration is executed 100 times, and the average runtime is reported.

Table 5: Benchmark analysis: transformer encoder vs. linear self-attention (LSA) encoder (Batch size 1 and 16)

Encoder	Batch size	Peak Memory used	Avg Inference Time	Throughput
Transformer	1	893.22 MB	4.7541 ms	210.35 samples/sec
LSA	1	709.09 MB	0.8143 ms	1228.07 samples/sec
Transformer	16	3824.70 MB	51.2782 ms	312.02 samples/sec
LSA	16	911.70 MB	3.9677 ms	4032.52 samples/sec

As shown in Tables 6–8, the linear self-attention encoder offers clear efficiency advantages over the standard Transformer encoder. In terms of memory usage, the Transformer encoder increases from 893.22 MB to 3,824.70 MB when the batch size increases from 1 to 16, whereas the linear self-attention

encoder increases only from 709.09 MB to 911.70 MB. A similar trend is observed in inference time: the Transformer encoder becomes approximately 10.8 times slower at batch size 16, while the linear self-attention encoder increases by only about 4.9 times. At batch size 16, the linear self-attention encoder also achieves 12.9 times the throughput of the Transformer encoder. These results show that the standard Transformer encoder becomes inefficient when processing larger batches of long raw-log sessions, whereas the linear self-attention encoder maintains substantially better memory usage, latency, and throughput.

Table 6: Memory efficiency comparison between transformer encoder and linear self-attention (LSA)

encoder

Encoder	Batch Size	Memory Growth (1 $\rightarrow$ 16)
Transformer	1 $\rightarrow$ 16	893.22 MB $\rightarrow$ 3824.70 MB (+2,931.48 MB)
LSA	1 $\rightarrow$ 16	709.09 MB $\rightarrow$ 911.70 MB (+202.61 MB)

Table 7: Inference time efficiency comparison between transformer encoder and linear self-attention

(LSA) encoder

Encoder	Batch Size	Inference Time Increase (1 $\rightarrow$ 16)
Transformer	1 $\rightarrow$ 16	4.7541 ms $\rightarrow$ 51.2782ms (10.8x increase)
LSA	1 $\rightarrow$ 16	0.8143 ms $\rightarrow$ 3.9677ms (4.9x increase)

Table 8: Throughput comparison between transformer encoder and linear self-attention (LSA) encoder

Encoder	Batch Size	Throughput (samples/sec)
Transformer	1 $\rightarrow$ 16	210.35 $\rightarrow$ 312.02
LSA	1 $\rightarrow$ 16	1228.07 $\rightarrow$ 4032.52

4.3 Reasoner

This section describes the experimental configuration and evaluation metrics for the Reasoner pipeline. The Reasoner is evaluated from three perspectives: automated verification, human evaluation, and cost efficiency. Automated verification checks whether the generated explanations satisfy the required structured and evidence-grounding constraints. Human evaluation assesses whether the explanations are semantically correct, complete, and sufficiently grounded in the cited evidence. Cost efficiency is measured by token consumption and LLM latency.

4.3.1 LLM configuration

The Reasoner pipeline uses GPT-5.1 as its core language model, accessed through the OpenAI chat-compatible API. JSON mode is enabled to produce parseable structured outputs, which are subsequently checked by the verifier. The sampling temperature is set to 0.1 to reduce generative variance while allowing minimal flexibility in explanation wording. Each request includes the full system prompt (including dataset context, component vocabulary, JSON schema, and evidence blocks E_0 through E_5) and the user prompt, which contains the query session. The LLM produces a single-turn explanation with a maximum of 2048 completion tokens and a 120-second timeout. The same model configuration and decoding parameters are used across all Reasoner experiments.

4.3.2 Evaluation Metrics

Reasoner quality is measured along three complementary axes: automated verification, human evaluation, and cost efficiency. In automated verification, the Verification Pass Rate (VPR) is defined as the fraction of explained sessions whose outputs pass all required verifier checks without failures. Because VPR is binary at the session level, it captures worst-case compliance; a single failed check is sufficient to mark the explanation as failed. We also report evidence utilization, defined as the fraction of retrieved evidence slots (E_1 through E_5) explicitly cited by at least one claim, and signature coverage, the proportion of training-derived anomaly pattern cards that appear in the full run output across all test sessions. For the cost dimension, we record total token consumption, average tokens per session, and LLM latency at the mean and 95th percentile.

Human evaluation complements the automated metrics on a stratified random sample of 50 sessions per dataset. A domain expert evaluates each explanation based on three 5-point criteria: correctness, completeness, and evidence grounding. Correctness measures the factual accuracy of the generated explanation. Completeness measures whether the main anomaly signals in the logs are covered. Evidence grounding measures whether the cited evidence adequately supports the generated claims. A binary actionable label is also assigned to indicate whether an engineer can act on the explanation without first reviewing the raw logs. These human evaluation results provide a complementary semantic assessment beyond that of the rule-based verifier.

4.4 RQ1: Explanation Quality

4.4.1 Automated Verification Results

Table 9 summarizes the Reasoner pipeline results on both datasets. The pipeline explains all Screener-flagged anomaly sessions: 5,850 for BGL and 2,527 for HDFS. Within the scope of rule-based verification, all generated explanations pass the verifier checks, resulting in a verification pass rate (VPR) of 100% on both datasets. This result indicates that the combination of JSON-mode structured output and the evidence-grounded prompt design described in Section 3.2 can consistently produce structurally

valid and citation-supported explanations at scale. However, this high VPR should be interpreted according to the verifier’s design scope. The verifier targets structural and grounding failures, including invalid schema fields, unsupported evidence identifiers, malformed span references, and insufficient grounding. It does not assess whether the natural language in each claim is semantically correct, which is addressed separately through human evaluation in Table 10. A session passes VPR only when its output satisfies the required schema, and all cited evidence identifiers and spans are valid. Therefore, VPR represents the minimum structural guarantee provided by the framework rather than a complete measure of explanation correctness. The average token usage per session is 6,496 for BGL and 8,120 for HDFS. The average LLM latency is 7.6 seconds for BGL and 8.1 seconds for HDFS, with the 95th percentile not exceeding 14.4 seconds in either dataset.

Table 9: Reasoner pipeline results on BGL and HDFS

Metric	BGL	HDFS
Anomalies Explained	5,850	2,527
Verification pass rate	100%	100%
Unique signatures	237	98
Avg tokens/session	6,496	8,120
Mean latency (seconds)	7.6	8.1
P95 latency (seconds)	13.2	14.4

4.4.2 Human Evaluation

Human evaluation results on a stratified random sample of 50 sessions per dataset are shown in Table 10. Across both datasets, correctness scores exceed 4.9 out of 5, and completeness averages 4.98 for BGL and 4.96 for HDFS. These results indicate that the generated explanations are factually accurate and capture the primary anomaly signals in each session. Evidence-grounding scores are slightly lower, with 4.8 for BGL and 4.64 for HDFS. This difference is partly due to HDFS sessions sometimes consisting only of brief INFO-level messages, which provide less lexical evidence for supporting the generated claims.

Table 10: Human evaluation results (5-point scale, n = 50 per dataset)

Dimension	BGL (mean $\pm$ std)	HDFS (mean $\pm$ std)
Correctness	4.94 $\pm$ 0.24	4.92 $\pm$ 0.27
Completeness	4.98 $\pm$ 0.14	4.96 $\pm$ 0.20
Evidence grounding	4.80 $\pm$ 0.40	4.64 $\pm$ 0.52

4.4.3 Anomaly Signature Analysis

The signature generator identifies 237 unique anomaly types across BGL and 98 across HDFS, providing a signature-based summary of observed anomaly patterns. The distribution follows a Zipf-like pattern: a few dominant signatures account for most sessions (for example, `KERNEL__DATA_TLB_ERROR` alone covers 2,337 of 5,850 BGL anomalies, or 39.9%), while the long tail includes 128 singletons in BGL (54%) and 36 in HDFS (36.7%). **Fig. 11** visualizes this distribution as a rank-frequency plot, and Table 11 lists the five most frequent signatures per dataset. The high singleton rate should not be interpreted simply as noise. Manual inspection suggests that many singleton signatures correspond to plausible rare failure patterns, such as one-time firmware faults or transient network errors. This long-tail discovery provides a practical benefit of the framework because operators can filter recurring issues by dominant signatures while still retaining visibility into rare or newly emerging anomaly patterns.

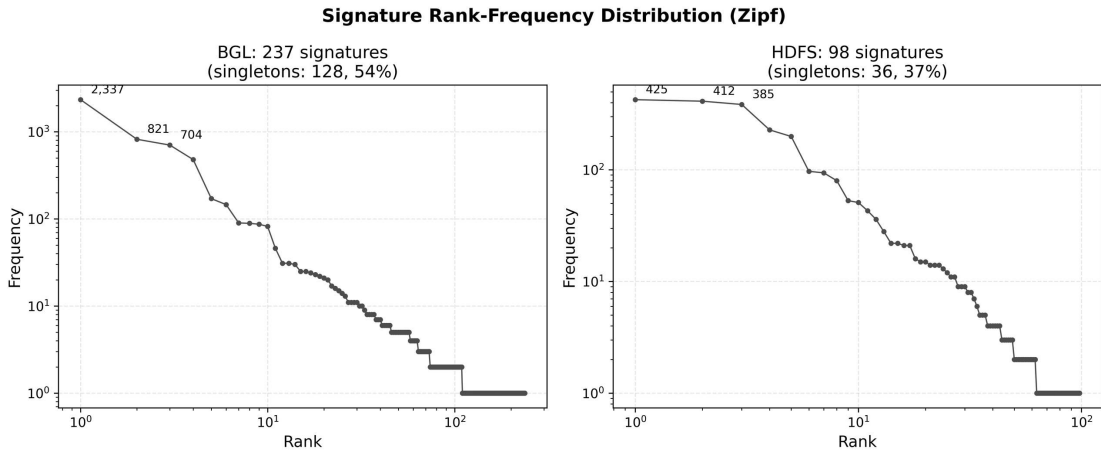
**Fig. 11** Rank-frequency distribution of anomaly signatures (log scale)

Table 11: Top 5 frequent anomaly signatures per dataset

Rank	BGL signature	Count	HDFS Signature	Count
1	KERNEL__DATA_TLB_ERROR	2,337	DATANODE__BLOCK_WRITE_STREAM_READ_FAILURE	425
2	KERNEL__DATA_STORAGE_INTERRUPT	821	FSDATASET__BLOCK_DELETION_METADATA_MISMATCH	412
3	APP__CIOD_STREAM_ERROR	704	NAMENODE__INCOMPLETE_PIPELINE	385
4	KERNEL__LUSTRE_MOUNT_FAILED	480	FSDATASET__BLOCK_DELETION_METADATA_MISS	228
5	KERNEL__KERNEL_TERMINATED	171	NAMENODE__EXCESS_REPLICATION	199

4.5 RQ2: RAG ablation

4.5.1 Ablation Design

Research question 2 investigates whether retrieval-augmented generation (RAG) improves evidence grounding in generated explanations, or whether the LLM can produce sufficiently grounded explanations using only the query session. This study evaluates this question through three complementary metrics.

Two metrics, Faithfulness and Context Precision, are adopted from the RAGAs framework [26] to fit the evidence-slot setting of this work. A third metric, Grounding Breadth, is introduced to measure external evidence usage in evidence-grounded explanations.

- (1) Faithfulness: It measures whether each generated claim is factually supported by the evidence spans that it cites. This work adopts a RAGAs-inspired LLM-as-a-judge protocol to evaluate whether each generated claim is factually supported by the evidence spans it cites. Unlike the original RAGAs setting, where free-form answers are decomposed into statements, the proposed Reasoner already produces structured atomic claims. Therefore, each claim is directly used as an evaluation unit. For a given session i , let S_i denote the set of claims generated for that session, and let $V_i \subseteq S_i$ denote the subset of claims judged as factually supported by their cited evidence spans. The per-session faithfulness score is defined as:

$$f_i = \frac{|V_i|}{|S_i|} \quad (30)$$

The overall semantic faithfulness is computed as the mean score over all N evaluated sessions:

$$\text{Faithfulness} = \frac{1}{N} \sum_{i=1}^N f_i \quad (31)$$

To determine whether a claim belongs to V_i , each claim is evaluated by an independent judge model. The judge is provided with the claim text and the actual log lines extracted from the evidence spans cited by that claim. A claim is counted as supported only if the judge determines that the cited log lines factually support the claim. Claims with unresolved evidence identifiers

or invalid span references are counted as unsupported.

- (2) Context Precision@K ($K = 4$): This is used as a rank-aware retrieval quality metric. For each ranked evidence slot E_k , where $k = 1, \dots, K$, a binary relevance judgment v_k indicates whether the evidence slot is cited by at least one generated claim. In this work, K is set to 4, corresponding to the four BM25-retrieved anomaly evidence slots E_1 through E_4 . The normal contrast slot E_5 is not included in Context Precision@4.

Context Precision@K is then:

$$\text{Context Precision @ K} = \frac{\sum_{k=1}^K \text{Precision @ } k \cdot v_k}{\sum_{k=1}^K v_k} \quad (32)$$

where

$$\text{Precision @ } k = \frac{\sum_{j=1}^k v_j}{k} \quad (33)$$

Higher values indicate that the retriever ranks cited and relevant evidence earlier. If no retrieved evidence slot is cited, Context Precision@K is defined as 0.

- (3) Grounding Breadth: this metric measures the fraction of claims that cite at least one external evidence slot, namely E_1 through E_5 , rather than only the query session E_0 . It is central to RQ2 because it captures whether RAG enables the Reasoner to cross-reference the current anomalous session with retrieved historical anomaly patterns and normal contrast evidence.

The ablation compares two conditions. In the RAG-on condition, the full pipeline provides E_0 (query session), E_1 through E_4 as BM25-retrieved anomaly sessions, and E_5 as the normal contrast session. In the no-RAG condition, the retrieval step is skipped, and only E_0 is included in the prompt. Both conditions use the same LLM configuration, prompt template, and verification loop described in Section 4.3.1. The ablation is performed on a stratified sample that covers all unique anomaly signatures, including 338 sessions covering 237 signatures for BGL and 245 sessions covering 98 signatures for HDFS.

A supplementary analysis is conducted to examine whether the train-only BM25 corpus provides sufficient retrieval coverage. For 100 randomly selected anomalous sessions per dataset, the top-4 retrieved evidence items are compared between the train-only corpus and an expanded corpus containing the full dataset. The overlap rate above 85% suggests that the train-only evidence store provides adequate retrieval coverage without requiring access to test data.

4.5.2 Faithfulness and Grounding Impact

Table 12 reports the metrics computed on the RAG-on pipeline for all anomaly sessions in BGL and HDFS. Context Precision@4 is 0.991 for BGL and 0.946 for HDFS, indicating that the BM25 retriever consistently ranks relevant evidence in the top positions. Evidence utilization, which is the fraction of available evidence slots cited by at least one claim, is 0.670 for BGL and 0.463 for HDFS. This difference

is consistent with the dataset-level patterns observed in RQ1, where HDFS sessions frequently contain only INFO-level messages and provide less distinctive lexical content for cross-referencing.

Table 12: Baseline metrics (RAG-on) for all anomaly sessions

Metric	BGL (n = 5,850)	HDFS (n = 2,527)
Context Precision@4	0.991 (+/- 0.053)	0.946 (+/- 0.152)
Evidence Utilization	0.670	0.463

Expanding the BM25 corpus from training data to the full dataset caused less than 13% of the top-4 anomaly retrievals to change on either dataset. The top-1 retrieved evidence, which has the highest BM25 relevance score, matches at 88% for BGL and 86% for HDFS. The mean number of new documents introduced by the full dataset corpus is 0.51 (BGL) and 0.31 (HDFS) out of 4 slots. These results indicate that the train-only evidence corpus provides sufficient retrieval coverage, and expanding the corpus to include the entire dataset offers limited additional benefit.

Table 13 presents a direct comparison between RAG-on and no-RAG conditions. Faithfulness is evaluated on the 338 BGL and 245 HDFS ablation sessions with matched RAG-on and no-RAG explanations. An independent GPT-4.1 judge determines whether each claim is factually supported by the actual log lines referenced in its cited evidence spans. Compared with the no-RAG condition, RAG-on improves Faithfulness by 12.4 percentage points on BGL and 17.1 percentage points on HDFS. These improvements indicate that retrieved evidence acts as a factual constraint during explanation generation. Grounding Breadth provides a clearer view of how retrieval changes the generated explanations. With RAG enabled, 66.7% of claims cite external evidence slots, including E_1 through E_5 , and connect the query session to known anomaly patterns or normal-behavior baselines in the evidence store. Without RAG, Grounding Breadth drops to 0.000, because no external evidence is provided in the prompt. The difference of 0.667 is consistent across both datasets because the prompt requires three claim types, where the pattern-match and contrast claims naturally depend on external evidence.

Furthermore, in the no-RAG condition, all sessions, including 338 BGL sessions and 245 HDFS sessions, contain one phantom citation. Specifically, the contrast claim in each session cites an E_5 slot that does not exist in the evidence mapping. This result shows that the LLM attempts to produce a normal contrast claim even when no normal evidence is provided. The absence of retrieved evidence therefore does not suppress cross-referencing behavior. Instead, it makes the cross-reference unsupported and structurally invalid.

Table 13: RAG-on vs no-RAG ablation results

Metric	BGL	BGL	HDFS	HDFS
	RAG-on	no-RAG	RAG-on	no-RAG

Faithfulness	0.714	0.590	0.732	0.561
Grounding Breadth	0.667	0.000	0.667	0.000
Claims per session	3.0	3.0	3.0	3.0
Phantom citations	0	338	0	245

4.6 RQ3: Cost-Quality Trade-off under Gating

4.6.1 Gating Strategy Comparisons

Research question 3 examines whether budget-limited gating can reduce LLM inference cost while maintaining coverage of the anomaly signature catalog. Given N anomalous sessions, the gating module selects $K = \lfloor B \times N \rfloor$ sessions for explanation, where $B \in (0,1]$ is the budget ratio. Four deployable strategies are evaluated.

- (1) Random: selects K sessions uniformly at random without replacement.
- (2) Uncertainty: selects the top- K sessions according to $u(x) = 1 - \text{margin}(x)$, where margin is the Screener’s softmax margin between the normal and anomalous classes. A higher $u(x)$ indicates lower Screener confidence and is expected to have higher potential value for an LLM explanation.
- (3) Novelty: selects the top- K sessions according to $n(x) = 1 - \text{bm25_max}(x) / \max(\text{bm25_max})$, where $\text{bm25_max}(x)$ is the highest BM25 similarity between session x and all documents in the evidence store, and $\max(\text{bm25_max})$ is the maximum value across all sessions in the dataset. Sessions with low lexical similarity to the evidence store are assumed to represent potentially novel fault types.
- (4) Combined: selects the top- K sessions according to $s(x) = \alpha u(x) + (1 - \alpha)n(x)$, where α is a tunable weight that balances uncertainty and novelty.

The primary evaluation metric is Signature Coverage, defined as the proportion of unique anomaly signatures in the full anomaly corpus that are represented among the selected K sessions. Across all strategies and budget levels on both datasets, every selected session achieves a Verification Pass Rate of 1.0. This result indicates that the gating strategies mainly affect signature coverage and inference cost. The generation and verification procedure remains unchanged across strategies.

Table 14 presents signature coverage across strategies at three representative budget levels. Uncertainty gating achieves the highest coverage among the evaluated strategies when $B \leq 0.5$ on both datasets. At $B = 0.1$, Uncertainty achieves 49.8% coverage on BGL and 55.1% coverage on HDFS, surpassing Random by 22.5 and 13.5 percentage points, respectively. At $B = 0.2$, its advantage over Random widens to 25.4 percentage points on BGL and 18.9 percentage points on HDFS. **Fig. 12** visualizes the full coverage-cost Pareto curves for all strategies.

Table 14: Signature coverage (%) by gating strategy and budget (B)

Strategy	BGL	BGL	BGL	HDFS	HDFS	HDFS
	$B = 0.1$	$B = 0.2$	$B = 0.5$	$B = 0.1$	$B = 0.2$	$B = 0.5$
Random	27.3	40.8	67.8	41.6	54.6	77.9
Uncertainty	49.8	66.2	73.0	55.1	73.5	93.9
Novelty	33.3	35.0	35.0	3.1	5.1	22.4
Combined ($\alpha=0.5$)	33.8	35.4	35.4	4.1	6.1	23.5

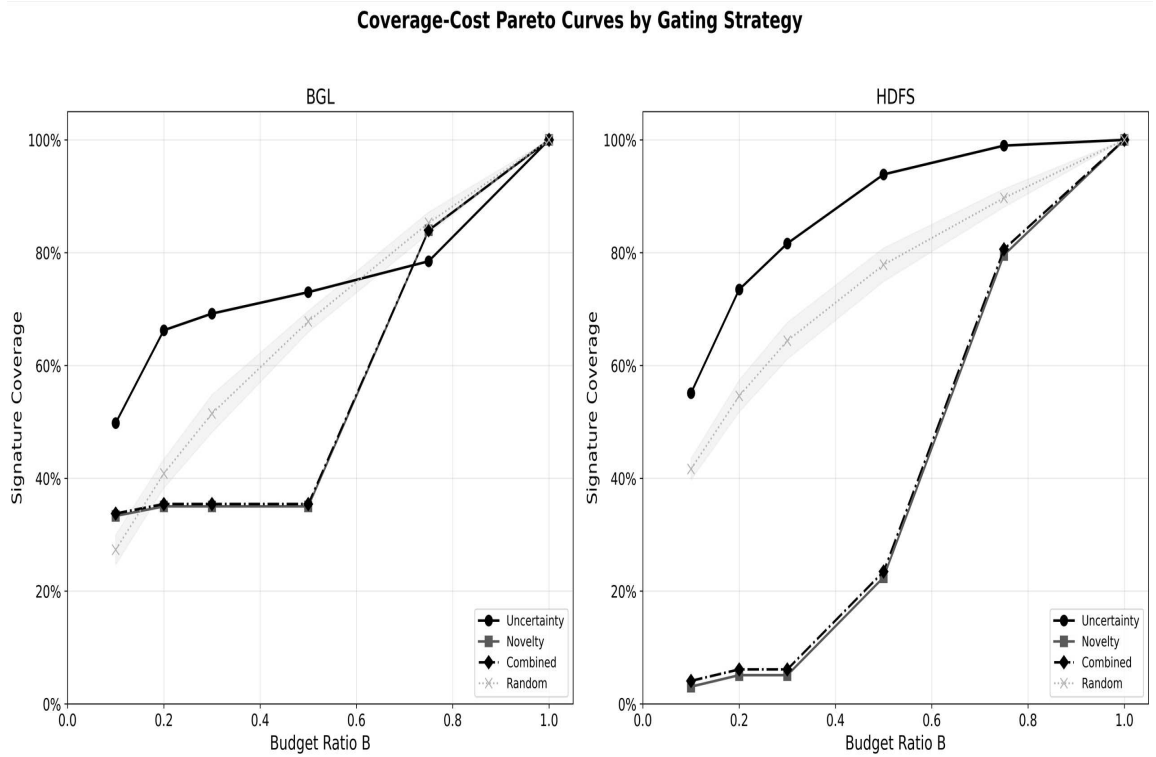


Fig. 12 Signature Coverage vs. Budget Ratio Across Gating Strategies

The Novelty strategy underperforms on BGL and performs substantially worse on HDFS. At B equals 0.1, it achieves only 3.1% coverage on HDFS, compared with 55.1% achieved by Uncertainty gating. This underperformance is mainly due to the limited discriminative power of the novelty signal. The coefficient of variation (CV) of the novelty score $n(x)$ is only 0.29 on BGL and 0.26 on HDFS, indicating that most sessions receive similar novelty values. In contrast, the uncertainty score $u(x)$ yields CV values of 51.84 and 43.79, showing much greater variation across sessions. The Combined strategy inherits this limitation because the two score components are not on the same numerical scale. The mean of $u(x)$ is much smaller than that of $n(x)$, with 0.0001 versus 0.61 on BGL and 0.0004 versus 0.72 on HDFS. Therefore, when $\alpha = 0.5$, the combined score is dominated by the novelty term. As a

result, the coverage of the Combined strategy closely follows that of Novelty across budget levels. The budget sensitivity analysis in Section 4.6.2 further shows that the best setting is $\alpha = 1.0$, where the Combined strategy reduces to pure Uncertainty gating.

4.6.2 Budget Sensitivity Analysis

Sweeping the mixing weight α from 0.0 to 1.0 across three budget levels shows that the best weight is 1.0 for both datasets. This setting corresponds to pure uncertainty-based selection with no novelty contribution. Adding any novelty weight monotonically reduces coverage, which is consistent with the coefficient-of-variation analysis in Section 4.6.1. Therefore, the production gating module uses $u(x) = 1 - \text{margin}(x)$ as the sole selection criterion.

When $B = 0.20$, uncertainty gating covers 66.2% of BGL signatures and 73.5% of HDFS signatures, while reducing LLM token usage by 79.6% and 75.4%, respectively. Increasing the budget to $B = 0.50$ reveals a dataset-dependent trade-off. On BGL, coverage increases by only 6.8 percentage points to 73.0%, suggesting a plateau due to its larger signature space of 237 types. On HDFS, coverage increases by 20.4 percentage points to 93.9%, approaching near-complete coverage because of its smaller signature space of 98 types. These results suggest that $B = 0.20$ provides a practical balance between cost and coverage when broad but incomplete coverage is acceptable, whereas $B = 0.50$ is preferable when near-total signature coverage is required and the signature space is limited.

4.7 Discussion

The three research questions indicate that the proposed framework should be interpreted as a connected decision-and-explanation pipeline, rather than a set of independent modules. RQ1 shows that the Reasoner can consistently produce explanations that satisfy the required rule-based verification checks. This result is important for RQ3 because gating changes which anomalous sessions are selected for explanation, but it does not change the generation and verification procedure. Therefore, the gating problem can be analyzed as a coverage-cost trade-off under a fixed explanation-generation pipeline. This design separates efficient anomaly screening from more expensive evidence-grounded reasoning, making the framework more suitable for deployment under limited LLM budgets.

RQ2 further shows that retrieval is necessary for evidence-grounded explanation in this framework. Without retrieved evidence, the LLM not only produces less grounded explanations but also attempts to cite evidence identifiers that are not present in the prompt. This indicates that the prompt structure encourages cross-session comparisons, but retrieval is required to provide valid evidence targets for these comparisons. RAG therefore plays a different role than simply adding background context. It constrains explanation generation by linking pattern-match and contrast claims to explicit examples of anomalies and normal contrast evidence.

The gating results offer a deployment-oriented view of explanation cost. Uncertainty gating performs better than Random, Novelty, and Combined strategies across the evaluated budget levels, suggesting that low Screener confidence is a useful signal for selecting sessions that deserve full LLM reasoning. The effective operating budget is also dataset-dependent. HDFS, with a smaller signature

space, continues to benefit from a higher budget, whereas BGL begins to plateau earlier because its anomaly signatures are more diverse. These findings suggest that a practical deployment should tune the reasoning budget according to the expected diversity of anomaly patterns rather than applying a fixed budget across datasets.

Several limitations should be recognized. First, the framework has been evaluated only with one LLM backbone, GPT-5.1, and its effectiveness with smaller or open-source models remains untested. Second, the experiments are conducted on BGL and HDFS, which represent high-performance computing and distributed storage workloads but do not cover microservices, network, or security logs. Third, human evaluation is conducted by a single domain expert across 50 sessions per dataset, so inter-rater agreement cannot be measured. Fourth, the anomaly signature taxonomy is generated by the LLM and normalized using deterministic rules, but it is not evaluated against an external gold-standard taxonomy. Finally, the evidence store remains fixed during evaluation, and the effect of temporal drift in evolving log environments is not assessed in this study.

5. Conclusion

This work demonstrates that log anomaly analysis can move beyond accurate classification toward evidence-grounded explanation while maintaining strong detection performance. On the BGL and HDFS benchmarks, the proposed Screener achieves F1-scores of 0.999 and 0.996, respectively, while operating directly on raw log sessions without log parsing or input truncation. The Reasoner further extends the output of anomaly detection from a binary decision to a structured explanation that satisfies the framework’s rule-based requirements for structural validity and evidence traceability. Because rule-based checks alone do not fully capture semantic correctness, explanation quality is further assessed through human evaluation, with mean scores above 4.6 out of 5 in correctness, completeness, and evidence grounding.

The experimental results also show that reliable explanations depend on retrieval and cost-aware deployment control, rather than detection accuracy alone. With RAG enabled, Grounding Breadth reaches 0.667 on both datasets, whereas it drops to 0.000 without retrieval, and all no-RAG sessions exhibit phantom citations. Faithfulness also improves from 0.590 to 0.714 on BGL and from 0.561 to 0.732 on HDFS when RAG is enabled, indicating that retrieved evidence acts as an important constraint on explanation generation. The gating analysis further shows that the effective operating budget is dataset-dependent. Signature coverage on BGL begins to plateau beyond a budget ratio of 0.20, whereas HDFS continues to benefit up to 0.50. Overall, these findings suggest that the key challenge in log-based anomaly analysis is no longer anomaly screening alone, but generating explanations that are traceable, verifiable, and operationally controllable. Future work will examine open-source LLMs, hybrid retrieval, and online evidence-store updates to improve deployability and robustness in evolving log environments.

Data availability

The BGL and HDFS datasets analyzed in the current study are publicly available in the LogHub repository [36] at <https://github.com/logpai/loghub>. The derived experimental records supporting the findings of this study, including processed session statistics, retrieval outputs, generated explanation outputs, and evaluation records, are available from the corresponding author upon reasonable request.

References

- [1] M. Chen, A.X. Zheng, J. Lloyd, M.I. Jordan, E. Brewer, Failure diagnosis using decision trees, in: International Conference on Autonomic Computing, 2004. Proceedings., 2004: pp. 36–43. <https://doi.org/10.1109/ICAC.2004.1301345>.
- [2] J. Breier, J. Branišová, Anomaly Detection from Log Files Using Data Mining Techniques, in: K.J. Kim (Ed.), Information Science and Applications, Springer, Berlin, Heidelberg, 2015: pp. 449–457. https://doi.org/10.1007/978-3-662-46578-3_53.
- [3] R. Vaarandi, M. Pihelgas, LogCluster - A data clustering and pattern mining algorithm for event logs, in: 2015 11th International Conference on Network and Service Management (CNSM), 2015: pp. 1–7. <https://doi.org/10.1109/CNSM.2015.7367331>.
- [4] M. Du, F. Li, G. Zheng, V. Srikumar, DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, 2017: pp. 1285–1298. <https://doi.org/10.1145/3133956.3134015>.
- [5] S. Lu, X. Wei, Y. Li, L. Wang, Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network, in: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), 2018: pp. 151–158. <https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037>.
- [6] H. Guo, S. Yuan, X. Wu, LogBERT: Log Anomaly Detection via BERT, in: 2021 International Joint Conference on Neural Networks (IJCNN), 2021: pp. 1–8. <https://doi.org/10.1109/IJCNN52387.2021.9534113>.
- [7] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, Z. Wu, LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection, in: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2023: pp. 273–280. <https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00045>.
- [8] W. Niu, Z. Li, Z. He, A. Wang, B. Li, X. Zhang, FSMFLog: Discovering Anomalous Logs Combining Full Semantic Information and Multifeature Fusion, IEEE Internet of Things Journal 11 (2024) 4442–4453. <https://doi.org/10.1109/JIOT.2023.3300690>.
- [9] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, Q. Tian, LogFormer: a pre-train and tuning pipeline for log anomaly detection, in: Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence, AAAI Press, 2024: pp. 135–143.

<https://doi.org/10.1609/aaai.v38i1.27764>.

- [10] M. Bekkouché, M. Meski, Y. Khodja, S.M. Benslimane, E. Tronci, Log-based anomaly detection using BiLSTM-Autoencoder, in: 2025 International Conference on Networking and Advanced Systems (ICNAS), 2025: pp. 1–6. <https://doi.org/10.1109/ICNAS68168.2025.11298040>.
- [11] W. Li, Y. Wu, W. Huang, F. Zhou, W. Ou, H. Wang, L. Deng, System log anomaly detection based on contrastive learning and retrieval augmented, *Sci Rep* 15 (2025) 38370. <https://doi.org/10.1038/s41598-025-22208-7>.
- [12] Y. Gao, T. Luo, K. Huang, J. Tang, X. Li, LogLAA: an adaptive integrated log anomaly analysis framework, *Cybersecurity* 9 (2026) 141. <https://doi.org/10.1186/s42400-026-00573-8>.
- [13] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, M.R. Lyu, Tools and Benchmarks for Automated Log Parsing, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019: pp. 121–130. <https://doi.org/10.1109/ICSE-SEIP.2019.00021>.
- [14] S. Thaler, V. Menkovski, M. Petkovic, Towards a neural language model for signature extraction from forensic logs, in: 2017 5th International Symposium on Digital Forensic and Security (ISDFS), 2017: pp. 1–6. <https://doi.org/10.1109/ISDFS.2017.7916497>.
- [15] Z.M. Jiang, A.E. Hassan, P. Flora, G. Hamann, Abstracting Execution Logs to Execution Events for Enterprise Applications (Short Paper), in: 2008 The Eighth International Conference on Quality Software, 2008: pp. 181–186. <https://doi.org/10.1109/QSIC.2008.50>.
- [16] P. He, J. Zhu, Z. Zheng, M.R. Lyu, Drain: An Online Log Parsing Approach with Fixed Depth Tree, in: 2017 IEEE International Conference on Web Services (ICWS), 2017: pp. 33–40. <https://doi.org/10.1109/ICWS.2017.13>.
- [17] P. He, J. Zhu, P. Xu, Z. Zheng, M.R. Lyu, A Directed Acyclic Graph Approach to Online Log Parsing, (2018). <https://doi.org/10.48550/arXiv.1806.04356>.
- [18] M. Nagappan, M.A. Vouk, Abstracting log lines to log event types for mining software system logs, in: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010: pp. 114–117. <https://doi.org/10.1109/MSR.2010.5463281>.
- [19] H. Dai, H. Li, C.-S. Chen, W. Shang, T.-H. Chen, Logram: Efficient Log Parsing Using n-Gram Dictionaries, *IEEE Transactions on Software Engineering* 48 (2022) 879–892. <https://doi.org/10.1109/TSE.2020.3007554>.
- [20] M. Du, F. Li, Spell: Streaming Parsing of System Event Logs, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016: pp. 859–864. <https://doi.org/10.1109/ICDM.2016.0103>.
- [21] V.-H. Le, H. Zhang, Log-based Anomaly Detection Without Log Parsing, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021: pp. 492–504. <https://doi.org/10.1109/ASE51524.2021.9678773>.
- [22] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, D. Yang, An empirical study of the impact of log parsers on the performance of log-based anomaly detection, *Empir Software Eng* 28 (2022) 6.

<https://doi.org/10.1007/s10664-022-10214-6>.

- [23] Z.A. Khan, D. Shin, D. Bianculli, L.C. Briand, Impact of log parsing on deep learning-based anomaly detection, *Empir Software Eng* 29 (2024) 139. <https://doi.org/10.1007/s10664-024-10533-w>.
- [24] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, H. Yang, LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis, in: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, Association for Computing Machinery, New York, NY, USA, 2024: pp. 364–365. <https://doi.org/10.1145/3639478.3643108>.
- [25] A. Alansari, H. Luqman, Large language models hallucination: A comprehensive survey, *Computer Science Review* 61 (2026) 100970. <https://doi.org/10.1016/j.cosrev.2026.100970>.
- [26] S. Es, J. James, L. Espinosa Anke, S. Schockaert, RAGAs: Automated Evaluation of Retrieval Augmented Generation, in: N. Aletras, O. De Clercq (Eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Association for Computational Linguistics, St. Julians, Malta, 2024: pp. 150–158. <https://doi.org/10.18653/v1/2024.eacl-demo.16>.
- [27] W. Guan, J. Cao, S. Qian, J. Gao, C. Ouyang, LogLLM: Log-based Anomaly Detection Using Large Language Models, (2025). <https://doi.org/10.48550/arXiv.2411.08561>.
- [28] R. Xiao, W. Li, J. Lu, S. Jin, ContexLog: Non-Parsing Log Anomaly Detection With All Information Preservation and Enhanced Contextual Representation, *IEEE Transactions on Network and Service Management* 21 (2024) 4750–4762. <https://doi.org/10.1109/TNSM.2024.3400283>.
- [29] R. Sennrich, B. Haddow, A. Birch, Neural Machine Translation of Rare Words with Subword Units, (2016). <https://doi.org/10.48550/arXiv.1508.07909>.
- [30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is All you Need, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (accessed June 26, 2025).
- [31] R. Child, S. Gray, A. Radford, I. Sutskever, Generating Long Sequences with Sparse Transformers, (2019). <https://doi.org/10.48550/arXiv.1904.10509>.
- [32] N. Kitaev, Ł. Kaiser, A. Levskaya, Reformer: The Efficient Transformer, (2020). <https://doi.org/10.48550/arXiv.2001.04451>.
- [33] S. Wang, B.Z. Li, M. Khabsa, H. Fang, H. Ma, Linformer: Self-Attention with Linear Complexity, (2020). <https://doi.org/10.48550/arXiv.2006.04768>.
- [34] S. Robertson, H. Zaragoza, The Probabilistic Relevance Framework: BM25 and Beyond, *FNT in Information Retrieval* 3 (2009) 333–389. <https://doi.org/10.1561/15000000019>.
- [35] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih,

- T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020: pp. 9459–9474.
<https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html> (accessed April 30, 2026).
- [36] J. Zhu, S. He, P. He, J. Liu, M.R. Lyu, Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics, in: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023: pp. 355–366. <https://doi.org/10.1109/ISSRE59848.2023.00071>.
- [37] X. Ma, H. Zou, P. He, J. Keung, Y. Li, X. Yu, F. Sarro, On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations, *IEEE Transactions on Software Engineering* 51 (2025) 243–261.
<https://doi.org/10.1109/TSE.2024.3513413>.
- [38] I. Loshchilov, F. Hutter, Decoupled Weight Decay Regularization, (2019).
<https://doi.org/10.48550/arXiv.1711.05101>.
- [39] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, R. Zhou, LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs, in: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macao, China, 2019*: pp. 4739–4745. <https://doi.org/10.24963/ijcai.2019/658>.
- [40] X. Li, P. Chen, L. Jing, Z. He, G. Yu, SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults, in: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020: pp. 92–103.
<https://doi.org/10.1109/ISSRE5003.2020.00018>.
- [41] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, W. Zhang, Semi-supervised Log-based Anomaly Detection via Probabilistic Label Estimation, in: *Proceedings of the 43rd International Conference on Software Engineering*, IEEE Press, Madrid, Spain, 2021: pp. 1448–1460.
<https://doi.org/10.1109/ICSE43902.2021.00130>.